ADAPTIVE FAULT-TOLERANT CONTROL USING REINFORCEMENT LEARNING

By

Ibrahim Ahmed

Dissertation
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in

Electrical Engineering

August 11, 2023

Nashville, Tennessee

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivation

Cyber-physical systems are ubiquitous in the modern world. They can be intricate and diverse as they are prevalent. Such systems may operate with tight time-constants in the order of milliseconds, like unmanned aerial vehicles (UAVs). Or, a relatively sluggish pace on the order of minutes or even hours, like heating, ventilation and air conditioning systems (HVACs). Similarly, systems may be completely analogue, continuous, or hybrid. Or yet, they may be monolothic, or a collection of opaque sub-components interacting with each other. Understandably, designing control for complex systems is complex. A control policy may sometimes be hierarchical, addressing different levels of a problem. Sometimes it necessitates predicting the system's state into the future. And sometimes it depends on learning from a system's behavior in the past.

Moreover, it is essential to acknowledge that the aforementioned systems are subject to dynamic environments characterized by significant operational variations. These variations manifest themselves through natural degradation over the course of their operational lifespan, environmental disturbances, and internal component faults. As a result, adapting the control policy becomes imperative to accommodate these systemic variations and maintain system stability and optimal performance.

Design principles traditionally popular in the controls domain, like Proportional Integral Derivative (PID) control, and Model Predictive Control (MPC), rely on understanding system's dynamics via system identification, and fine-tuning parameters to achieve acceptable performance. Once significant system or environmental changes occur, a controller may need to be re-tuned or redesigned. This can present a substantial design cost in terms of complexity and labor.

As an alternative, data-driven learning approaches address the design-time cost of control systems. Instead, the cost of designing and tuning a controller is offloaded to the operational phase. Data collected during operation is used to train a controller's parameters. That can be done indirectly, via learning a dynamics model of the system and then planning control on it. Or directly, by learning a control policy directly from observed data. In both cases, an objective function determines the quality of the policy and informs when a sufficiently optimaly policy has been found. By relying on data instead of purely theoretical models, real-world idiosyncrasies (noise, higher order external effects) of a particular control task can be modeled, instead of making simplifying assumptions. However, such control approaches, naturally, depend on a data-set representative of the operation of the system, and suffer from the bias-variance trade-off.

Given the susceptibility to faults and disturbances, some control approaches are designed to be fault-tolerant: to continue operation, albeit sub-optimally, under system and environmental changes. However, controller design comes with a cost. The cost of designing a robust controller that functions under uncertainties. Or the cost of designing a fail-safe system with redundancy or replication for failure modes. Or the cost of detecting and identifying a fault, modeling it, and deriving a new control policy. Given time, information and other resource constraints, designing a controller from scratch for the plethora of operating conditions may simply be an intractable problem.

In that case, a data-driven learning approach can seem an attractive recourse for achieving fault-tolerance through adaptive control. As the environment changes, so do the measurements of it, and so does the learned control logic. However, the aforementioned drawbacks of data-driven approaches remain.

Can they be mitigated? Transfer learning (TL) presents a solution. In transfer, the knowledge learned on one control task is used to get a head-start on another related task. A controller for one model of octo-rotor UAV will have some logic applicable to the control of a smaller octo-rotor, for example. Transfer learning operates under the assumption that there is some shared abstraction between the tasks, which may not need to be re-learned, saving time. This is the fundamental motivation behind transfer. Thus, using transfer learning, a data-driven controller may only need to observe fewer data-points had it begun its learning from the learned parameters of another "similar" controller.

However, the resolution of these challenges is not straightforward. The notion of similarity across diverse domains raises questions. When confronted with multiple interconnected control tasks, determining the optimal task for achieving effective knowledge transfer becomes problematic. Is it possible for two control tasks to exhibit apparent similarities while possessing significantly different behaviors? Moreover, even with the application of knowledge transfer instead of relearning from the ground up, there is a possibility that the time constraints for achieving functional control in the new task may not be met.

The optimal policy may not be evident for many control tasks. Sometimes there is no past data of optimal control actions. And sometimes deriving optimal control from scratch is computationally costly due to the curse of dimensionality: each new dimension in state and action variables exponentially scales up the search space for an optimal policy. Reinforcement Learning (RL) is a semi-supervised approach in machine learning. By accumulating step-wise feedback (or "reward" in RL domain) for each control action, a RL controller is able to learn actions with the highest expected cumulative feedback. Exploiting this immediate reward as a heuristic for optimality, RL is able to explore control actions in a more guided fashion. Once learned, such a controller can directly map observed state to the control action. This is unlike PID control, which is reactive and can only slide towards an optimal action over time. It is also unlike MPC, which never learns a mapping and acts by planning ahead using a system model recurrently.

The downside of RL approaches is their dependence on data. They can be sample-inefficient. A RL controller ("agent" in the RL domain) takes exploratory actions on the system to get a feedback signal. The exploration may be guided by the feedback. Thus the form of the feedback/reward function is extremely important towards the control policy learned. For example, a RL controller exclusively incentivized to save energy in a building will simply turn off the HVAC system for the maximum reward. Or, a UAV controller rewarded only for quickly getting to a destination may reach unsafe speeds in flight. It is therefore useful to set constraints on the mechanism of learning so that control does not optimize for indeterminate behaviors.

In view of the above, fault-tolerance via data-driven control, informed by principles of transfer learning and system dynamics, presents a unique symbiosis of data-driven approaches. The overarching motivation for this work is adapting to systemic variations efficiently by exploiting extant knowledge of the change.

By way of demonstrating the generality of this effort, the research will be applied to two classes of test-beds. Heating Ventilation and Air Conditioning systems (HVACs) are designed to modulate temperature and humidity as the weather and occupancy changes. Their dynamics are highly influenced by exogenous atmospheric and building load parameters. Changes in the reference signals take the order of minutes to manifest. And a fail state in operation can cause discomfort or high operational expenditure. For this work, specifically, optimization of a cooling tower of a HVAC will be examined.

Unmanned Aerial Vehicles (UAVs), on the other hand, operate with a much smaller lag. A failure state in the system can cause a crash and outright destruction of the machine and potential injury. Faults and responses to control actions manifest on the order of milliseconds, which require equally fast compensatory response.

Within both classes of test-beds, however, the physics of operation are consistent. That is, there is shared knowledge to be transferred when adapting control to systemic variations. Similarly, there are constraints informed by dynamics which, if violated, will lead to adverse costs. In both cases, the systems have hierarchical components which are controlled concurrently to make the whole system work.

1.2 Challenges

There are substantial challenges with data-driven control in general. Machine learning algorithms are stochastic, sensitive to the choice of hyperparameters and to the training data. The objective of the learning problem is often complex and it is not evident if the learned parameters are good enough (locally optimal) or the best (globally optimal). Learning experiments often require abundant processing power and time to converge to an optimum.

There are challenges with developing high fidelity test-beds. Often,simulation applications present a generic system model. Or, the actual system is not well-documented for an accurate virtual reproduction. Or, test-beds make simplifying assumptions about system dynamics for the sake of speed and explainability.

1.3 Contributions

The end goal of this work is to design machine learning approaches to achieve fast adaption to system changes, motivated from principles of control theory. That is, when a fault or disturbance occurs, a controller learned from a similar control task is used to improve the control performance on the new system, while respecting safety constraints. The controllers learn and adapt to faults via reinforcement learning. The contributions of this work are:

- Study on the generality of transfer for reinforcement learning control by:
  - Adaptation to system changes through environmental disturbances. In this case, environment disturbances are parameters external to the system which affect dynamics. For example, prevalent wind conditions for UAVs.
  - Adaptation to abrupt system faults. Faults are defined as unexpected parametric changes in the system's internal variables. For example, motor failure for UAVs.
- Developing high-fidelity test-beds for adaptive control experiments. While simplified simulations of real-world applications can be used as a proof-of-concept approach, they often miss higher-order physical effects that may become significant. A testbed which optionally allows for such simulations can be used to validate proposed approaches with a higher degree of confidence, and reduce *sim2real* transfer gap for virtually trained controllers being put into operation. Work on this is presented in section 6.
- Mitigating challenges of data-driven control by:
  - Synthesizing data from learned models of the system to make learning sample-efficient. Data collected from a system may be insufficient for updating a policy to be suitably optimal in time. If a system model is developed, a RL controller may be able to supplement the learning process by accumulating feedback of control actions virtually. For systems with slower dynamics specifically, computational parallelism can be used to update the policy concurrently. Some work is done on this topic in section 4.2.
  - Using insights from the distribution of tasks under faults to initialize controllers so they learn faster on a new system. If a RL controller has access to previous experience on related control tasks, it is able to exploit them. If the new disturbed or faulty system is related to a related control problem already solved, the corresponding policy may be reused for this case for a jumpstart improvement. However, determining which past experience is suitable, and how to reuse the

experiences once selected, are open problems. Some approaches are discussed in sections 4.3 and 4.4.

  – Using insights from control theory, and knowledge of faults and disturbances, to transform control policies to mirror how the system changes. For example, in a UAV, if the polarity of control actions reverses such that “forward” and “backward” commands are reversed, the controller has to mirror its action outputs to adapt. However, for policies generated from stochastic iterative learning algorithms, this adaptation will not be trivial. Instead, the small learning rate will cause the control policy to change in small increments. Therefore in section 5.1, a model-based adaptation scheme is proposed which reasons about adaptation in terms of system dynamics.

- Demonstrating utility of this work on complex, hybrid systems. While simpler examples popular in controls literature help motivate an approach, its efficacy under performance constraints is shown by applying it to complex systems. In chapters 7 and 8 a UAV control problem is used to demonstrate the theoretical work presented in preceding sections.

1.4 Organization

The rest of this work is organized as follows. Chapter 2 gives a theoretical overview and a summary of applications of several seminal concepts relevant to this work. These include: transfer learning, reinforcement learning, meta learning, HVAC systems, and UAVs. Chapter 3 breaks down the research problem, and walks through individual research objectives and the proposed approach to addressing the problems identified. In chapter 4, preliminary experiments conducted on machine learning algorithms to mitigate aforementioned challenges are explained. Specifically, solutions for machine learning control, agnostic to the system dynamics, are presented. Chapter 5 explains adaptive model-based control using fault-identification and control systems theory, where adaptation is motivated by knowledge of the change in dynamics. Chapter 6 develops a high-fidelity UAV testbed for validation experiments. Eventually chapters 7 and 8 demonstrate robust and adaptive control under disturbances and faults on a UAV test-bed respectively.

CHAPTER 2

Related Work

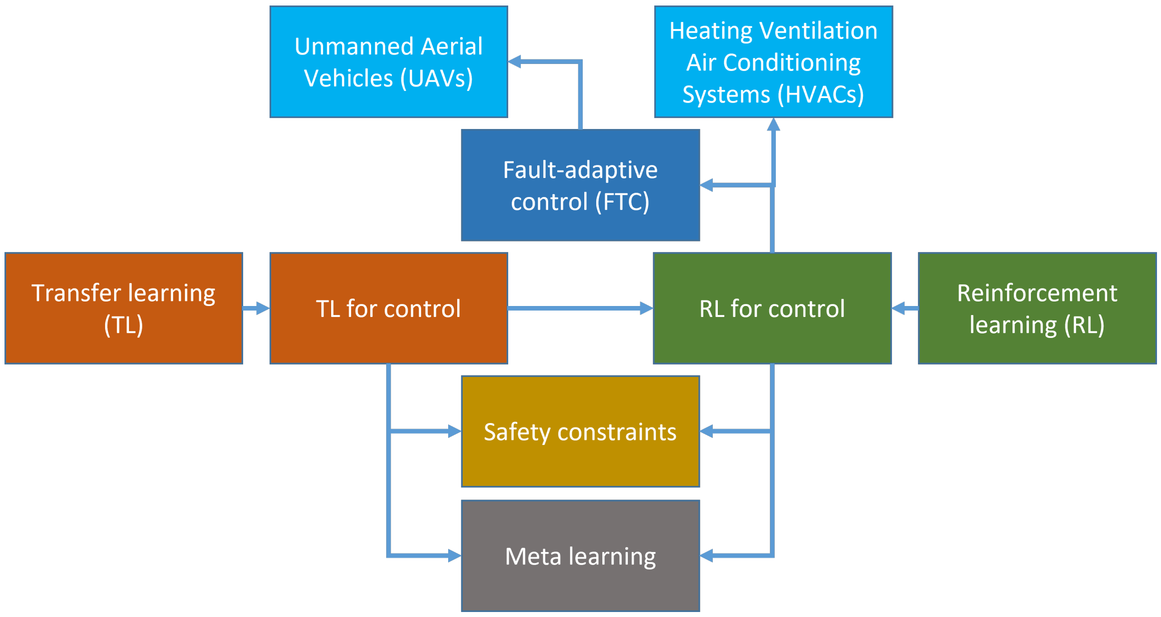

Unmanned Aerial Vehicles (UAVs)
Heating Ventilation Air Conditioning Systems (HVACs)
Fault-adaptive control (FTC)
Transfer learning (TL)
TL for control
RL for control
Reinforcement learning (RL)
Safety constraints
Meta learning

Figure 2.1: Conceptual relationships between background concepts.

2.1 Preliminaries on Transfer learning

Transfer learning is a machine learning technique that enables the reuse of knowledge from a source task to improve learning in a related target task. Transfer learning has been applied to various domains, such as computer vision, natural language processing, and reinforcement learning.

In machine learning, a function $f : X \rightarrow Y$ is learned that maps the feature space X to the desired output (label) space Y. A task $\mathscr{T}$ is then a combination of $\{Y, f\}$, where f is to be learned. The domain $\mathscr{D}$ is the feature space and the marginal distribution of features $\{X, P(X)\}$. During transfer, given a set of source domains and tasks D_s, T_s, and a set of target domains and tasks D_t, T_t, the performance of the functions of the latter is improved using knowledge implied in the former. For this work, the subscripts $*_s, *_t$ refer to source and target tasks respectively.

Zhuang et al. (2020); Lu et al. (2015) review and summarize categorizations of transfer learning techniques in machine learning. Based on the nature of spaces, if the source $\mathscr{D}_s$ and target $\mathscr{D}_t$ domains, and the source Y_s and the target Y_t label spaces are same, it is *heterogeneous* transfer, otherwise it is *homogeneous*. If the domains are the same but the tasks are different, it is *inductive learning*. And if the domains are different but the tasks are the same, it is *transductive learning*.

Based on the nature of knowledge, transfer may be done one of four ways:

Instance based transfer is done by weighing instances and labels in the source task such that the marginal distributional shift between the two domains is minimized. This is akin to recollection-like machine learning algorithms, for example k-Nearest Neighbors.

Feature based transfer done by transforming or learning a shared latent representation between the source and target domain features, and by minimizing the marginal distribution shift between domains in the latent space. It may warrant augmenting the latent representation vectors with features both from the source and target domains. Finally a mapping is learned from the latent representation to the target task's label space Y_t.

Relational transfer is done by reasoning about the logical relationships between features and labels.

Parameter based transfer is done by using the weights and parameters of the learned function f for the source task for the target task. It can be done via adding regularization to the models during training to adapt between domains. Or by either freezing a model's parameters, or constraining them to be similar between source and target tasks. Another way is to construct weak models from different source domains, and selecting an ensemble of models that lead to the highest performance on the target task. Transfer learning in neural networks and deep learning falls into this category.

In more detail, in neural networks and deep learning algorithms, transfer strategies rely on improving the target task performance, and minimizing the distribution shift between source and target domains. The former, trivially, trains the target network to perform well (classification, reconstruction, regression), given the target domain instances. The latter constrains the target network to exploit feature abstractions shared with the source domain, and not re-learn new abstractions. Distribution shift can be quantified by metrics like the Jensen-Shannon divergence and the Kullback-Leibler divergence of the feature distributions between domains. For example, in domain adaptation (Long et al., 2015) in deep transfer learning the distribution shift between domains is minimized by minimizing the intermediate representations after each layer between the parameters of the source and target networks.

Adversarial approaches, inspired by Generative Adversarial Networks (GANs) (Goodfellow et al., 2020), rely on the assumption that in a successful transfer there should be no discrimination in latent representation between source and target domains. In domain-adversarial neural networks (Ganin et al., 2016), the distribution shift loss is instead replaced by an explicit discriminator model. It tries to determine the origin of features (source or target domain), all the while the feature extraction network (the generator component in GANs) competes to generate representations to fool the discriminator. Ultimately those constrained representations are used to train the remainder of the network for the target task.

Related to transfer learning are several fields of study. In *Lifelong learning* (Parisi et al., 2019), a model learns to represent knowledge as new data become available over its lifetime. The concept can be viewed as a super-set of transfer learning. Lifelong learning can be achieved by viewing it as a sequence of transfer tasks, one after another, showing marginal shift in their attributes. Or it can be viewed through the lens of *curriculum learning*, where the learning process is itself supervised by, for example, designing a learning rate schedule. Similarly, *multi-task learning* (Zhang and Yang, 2021; Da Silva and Costa, 2019) views handling multiple tasks concurrently, contrasted with the sequential approach of the previous class of methods.

2.2 Transfer learning for control

In this section, we focus on the application of transfer learning to control systems, which regulate the behavior of dynamic processes using feedback mechanisms. Control systems may need to adapt to changing environments or tasks, which requires continuous learning and updating of the control policies or models.

A control system comprises of the process $P : X \times U \to X$. Given the process state $x \in X$, and the input action $u \in U$, the process transitions to a next state $x \in X$. The action depends on the control policy $\pi : X \to U$. In the context of control, the learning task T comprises of the process being controlled, P and the control policy, π. Transfer occurs by learning the target policy on the target process.

Transfer learning can reduce the data and computational requirements for learning new control tasks, as well as improve the generalization and robustness of the control systems. It can be applied to different aspects of control systems, such as system identification, model predictive control, reinforcement learning control, and anomaly detection.

System identification is the process of building mathematical models of dynamic systems based on input-output data. Transfer learning can be used to improve system identification by transferring knowledge from previously learned models of similar or related systems, or by using pre-trained deep neural networks as universal function approximators.

Model predictive control (MPC) is a control strategy that optimizes the control inputs over a finite horizon based on a prediction model of the system dynamics and constraints. Transfer learning can be used to improve MPC by fine-tuning pre-trained deep neural networks that approximate the MPC solution for a new process automation task, or by transferring knowledge from MPC solutions of similar or related tasks.

Reinforcement learning (RL) control is a control strategy that learns optimal control policies from trial-and-error interactions with the system and a reward function. Transfer learning can be used to improve RL control by transferring knowledge from pre-trained policies or value functions of similar or related tasks, or by using pre-trained deep neural networks as policy or value function approximators.

Anomaly detection is the process of identifying abnormal or malicious behaviors of dynamic systems based on input-output data. Transfer learning can be used to improve anomaly detection by transferring knowledge from pre-trained deep neural networks that have learned features of normal or anomalous behaviors from similar or related systems, or by using pre-trained deep neural networks as anomaly classifiers.

Liu et al. (2020) surveys transfer learning in robotics where knowledge is transferred from a human demonstrator. Helwa and Schoellig (2017) represent the mapping between the outputs of the source and target tasks' dynamics as a transfer function which takes the relevant output/label from the source task and maps it to the output of the target task. It assumes both tasks have the same inputs. In this way data collected from a more convenient source task can be transferred for training to a harder target task. The approach is applied to mapping trajectory measurements between two quadcopters.

Bocsi et al. (2013) employ feature based transfer for learning forward kinematics models of different robots. The source and target domains are projected to their own low-dimensional manifold using Principal Component Analysis. Then a transformation from the source to the target manifold is derived, to align the two spaces. Then dynamics data from the source domain can be transformed and used to learn dynamics model of the target robot.

Chen et al. (2020) use a parameter based transfer approach for learning the control of a HVAC system in a smart building. A neural network model of the system is used. The model trained on a year's worth of data in one location is transferred to another. For transfer, the bulk of the model's parameters are frozen, and the remaining are fine-tuned with fifteen days of data from the new system. The model is then used for planning with model-predictive control and achieves a higher performance than a model trained only on the target building's data.

Similarly, Grubinger et al. (2017) develop a temperature prediction model for a building for model-predictive control. Instead they incorporate multiple source tasks by using feature based transfer to learn a generalized representation space for multiple source tasks. The new prediction model is learned by dynamically weighed sum of the models of the source tasks and the target model. The weights are decayed until the target task's model has converged.

2.3 Preliminaries on Reinforcement learning

Reinforcement Learning (RL) is a class of semi-supervised machine learning methods, where a sequence of optimal interactions is learned with a process. An interaction consists of a measurement of state x, execution of an action u. For learning, instead of an explicit label ($y \in Y$) for the optimal interaction, a feedback/reward signal is observed with each interaction. The reward evaluates the immediate optimality of actions. The RL task is to map states to actions, such that the cumulative reward over a sequence of actions is maximized.

Specifically, a RL controller ("agent", or "policy") interacts with a Markov Decision Process (MDP), M. An MDP is specified by the state space X, action space U, reward function $R : X \times U \times X \to \mathbb{R}$, and the transition probability function $Tr : X \times U \times P(X)$. For deterministic MDPs, $Tr = P$. A MDP satisfies the Markov property, which states that a process is memoryless. The conditional probability of future states of the process depends only on the current states. In other words, the current measurement of state is informative to the policy.

While the reward r is the immediate feedback from an action, returns G are the cumulative feedback from a sequence of actions. And value V of a state is the maximum return, or the maximum cumulative reward from a sequence of actions following an optimal policy. The Q-value Q is the maximum return of an action from a state. Assuming i indexes the time into an episode of interactions:

$$\begin{aligned}
r_i &= R(x_i, u_i, Tr(x_i, u_i)) \\
G(x_i, u_i) &= \sum_{k=t} \gamma^{k-t} \cdot r_i \\
V(x_i) &= \max_{u_i} \left(R(x_i, u_i, x_{i+1}) + \gamma \cdot V(x_{i+1}) \right) \\
Q(x_i, u_i) &= R(x_i, u_i, x_{i+1}) + \gamma \cdot \max_{u_{i+1}} \left(Q(x_{i+1}, u_{i+1}) \right)
\end{aligned} \tag{2.1}$$

The recurrent relationship is known as the Bellman equation (Bellman, 1966). Due to the curse of dimensionality, this dynamic programming problem quickly becomes intractable. Therefore temporal difference methods (Tesauro, 1992), using bootstrapped approximations of values, are used. In that case, lookup tables or function approximations (Melo et al., 2008) are used to represent the most recent estimate of value by only taking a finite sequence of steps. Over time, the approximations converge to the true values V^* and Q^*.

The behavior of a RL controller is called a policy $\pi : X \to U$. A policy may be deterministic or stochastic. The objective of RL is to take actions with the highest value:

$$\begin{aligned}
\pi(x_i) &= \arg\max_u V(x_{i+1}) \\
\pi(x_i) &= \arg\max_u Q(x_i, u)
\end{aligned} \tag{2.2}$$

Since V, Q can be derived from each other, for simplicity, both will be used as V. A policy may be derived by approximating V. This encodes the knowledge of the MDP in the policy function, and is known as model-free RL. Or, a policy may be generated on the fly by planning on the MDP, known as model-based learning. Such approaches require learning a model or having one provisioned already.

In model-free RL, V may be explicitly modeled, where the policy π is simply a max function over it, known as General Policy Iteration (GPI) (Sutton and Barto, 2018; Watkins and Dayan, 1992). Or it can be implicitly modeled in the policy space where π, parametrized by θ, directly learns which action to take, by optimizing for returns G. Learning in policy space allows for policies that operate over more complex, continuous action spaces, because the need to max over the value function every time is bypassed by directly mapping to an action. This can be done through evolutionary and gradient based algorithms, and is known as policy search. The latter relies on the policy gradient theorem (Sutton et al., 2000). The theorem simplifies using gradient ascent, with some step size α, on the expected returns objective with respect to policy parametrizations θ:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_{\theta} \mathbb{E}_{x \sim \pi_{\theta}} G(x) \tag{2.3}$$

A combination of GPI and policy gradient approaches is known as Actor-Critic (Konda and Tsitsiklis, 2000). There, a value function approximation is learned, which is fed the experiences generated by the policy function to evaluate and therefore improve the policy. In an interleaving manner, the value function (critic) and the policy function (actor) are trained using gradient based learning, so valuation is more accurate and therefore the policy is more optimal. Further advancements have proposed deterministic policies (Silver et al., 2014; Haarnoja et al., 2018), and multiple concurrent actors to sample more experiences (Mnih et al., 2016) for efficiency.

Recently, the use of deep neural networks (Mnih et al., 2013) has hastened the use of function approximation approaches on more complex MDPs which warrant equally complex policy functions, needing policy gradients.

The advantage of gradient based approaches is a gradual optimization along the objective function, such that iterations of policies are not drastically different. An issue with gradient based approaches is that they may get stuck in a local optimum (Peters and Schaal, 2008) in the objective surface (if the step size is too small), or they may skip over a solution entirely (if the step size is too large). In that case, a "trust region" in the policy space is needed within which the largest update step may be taken, knowing that an optimum will not be missed. Schulman et al. (2015) proposes just that approach as the Trust Region Policy Optimization (TRPO) algorithm. It tries to guarantee monotonic policy improvement by using second-order gradients of objective to ensure the update step will be large enough in the direction of optimality. However, second order gradients have quadratic complexity in the number of policy parameters, and can therefore be slow and memory intensive. Schulman et al. (2017) proposes Proximal Policy Optimization (PPO). It iterates upon TRPO by foregoing second-order gradients, and approximating the "trust region" by clipping probability ratios of policies between successive iterations, so that the action probabilities in the new policy are not drastically different from the prior iteration.

While advances have been made towards policies able to operate in complex spaces with stable updates and low memory cost, much remains to be done. Dulac-Arnold et al. (2019) describes the challenges of RL in real-world applications. They are derived from the intrinsic properties of RL: the need to explore without prior knowledge of optimal behavior. Challenges have to do with sample inefficiency, safety constraints, partially observable environments, complex spaces, and ill-defined reward functions.

In RL, "how" the policy learned is important (gradient, evolution). But so is "what" is used to update gradients. In *off-policy* updates, the agent uses a different exploratory policy to collect experiences to calculate returns to optimize. In contrast, *on-policy* updates use a single policy both for exploration and exploitation. The former have a smaller variance, due to a stable exploratory policy, but a high bias because the data do not fully reflect the distribution of experiences being seen at operation. On-policy updates have a smaller bias but a larger variance because the policy constantly is being updated.

Along with the "what", the "when" also becomes important. In offline RL, the policy is updated in periods when the agent is not interacting with the environment. This may be done with the help of a digital twin model of the process to interact with. In online RL, the experiences are sampled by interacting with the actual environment itself. The former lends some benefits with regards to sample efficiency, but comes with the design cost of system modeling.

2.4 Transfer in reinforcement learning

While transfer learning, generally, has availed itself to supervised learning problems, the RL formulation sets out to solve decision tasks where optimally labelled actions are not available for training. Reinforcement-learning is semi-supervised. The learning problem is to optimize a sequence of decisions/actions on a process in the long run. Instead of lthe optimal decisions/actions for the target process, there are reward signals that evaluate the the current decision of the learned policy.

Formally, in RL, the domain D of a task is the set of observations of states, actions, and rewards sampled from a Markov Decision Process (MDP) $D = \{X, U, R, M\}$. The transfer learning task $T = \{M_t, \pi_t\}$ is to learn a policy from a set of source domains $\mathscr{D}_s : \{D \ldots\}$ to a target domain.

Like transfer learning in general, the holistic goal of transfer RL is to exploit abstractions between tasks to increase performance. There are related approaches which rely on different abstractions. For example, imitation learning (Hussein et al., 2017) seeks to abstract away supervised behavior from an expert RL agent on the same environment. Hierarchical RL (Al-Emran, 2015) abstracts the RL problem into sub-tasks (high-level direction set-point vs. low-level actuator control to get there, in a robot). And multi-agent RL (Zhang et al., 2021) abstracts a task such that multiple agents cooperatively or competitively interact with the process. Recent demonstrations in multi-player video games have illustrated this approach (Shao et al., 2019). In lifelong learning (Abel et al., 2018), the agent sequentially learns multiple tasks as they evolve into each other, with the abstraction being the transition between tasks. Alternatively, Da Silva and Costa (2019) looks at transfer learning between agents participating in a multi-agent environment. Using multi-task learning methods, Cabi et al. (2017) introduces an agent trained for solving multiple control tasks from the outset.

Taylor and Stone (2009) and Zhu et al. (2020) survey transfer learning in deep reinforcement learning domains. The mechanisms of transfer described in section 2.1 are further conditioned by the MDP and the algorithm used to train the policy π. For example the nature of knowledge that is transferred: value functions, a set of experiences from an optimal policy, a sub-optimal “teacher” policy? The similarities between the source and target tasks: whether they share the same state and action space, whether the reward function incentivizes similar behaviors, whether there are shared state transition dynamics? What algorithms are viable for the transfer approach: discrete policies, model-based, asynchronous agents? What is the sample efficiency requirement of the transfer problem?

Prior work, has proposed measures for quantifying performance of transfer in reinforcement learning. Some of these are:

Jumpstart: The relative change in initial performance on the target task.

Asymptotic: The relative change in limiting performance when learning is considered finished.

Transfer ratio: Fraction of the reward accumulated by an agent had it used transfer learning, versus had it not.

Time to threshold: Interval needed to reached a specific performance level.

Performance sensitivity: Variance in returns during learning.

Necessary knowledge: The amount of experiences, policies, source tasks needed to achieve predetermined performance.

Necessary quality: The expertise/specificity represented by expert policies, experiences from source tasks to achieve predetermined performance.

The following sections describe the array of ways to conduct transfer in RL.

2.4.1 Reward shaping

In reward shaping, a surrogate reward function informs the nature of exploration for improved transfer. Given the environmental reward function R, the reward shaping function R' evaluates states and actions on their value towards the transfer itself, and not the objective of the environment. This means that the learned policy will, in fact, vary from the theoretically optimal policy for the target task. Furthermore, a malformed R' may cause the policy to go into a state loop to reap infinite rewards.

Potential-based reward shaping (Ng et al., 1999) approaches step over this problem, by defining a potential function to evaluate a state $\phi(x)$, or a state and action pair $\phi(x,u)$. The reward shaping function is then the potential difference between two states or state-action pairs: $R'(x,u,x') = R(x,u,x') + \phi(x) - \phi(x')$. The potential, being a scalar field over the state and action space, is conservative. Therefore looping back to the same state yields zero shaping rewards. Furthermore, the state-action value of the MDP with the original reward can simply be obtained by adding the state potential to the state-action value of the shaped MDP. Subsequent works have developed strategies for shaping the reward shaping functions themselves over time.

2.4.2 Learning from demonstrations

In Learning from demonstrations (LfD) (Schaal et al., 1997; Argall et al., 2009; Ravichandar et al., 2020), an "expert" domain D_e is used to train on the target task. The policy may be optimal or sub-optimal and only used as a starting aid. D_e may be represented by a buffer B_e of expert experiences as the expert policy interacts with M_t. The experiences may be used to learn attributes of the target task. This means learning the policy π_t directly, or implicitly by learning the value function V. B_e may also be used to learn environment dynamics Tr_t for planning and model-based approaches.

Of note is research done in Brys et al. (2015a), where a LfD approach addresses the quality of demonstrations for transfer. This is related to the need of evaluating a worthwhile transfer source for adaptive control in this work. The demonstrations are used to make a potential function $\phi(x)$ to shape rewards. This biases exploration, such that sparse rewards in M_t are collected faster. A similarity metric is used for state-action experienced in M_t and sampled from B_e. The the potential of a state is then:

$$\phi(x,u) = \begin{cases} \max_{x_e,u_e \sim B_e} \texttt{GaussianKernel}(x, x_e) & u_e = u \\ 0 & u_e \neq u \end{cases} \tag{2.4}$$

This approach assumes that the same action taken from a state similar to the one encountered in expert experiences can lead to higher returns. Experiments using Q-Learning show that sub-optimal policies are able to produce higher quality results more sample-efficiently. The experiment assumes that the expert and targed MDPs are same.

2.4.3 Policy transfer

Policy transfer approaches mimic the parameter and instance based categories of transfer mechanisms described in section 2.1. In policy transfer, the the parameters of expert policies from source tasks, or the trajectories of experiences generated by such policies are used to train a policy on the target task.

In *policy distillation* (Rusu et al., 2016; Czarnecki et al., 2019) the target policy parameters are learned by minimizing the marginal distribution of action probabilities between the source policies and the target policy, given trajectories of experiences sampled from either of them.

In *policy reuse* (Fernández and Veloso, 2006; Fernández et al., 2010), a set of expert policies from source tasks are available. During transfer, from the super-set of source policies and the target policy one is sampled using Maxwell-Boltzmann statistics. Here the "energy" corresponds to the expected performance improvement on the target task (i.e. returns), and the "temperature" is a noise parameter that is increased over time to sample policies more liberally. The sampled policy is then used to collect experiences on the target MDP for improvement.

Rosman et al. (2016) formalises the problem of policy reuse and presents an algorithm for efficiently responding to a novel task instance by reusing a policy from a library of existing policies, where the choice is based on observed 'signals' which correlate to policy performance.

2.4.4 Representation transfer

Representation transfer (Taylor and Stone, 2007) approaches are applicable when the source and target domains and tasks differ on their representations of states, actions, learning algorithms, and values. For example, representation transfer will use the knowledge from a source policy learned using Q-Learning algorithm to transfer to a target policy to be learned using policy gradient. In several cases, experiences recorded $B_s : \{(x,u,x',r)\}$ using the source representation are used to train the target representation. For example, for e.g. transferring to from a value function-based algorithm $\pi_s(x) = \arg\max_u Q_s(x,u)$ to a policy function algorithm $\pi_t(x) = f_\theta(x)$, the new function can simply be learned in a supervised manner.

Such approaches can be used to learn a task using simpler representation models, and then once a rudimentary performance is achieved, transferring them to a complex representation for higher performance.

2.4.5 Inter-task mapping

Inter-task mapping (Fachantidis et al., 2011) methods run parallel to feature-based transfer. It is assumed that there is a correspondence between the source and target domains and tasks, and a one-to-one mapping can be learned between states, actions, and rewards. For example, $\chi : X_s \rightarrow X_t$.

Works in the past have used a combination of transfer mechanisms with inter-task mapping. For example, Barrett et al. (2010) learn an inter-task mapping of state-action features for transferring RL-based control. They then reuse the source state-action values Q_s, by adding them to the update of Q_t of the target task. This acts like a form of reward shaping. Tested on a robotic manipulation task, transfer from sources leads to higher rewards compared to learning from scratch.

2.5 Safety constrained transfer in reinforcement learning

Safety in reinforcement learning broadly adheres to the following concepts (Garcıa and Fernández, 2015):

- Speeding up convergence of the optimization problem, either by identifying the environment model or by training the policy fast. This reduces the time for uncertain, exploratory actions, which may be unsafe.
- Modeling the bounds of safe states and actions $X_{t,safe}, U_{t,safe}$ in the target task. Trivially, this requires sufficient system knowledge. But for dynamic, semi-supervised modeling, an exploratory phase and a definition of safety constraints is needed.
- Modeling uncertainties, whether in the environment or the learned policy. Once done, the bounds of outcomes as a result of actions can be determined.

Alternatively, Xu et al. (2021) characterize safety approaches along two lines:

- *Primal-dual* approaches introduce a safety objective in the RL problem which is then optimized using the standard gradient descent methods. However, they introduce complexity in the choice of weights of the safety objective.
- *Primal* approaches instead add constraints on the MDP or the learning algorithm itself, and leave the RL objective untouched.

The bulk of research in safety-constrained RL does not look at transfer, or safety in multi-task or continual learning problems, and instead focuses on a single target task.

Yu et al. (2019) Formulates the non-convex policy optimization problem as piece-wise convex optimization problems so convergence is faster to achieve. Fulton and Platzer (2018) uses model verification methods to do safe learning. When model bounds are violated, the agent prioritizes return to the modeled state space. Similarly Cheng et al. (2019) uses a combination of model-free and model-based controllers and learning the system model online for safe control.

Wachi and Sui (2020) use constrained exploration, where an agent first learns safe bounds in a subset of state space, then optimizes for rewards by exploring in the bounds. It repeats for different regions. Similarly, Junges et al. (2016) discuss safety-constrained RL by iteratively learning sequences of safe exploration strategies instead, and then interleaving RL to optimize for the performance objective. Thananjeyan et al. (2021) also uses a two step, two pronged approach. The first step uses past data to learn about constraint violations. Learning is done for two tasks: one prioritizing constraint violation recovery, and another for task performance improvement. Gros et al. (2020) similarly achieves safe RL by identifying bounds. A safe set of actions for states is first learned by satisfying a safety constraint. The policy learned for optimizing the RL objective is then projected to the safe-set by finding the action in the set that is closest to one recommended by the optimal policy. However, this may not work where euclidean projections are meaningless between actions.

Instead of explicitly delineating safe-sets, constraints can be added to the MDP and the learning algorithm itself to prevent unsafe behaviors. Fernandez-Gauna et al. (2013) introduces the concept of Partially Constrained Models (PCM) for training a reinforcement learning policy on a target task. This builds upon Constrained MDPs where a risk term is included in the reward (Geibel, 2001). A PCM is a simplified version of the target task, and acts as the source. Its domain is a subset of the target domain but with the same action/label space. In a PCM, some safety constraints are removed and an optimal source policy is trained. However, a record is kept for state-action pairs leading to a restricted state. That decision function is predetermined and domain-specific. The source policy is then reused on the target MDP but unsafe actions are vetoed as per the record. The approach is tested on a discrete state and action space, leading to substantially higher successes compared to tabular Q-learning from scratch on the target task. Similarly, Chow et al. (2018) uses Lyapunov based constrained MDPs for safety constraint guarantees.

Alternatively, Wen and Topcu (2020) model the space of safe policies. They introduce a safety cost of sampled trajectories as a surrogate objective along with the expected return. The algorithm generates a population of policies from a parametrized distribution. The policies are run on the MDP to get returns. The policy distribution is optimized such that the sample of policies generated yield higher returns and lower safety costs. However, it is relatively sample inefficient, by virtue of having to run multiple policies for each update to the policy generator.

In a yet another approach, Li and Bastani (2020) uses shielding, where a function evaluates the risk of a policy. A backup non-linear model-predictive controller is used in case the nominal policy is deemed unsafe.

The safety of the policy is evaluated dynamically. Along the same lines, Wang et al. (2020a) uses a hierarchical approach. Low-level polynomial controllers are trained for safe-sets of states where they are guaranteed to have safe operation. Then a reinforcement learning agent is trained to switch between controllers to adapt to changing states.

Lütjens et al. (2019) Instead models neural network prediction uncertainty to provide bounds for safe model-based RL-based control. They test on a collision avoidance system with pedestrians. Chen et al. (2021) Uses latent variable representation of environment dynamics to detect mode change. Uses the latent representation to predict future trajectories with uncertainties, and uses model predictive control for action derivation.

Garcia and Fernández (2012) propose the Policy Improvement Through Safe Reinforcement Learning algorithm (PI-SRL). They *reuse* a sub-optimal policy to approximate baseline behavior with behavioral cloning using case-based reasoning. Following that, exploration is done by adding Gaussian noise as a risk tolerance parameter to actions of the baseline policy, and increasing it as the process matures. The parameter is manually scheduled. Finally, safety is explored in the context of "known" and "unknown" states, based on Euclidean distance with states already seen in the baseline policy. However, this assumes that states local in Euclidean space have similar optimal actions and values. García and Fernández (2019) build upon the last work by using probabilistic policy reuse. The policy is trained to model its own risk tolerance, which increases monotonically, such that it can relax the noise bounds for baseline actions in "unknown" states when it is confident in its evaluation of actions.

2.6 Task similarity in transfer learning

An open question in transfer learning is picking a source to transfer from. A bulk of prior research on similarity measurement between transfer tasks has focused on image/classification tasks, which have generally been under the supervised learning domain.

Carroll and Seppi (2005) Outlines the principles of task similarity for transfer learning. Broadly the level of similarity between tasks is the advantage gained when transferring from source to target task. Because, ultimately, that is the goal of transfer learning. A similarity measure need not be a distance metric (i.e. satisfies the triangle inequality). They present three criteria for a task similarity measure:

1. It approximates learning improvement using source task to learn target task via transfer.

2. That a higher similarity leads to a higher transfer performance.

3. The measure can be computed before running a transfer learning trial.

Shui et al. (2019) analyzes utility of H-divergence and Wasserstein distance in adversarial multi-task learning. Shows that explicitly including task similarity penalty in multi-task optimization leads to superior performance on a class of image classification tasks.

In (Fernandes and Cardoso, 2019), a generalized form of hypothesis transfer learning. This is a parameter-driven approach, where the model weights are modified. Opposed to a data-driven approach where source instances or feature representations are memorized for use in target task. In the objective function, a regularization term is added to reflect the similarity between the source and target task models. The approach is demonstrated on three categories of transfer learning. Sparse transfer, where the tasks are nominally different. Partial transfer, where the source task is partly observable, And cross-model transfer, where the underlying model architecture is entirely different. Distance between model weights is used as a similarity measure. In a majority of cases, classification and regression tasks saw a better performance over standard literature baselines like SVMs and L_1 or L_2 regression.

Dwivedi and Roig (2019) proposes the use of Representation Similarity Analysis for measuring task similarity for transfer. RSA is calculated as a correlation of the neural layer activations of two models over a batch of inputs. When tested on an image classification transfer task, encoder layer activations provide adequate measures of similarities such that the transfer performances between similar tasks are higher. While it requires networks trained on both tasks to quantify similarity, smaller models can be preemptively generated for a similarity estimate before effort is invested in deep transfer learning.

Lu et al. (2017) employs a binary sparse reconstruction classifier to train a model to predict a new category from the same domain of input samples. Assuming the domain is a subspace, a test sample may be reconstructed from a linear combination of the training samples. A sparse representation is obtained by regularizing the weight for each training sample so the fewest samples to represent the test instance are found. The category of instances with the smallest reconstruction residual can then be used as initialization to learn the classifier for the new test category.

Zhang and Yeung (2010) proposes multi-task relationship learning (MTRL), where a model learns multiple tasks concurrently. The method assumes each task's solution i is modeled by weights θ_i. The prior probabilities of weights are modeled as a matrix-variate normal distribution, where the covariance matrix models the relationships between tasks.

Some work has been done with regards to quantifying task similarities in RL. Tasks in RL can be dissimilar by dynamics, state space, or reward function. Ammar et al. (2014) Uses restricted Boltzmann machines to represent Markov Decision Processes, and then uses the reconstruction error between different MDPs as a representation of task similarity. They show that the performance improvement is correlated with the similarity measure. Task differences can be on the state domain, reward function, or system dynamics. Sometimes tasks that are apparently dissimilar (mountain car, cart pole) can have a viable transfer. Tabular q-learning is used on simple RL tasks to show transfer performance increases with more similarity. MDPs with similar parametrization cluster together using this measure.

2.7 Preliminaries on Meta learning

Meta-learning, or learning to learn (Hospedales et al., 2021), is another paradigm for mitigating possible sample inefficiency of machine learning algorithms. Both transfer- and meta- learning depend on observing multiple tasks to improve on a target task. Transfer learning exploits the knowledge learned from those tasks to improve on the target task. Meta-learning instead exploits the knowledge of how the learning algorithm itself performed on a task, to improve the algorithm, which in turn will eventually improve on the task.

In machine learning, a task T constitutes of the labels Y, and a decision/prediction function being learned $f_\theta : X \to Y$, with parameters θ. The function operates on the domain D of inputs X with a prior distribution $P(X), D : \{X, P(X)\}$.

In meta-learning there are two objectives being optimized. The "inner" or "base" objective $L : T \to \mathbb{R}$ finds optimal parameters θ for a single task given the learning algorithm's parameters ω (for e.g. learning rate). The "outer" or "meta" objective optimizes the learning algorithm's parameterization ω itself. The goal is to learn the best parametrization ω of the algorithm, given a set of tasks $\mathscr{T}$, where each task T comprises of

$$\begin{aligned} \theta^* &= \arg\min_{\theta} L(D, T \mid \omega) \\ \omega^* &= \arg\min_{\omega} \mathbb{E}_{D,T \sim \mathscr{D},\mathscr{T}} \left(L(D, T)\right) \end{aligned} \tag{2.5}$$

In the context of maximizing the likelihood of parameters, given the evidence:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} \log P(\theta \mid D, T, \omega) \\ \omega^* &= \arg\max_{\omega} \log P(\omega \mid \mathscr{D}, \mathscr{T}) \end{aligned} \tag{2.6}$$

Hierarchically, meta-learning algorithms typically proceed in two steps:

- In the inner loop, given the algorithm parameters ω, the base objective L is optimized to learn θ for a single task.

- In the outer loop, the effect of ω on L for different tasks is used to optimize algorithm parameters for the next iteration.

Depending on the nature of the task, the nexted optimization problems can be solved with a range of methods, including gradient descent, evolutionary search, and reinforcement learning.

For a broad class of neural networks, Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017a) is a suitable meta-learning algorithm. It meta-optimizes the initialization of the function parameters θ, such that they are likelier to adapt to a new task fast. Assuming a differentiable loss function L and a set of tasks with their domains $M \in \mathscr{M}, T \in \mathscr{T}$, it follows the same structure:

1. In the inner loop, for each T_i, D_i, set $\theta_i \leftarrow \theta_i + \nabla_\theta L(D_i, T_i \mid \theta)$
2. In the outer loop, update $\theta \leftarrow \theta + \mathbb{E}_i(\nabla_\theta L(D_i, T_i \mid \theta_i))$.

The second meta-update step introduces second-order gradients $\frac{\partial L(D_i,T_i|\theta)^2}{\partial \theta^2}$ which are computationally expensive. Instead first order approximations (Nichol et al., 2018) are able to achieve comparable performance. MAML is sensitive to hyperparameters, and work has been done towards investigating stable performance Antoniou et al. (2018a).

2.8 Meta reinforcement learning

Many approaches in meta-learning are applicable to RL as well, and some meta-RL approaches are unique in themselves. Broadly, meta-RL approaches have learned different parts of the RL algorithm: loss functions, state representations, transition dynamics

Pritzel et al. (2017) uses a dictionary to store mappings between state features and values. State features are generated from a neural network, and the expected values per action are stored in a dictionary. Using a shared dictionary with features speeds up accurate recall across similar tasks, and rapidly incorporates recent experience by explicitly modeling each state action pair value. However this works with discrete action spaces only.

Gupta et al. (2018) supplements MAML for meta-reinforcement learning. The policy function takes the state and a random latent variable generated from a parametrized distribution. During training, in the inner loop, for each learning task, the policy and distribution parameters are updated to optimize returns. In the outer loop, along with the returns objective, a penalty for the difference between the latent variable and its prior distribution is added. Thus, the learned policy is conditioned to optimize returns when the latent random variable follows its prior distribution, as is the case when a new learning task is encountered.

Xu et al. (2018) introduces a variation of MAML which meta-learns the return function parameters instead of the initialization parameters θ. Namely, the discount factor γ and the look-ahead horizon for calculating returns. In the inner loop, RL is done by calculating returns using sampled return parameters. And in the outer loop, the loss gradients with respect to the value function parameters are used to tune them.

Houthooft et al. (2018) introduce Evolved Policy Gradients, a meta-learning approach for gradient-based reinforcement learning. This learns the loss function itself. The loss is parametrized as a function of a history of experienced states and actions. The outer loop optimizes the loss function parameters using evolutionary strategies. The inner loop optimizes the loss for a specific learning task. The algorithm can generalize over different tasks such as where the goals are reversed. Adaption is good for a family of tasks but not for different classes entirely. This approach can be combined with MAML to yield marginally better results.

Sæmundsson et al. (2018) proposed model-based meta-learning. A meta-learned Gaussian process is used to model system dynamics, by learning a latent vector representation of different tasks sampled from the population. Durig testing, the latent representation is inferred for a new task. Model-predictive planning is then used to output control sequences. This approach reduces number of steps needed to adapt to a new task.

Clavera et al. (2019) develop online model-based reinforcement learning. A dynamics model is trained to predict a horizon of future steps, given a history of recent experiences. The model parameters are modified by a meta-function that operates over recent experiences and the extant model parametrization. In the outer loop, the iterated model is then used in planning control actions. The resulting loss modifies the meta-function. System dynamics reflected in recent state distributions are quickly modeled and planning is done in the order of seconds.

Wang et al. (2016) use a recurrent network to sequentially train across instances from a Markov Decision Process sampled from a parametrized population $\mathscr{M}$. The expectation is that the variation in state trajectories will condition the recurrent connections to identify changed dynamics of a new task and to quickly adapt.

Interestingly, Fakoor et al. (2020) show that off-policy reinforcement learning can maintain parity with meta-learning approaches like MAML. Using importance sampling over observations of prior tasks, and a context variable identifying a task that conditions the policy and value functions, a policy for a test task is derived efficiently over new samples.

2.9 Preliminaries on fault adaptive control

Fault-adaptive and fault-tolerant control (FTC) refers to a set of techniques developed to increase system availability and reduce the risk of safety hazards. Its aim is to prevent simple faults from developing into serious failures. Fault-tolerant control systems can be classified into passive and active. Active fault-tolerant control systems are mainly composed of two integrated processes: a Fault Detection and Isolation (FDI) scheme which monitors the performance of a system, detects the occurrence of a fault, and isolates the faulty component. Passive fault-tolerant control is designed to stand strong against several faults without needing fault detection and isolation or controller reconfiguration.

The architecture of FTC systems is surveyed in Blanke et al. (1997). FTC detects faulty conditions, accommodates faults during system operation, and slows down the progress to complete system failures. This is differentiated from fail-safe systems, where redundancies prevent failure conditions from occurring in the first place. Thus FTC approaches require comparatively fewer hardware add-ons to the system as the focus is on mitigating faults after the fact by adapting the controller to the changing conditions in the system.

Patton (1997) surveys constituent research areas of FTC. Fault-tolerant supervision has been explored in several fields including probabilistic reasoning, fuzzy logic, and genetic algorithms. With recent advances in computing and communication systems, the fault-tolerant control problem is being applied to a wide range of industrial and transportation applications, primarily due to increased safety and reliability demands beyond what a conventional control system can offer Yen and Ho (2003); Zhang and Jiang (2008). These applications include aerospace, nuclear power, automotive, manufacturing and other process industries, and fault tolerance is no longer limited to high-end systems. Consumer products, such as automobiles, increasingly dependent on microelectronic and mechatronic systems, on-board communication networks and software, are pushing the frontiers of fault tolerance.

There is ample literature on model-based FTC with degrading faults. Noura et al. (2000) discusses FTC under two regimes: sensor faults and actuator faults. In the former, the controller does not need to respond as the fault only lies with perception of state. Whereas in the later, a change in control law is required. In both cases accurate fault detection and diagnosis is required.

In (Zhang and Jiang, 2003; Jiang and Zhang, 2006), performance degraded reference models are used to achieve fault tolerant control. Once a fault is diagnosed, a reference model matching the fault signature is employed for control. This can be applied to progressively degrading systems by generating reference models for various levels of degradation. This, however, requires *a priori* knowledge of possible faults and an ability to generate accurate system models.

Adaptive control is a control method where a controller must adapt to a controlled system with parameters that vary or are initially uncertain. For example, as an aircraft flies, its mass will slowly decrease as a result of fuel consumption; a control law is needed that adapts itself to such changing conditions. Therefore in case of parametric faults, adaptive and fault-tolerant control will see substantial overlap. One similarity between meta-learning and adaptive control is that both involve learning and adapting to changing conditions. In adaptive control, the controller adapts to changing system parameters, while in meta-learning, the learned model adapts to new data more efficiently by using prior experience.

2.10 Reinforcement learning for fault-adaptive control

RL-based adaptive control is an active area of research. It has two inherent advantages: (1) A direct mapping from states to actions can be learned, for policy search algorithms, instead of planning or sliding to an optimal action. And (2) being data-driven, a controller that learns online can naturally adapt to maximize feedback representing distributional shifts in the states due to system changes. However, the cost is the stochasticity and sample inefficiency, inherent to these approaches as well.

RL-based adaptive control has been applied to robotics (Kober et al. (2013)), board games (Silver et al. (2018)), and smart buildings (Naug et al. (2019)) among other things. Lewis et al. (2012) discusses application of RL to designing optimal and adaptive controllers. Adaptive fault-tolerant controllers develop and learn from a system model on the fly, but at the cost of true optimality in control. Liu et al. (2016) uses neural networks to approximate tracking error. Their focus is on making the training of the networks faster so the system model can be updated regularly in response to faults to generate a control policy. Both incipient and abrupt faults are demonstrated on a non-linear system. Zhao et al. (2017) develops an RL-based fault-tolerant controller for actuator faults. Instead of rewards and values, costs and objective functions are used. The objective function is a cumulative sum of the actuator fault and the cost of the control action over time. The goal is to minimize the objective function over actions.

2.11 Adaptive control in UAVs

Unmanned Aerial Vehicles (UAVs) are highly dyanamical systems, with small time constants and fast moving parts that are susceptible to internal and external disturbances. Prior work on adaptive control in UAVs has covered classical and data-driven approaches in some detail.

Zulu et al. (2014) reviews linear, non-linear and other control approaches for autonomy in quadrotors.

The most prevalent control approaches are PID control. The advantages are simpler controller construction, but which comes at the cost of non-linearity in the system and unmodelled disturbances. Work has been done to make control adaptive by tuning PID gains when a fault is detected (Sadeghzadeh et al., 2012). Somewhat similarly, (Pounds et al., 2012) analyze effects of abrupt changes in a UAV's load and the bounds in which PID control adapts, and find that for a range of loads, PID control is capable of adapting to disturbances. (Marks et al., 2012) uses cascaded PID controllers on an octo-rotor to reallocate control to redundant rotors in case of multiple actuator failures.

Linear Quadratic Regualizer control (LQR) minimizes a quadratic cost criterion. In literature it has been used for optimal trajectory planning (Cowling et al., 2006; Suicmez and Kutay, 2014) and dynamics (Reyes-Valeria et al., 2013). A LQR controller does not necessarily need the entire state information, when combined with a Kalman filter, given Gaussian noise. This is a useful application in case of partial observability due to faults.

Model-Predictive Control uses a dynamics model of the system to detect faults and plan adaptive control. Such approaches have been occasionally applied (Yu et al., 2015). Similarly, in Saied et al. (2015) FTC of co-axial octo-rotor is presented. It uses residual based fault detection, then exploits built-in co-axial motor redundancy to accommodate faults by increasing motor speeds.

Alwi and Edwards (2013); Zeghlache et al. (2018) demonstrate the utility of variants of sliding mode control for octor-rotor faults.Razmi and Afshinfar (2019) demonstrates sliding mode quadrotor attitude control with a neural network to learn sliding mode parameters. Hamadi et al. (2020) use a self-tuning sliding mode controller to achieve fault-tolerance after detection has been made.

Generally, however, classical FTC schemes using classical control rely on fault detection and isolation to achieve their goals (Chamseddine et al., 2015; Park et al., 2013). Recently, data-driven and reinforcement learning approaches have seen adoption in this domain as well. Using the inherent adaptive and semi-supervised advantages of RL, and to mitigate the sample inefficiency drawbacks, they are usually employed in conjunction with higher- or lower- level controllers.

For example, Razmi and Afshinfar (2019) uses a neural network to tune parameters of a sliding mode controller for a UAV. The neural network helps address the drawbacks of sliding mode such as susceptibility to chattering and measurement noise.

RL approaches are inherently adaptive, as they learn from the shifting data reflecting parameter changes in the system/MDP, but the exploration period to adapt may be unsafe or cause failure states. Li and Xu (2020) employ General policy Iteration and an inverse dynamics approach to learn value functions and then take greedy actions. The sample inefficiency and stochastic exploration of RL is mitigated by constraining the state space to follow only certain arcs of trajectories during eploration.

In Fei et al. (2020), a RL-based quadrotor FTC is developed under actuator and sensor faults (cyber attacks). The RL controller runs concurrently with a cascaded PID controller and adds control signals to the PID controller's outputs. The RL controller is trained in simulation on faults, and is able to compensate for sub-optimal PID actions when such faults occur in the real system. It does not need a fault detection and isolation state. This also demonstrates a RL policy trained on a model can be transferred to actual vehicle for fault recovery.

Faults in a system by lead abrupt reconfiguration which may not be modeled by a controller. Pi et al. (2021) develop a low-level UAV control using RL which is system agnostic (quad- and hexa- rotor platforms). The learned policy outputs translational and rotational accelerations needed, which are then separately allocated to rotors via a control allocation stage. Their approach demonstrates the utility of machine learning approaches in abstracting knowledge between tasks for fast transfer when needed.

Another approach combines neural networks with a physics-based model (Shi et al., 2019). The physics based model learns the lower order dynamics, whereas the network can model higher effects like noise. This is then used for more accurate model-based control derivation.

All in all, data-driven approaches, by implicitly being able to learn a representation of the task and relating it to the output space, are able to adapt to system changes. The high complexity also allows them to model minute effects which otherwise may be ignored from a purely principal-driven approach.

2.12 Adaptive control in HVACs

Optimal control of HVAC systems has been extensively addressed in research work. Wang and Ma (2008) surveys the landscape of control approaches in HVAC systems and categorizes them into local and optimal control. Local control is a rudimentary class of approaches where a system operates based on a rule-set or tracking error with reference signals. Examples include proportional-integral-derivative (PID) control or simple thresholded on/off control. Optimal control seeks to minimize a cost function with respect to overall system performance and controllable variables. The cost function can be based on a physics or data-driven model and then minimized.

The survey Wang and Ma (2008) further documents optimization algorithms used in optimal control. Linear approaches include least squares and its variants. Non-linear optimization is divided into local and global approaches. In local optimization, successive solutions are in each other's vicinity. This includes gradient-free (simplex) and gradient-based approaches (gradient descent, Newton's method, Lagrange multipliers). Global optimization explores solutions all over the domain of the cost function. This includes simulated annealing and evolutionary algorithms.

Model-predictive control (MPC) for HVAC systems is explored in depth by Afram and Janabi-Sharifi (2014). The review classifies control approaches four ways. Classical control involves corrective control like PID systems. Hard control includes MPC and optimal control. Soft control encapsulates fuzzy logic and data-driven input-to-control-action mappings like artificial neural networks. Hybrid control is a combination of any number of these approaches. For both MPC and optimal control, the system model and/or the cost function need to be optimized. The survey documents approaches including linear programming, genetic algorithms, and particle swarm optimization for control design. Reinforcement learning is a form of optimal contorl.

HVAC control is evaluated on several metrics. These include energy and economic savings, smoothness of control actions, thermal efficiency of HVAC systems, computational complexity of controllers, and robustness to disturbances in the environment.

Control based on physical models (Jin et al., 2007; Cortinovis et al., 2009b) is done in a follow up work to Cortinovis et al. (2009b) in Cortinovis et al. (2009a). The cost function is a sum of economic costs of fans and water pumps. The control variables are fan speed and excess hot water removal rate from the cooling tower. A grid search is done over the domain to optimize cost. They conclude that prioritizing fan speed increase over hot water removal leads to lower overall costs.

In Sayyaadi and Nejatolahi (2011), a comparison is drawn between single- and multi-objective optimization approaches for economic and thermal costs for a refrigeration system. The model used is physics-based. Genetic algorithms are used to find optimal parameters for a single cost function, or a pareto-frontier of parameters for multi-objective optimization. For the case of multi-objective optimized parameters, they deviated less from the economically and thermally ideal points, than did the parameters optimized for a single cost metric.

Vu et al. (2017) exploits domain knowledge, particularly affinity laws, to develop a composite model of a chiller plant using polynomial regression (PR) and multi-layer perceptrons (MLPs). The model predicts total power consumption as a function of temperature and flow rate of chilled water coming from cooling towers. The training data is augmented by randomly perturbing control variables to aid the model's generalization.

A more general optimization problem is addressed by Wei et al. (2014). Total energy cost of 4 chiller plants with different thermal efficiencies is minimized. Control variables are water flow rate, water temperature change, and an on/off switch for each plant. Energy models using MLPs are learned for each plant. Gradient-free optimization is used to find control points. First, a genetic algorithm selects which plants are on, then particle swarm optimization (PSO) selects candidate points using the remaining two control variables.

Kusiak and Xu (2012) employ MLPs using auto-regressive features for indoor temperature and energy consumption models with PSO. Two MLPs are used with a time-window of features to model temperature and energy consumption. The time window depends on the auto-correlations of each feature. Particle swarm optimization with constraints in indoor temperature is used to find optimal control.

Fault tolerant approaches also have relied on a model of the system. Bengea et al. (2015) use a concurrent fault-detection algorithm to model faults in a model-predictive controller to output control actions.

2.13 Summary and overview of research problem

In prior sections disparate concepts and their overlap was reviewed. While transfer learning has a large body of work in the supervised learning domain, it has been used for control and reinforcement learning approaches too. In that case, transfer usually has been via learning a shared representation of domains, or learning a data-driven model for planning-based control. Policy transfer strategies have been found useful as well. However the methods for selection of policies and predicting their performance prior to learning presents an area of further investigation for optimal transfer.

Alone, reinforcement learning has lent itself to control problems where knowledge of optimal solutions is not available in a supervised manner. Work has been done in to speed up RL by using meta-learning approaches. To mitigate the exploratory nature of algorithms under real-world constraints, research on safety also has been done by way of reward shaping and constrained MDPs.

In controls literature too, the adaptability inherent to RL algorithms has been exploited to learn policies that can learn to adapt to abrupt system changes. Similarly, model-based control has benefited from learning data-driven models of the system.

Some work has been done in the combination of adaptive control, transfer learning, and reinforcement learning. Partially Constrained Models rely on designing a simplified model of a target task with restrictions removed to quickly learn the basics and safe-sets, before transferring to the target task. However, the simplified task formulation is manual effort, and the learning algorithms do not themselves incorporate structural relationships between domains. To that end, some preliminary work has been done using reconstruction errors for MDPs as measures of similarity, however they have not thoroughly investigated their use for selection of source tasks to transfer from. Safety has also been explored in safe exploration, where a baseline policy is learned and the agent becomes less risk-averse over time. However, a teacher baseline policy is already assumed to have been selected as a source for transfer.

The conjunction of these areas merits further research. By understanding the relationships between tasks autonomously, can transfer of RL knowledge across domains be done with less redundant exploration across candidate policies and states, smaller variance in learned actions, and with more efficiency with respect to available data?

CHAPTER 3

Approach

3.1 Problem formulation

The overarching aim for this work is to study the application of RL for fault-adaptive control, leveraging transfer learning and optimal control theory. A combination of these two approaches is intended to make control scalable and adaptive. Scalable, by reducing the design and operational cost of applying control to a population of control tasks via transfer. And adaptive, by learning to optimize control in presence of abrupt or incipient changes in the system.

Research towards these goals will steer related work in transfer and reinforcement learning which addresses disparate goals of this work. For example policy transfer in reinforcement learning (section 2.4.3), reward-shaping (section 2.4.1), meta-learning gradient-based optimization (section 2.7), and constrained MDPs for safe RL (section 2.5). Figure 3.1 illustrates the research goals: methods for transfer of reinforcement learning control across hybrid systems such that domain-specific safety constraints are efficiently met. In the following section, the problem is formally defined. Later still, the formal specifications are contextualized with the research goals.

3.2 Problem definition

Formally, given a set of source control tasks $\mathscr{T}_s = \{T : (M \in \mathscr{M}_s, \pi \in \Pi_s)\}$, *curate* the set of source policies via distillation, such that the smallest $\mid \Pi_s \mid$ can yield the highest expected value on $\mathscr{M}_s$. Given a target task $T_t : \{M_t, \pi_t = \emptyset\}$, select a policy or policies for *transfer* to M_t. The selection of policies is informed by the relationship between MDPs and their policies: $DIST_M : M \times M \rightarrow \mathbb{R}$, and $DIST_\pi : \pi \times \pi \rightarrow \mathbb{R}$. Based on the relationships, a loss function L_t will evaluate the expected performance of each $\pi \in \Pi_s$ on M_t. The selected policy may be reused $f_{reuse} : \Pi \rightarrow \Pi$, or the selected policies may be distilled $f_{distil} : [\Pi]^{\mathbb{Z}^+} \rightarrow \Pi$ into an initialization π_t^0 for M_t. Where M_t will be safety-constrained via reward-shaping with $f_c : \mathscr{M} \rightarrow \mathscr{M}$ to M_t'. Training π_t^0 on M_t' will be done using on-policy RL. Figure 3.1 illustrates this framework.

The different functions in the formulation represent the range of considerations needed for the proposed solution. The low-level purpose of this research is to propose, compare, and theorize about these candidate functions, which are hypothesized to fulfill the high-level goals as described previously. In the following subsections, the properties of these functions are discussed. Later in this work, some candidates for these are presented.

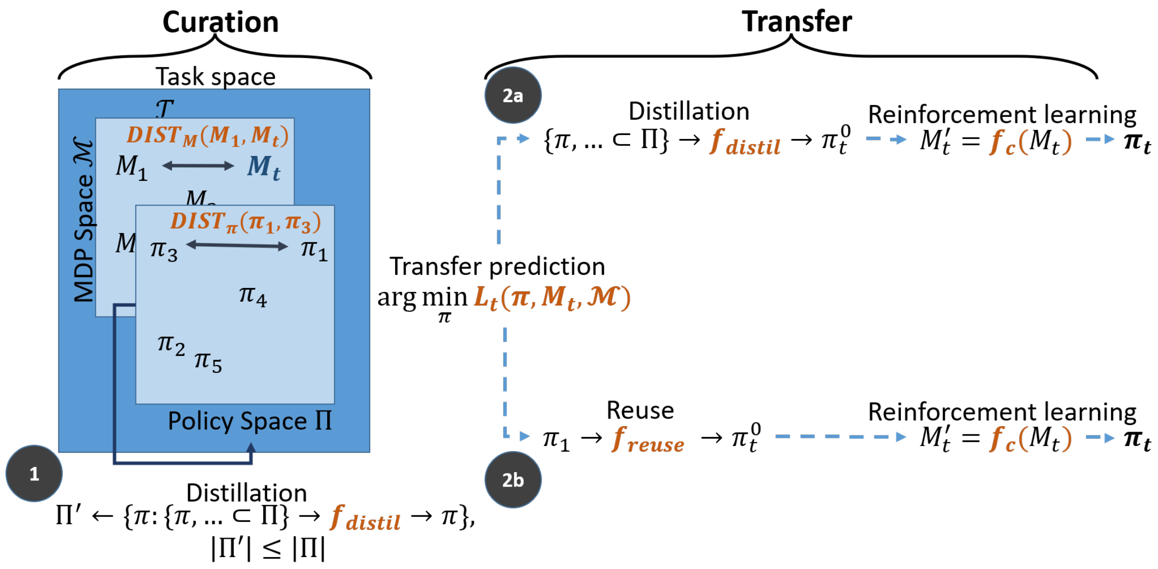


Figure 3.1: A formal representation of the problem, connecting research concepts, divided into two stages. (1) Curation: the set of source policies is distilled to keep the size small but representative of the population of tasks. (2) Transfer: Either selecting a (2a) single policy for reuse, and training on the new task. Or (2b) sampling a subset of policies for distillation and training on the new task. Training is safety-constrained. The functions $DIST_M$ and $DIST_\pi$ represent the structural relationships in the task space. A loss function L_t evaluates policies for transfer. Multiple policies are initialized into one using f_{distil}, and single policies are reused using f_{reuse}. The initialized/reused policy π_t^0 is trained on M_t' which has added safety constraints via f_c.

3.2.1 Curation of policies

The approach is divvied into two stages. The offline stage comprises of curating the set of source tasks for optimal sampling for the second stage. The complexity of selecting a policy to transfer scales linearly with the size of the source task space, $\mathcal{O}(\mid \mathcal{T}_s \mid)$. In that case, the approach can benefit from having a subset of eligible policies curated for transfer. To that end, a distance measure $DIST_\pi$ is needed such that, for two tolerance values $\varepsilon, \delta \in \mathbb{R}$ for policy differences and transfer performance respectively,

$$\begin{aligned}
&\forall \pi_1, \pi_2 \in \Pi_s, \\
&\texttt{if } DIST_\pi(\pi_1, \pi_2) \leq \varepsilon, \\
&\texttt{then } \mathbb{E}_{M \sim \mathcal{M}_s} V(x \mid \pi_1 \rightarrow \pi_t, M) \leq \mathbb{E}_{M \sim \mathcal{M}_s} V(x \mid \pi_2 \rightarrow \pi_t, M) + \delta
\end{aligned} \tag{3.1}$$

Therefore, given an acceptable tolerance δ in performance on the target task, policies expected to perform similarly within a tolerance of ε can be distilled. However, policies should be diverse enough such that for each MDP in the source set $\mathcal{M}_s$, there exists a policy in the distilled set Π_s' which has an expected performance comparable to the highest performing policy for that MDP in the original set. The measure may also be used to evaluate new tasks and policies for inclusion in the set over time, as a way of conducting lifelong learning. This can be presented as a constrained minimization problem:

$$\begin{aligned}
&\min_{\Pi_s} \mid \Pi_s' \subset \Pi \mid \\
&\texttt{s.t.} \quad \mathbb{E}_{M \in \mathscr{M}_s}\left(\max_{\pi \in \Pi_s'} V(x \mid \pi, M)\right) \geq V_{max} - \delta \\
&\texttt{where} \;\; V_{max} = \max_{\pi \in \Pi_s} \mathbb{E}_{M \in \mathscr{M}_s} V(x \mid \pi, M)
\end{aligned} \tag{3.2}$$

Work on a candidate distance measure and policy curation is presented in sections 4.3, 4.4.

3.2.2 Transfer of policies

The online adaptation stage conducts transfer. Central to the solution is a measure predicting the quality of transfer performance, L_t. The measure should rank policies which are liable to converge faster, yield a higher cumulative reward, or perform well in a subset of transfer metrics as defined in section 2.1. Very trivially, L_t should measure the expected returns when following a policy on the target task:

$$L_t(\pi) = \mathbb{E}_{x \sim X_t} V(x \mid \pi, M_t) \tag{3.3}$$

However, predicting value of states on the target task preemptively is the challenge. Simply because the learning process is stochastic, and the target MDP has not been explored. It can be addressed by learning a data-driven model of the MDP, or by importance sampling of rewards from a buffer of experiences:

$$\begin{aligned}
B &= (x, u, x', r), \ldots \sim M_t \\
L_t(\pi, B) &= \mathbb{E}_{x,u,r \sim B} P(u \mid x, \pi) \cdot r
\end{aligned} \tag{3.4}$$

A similar approach is used as a selection heuristic in section 4.4, where its limitations as a heuristic are also discussed.

Adaptation is achieved via *policy transfer*. Whether reused, or distilled, transfer is looked through the shared lens of policy initialization $f_{init} : \pi \in \Pi_s \rightarrow \pi_t^0$, $f_{init} \in \{f_{reuse}, f_{distil}\}$. The initialization takes inspiration from Model Agnostic Meta Learning, where the meta-learning step updates parameters in a direction most likely to adapt to a range of tasks from that same population. Once a policy/policies has been selected for transfer, f_{init} transforms them further to reduce the transfer gap. That is, the expected valuation of experiences, after training on M_t, from an initialized policy should be greater than that of an uninitialized policy. That is:

$$\begin{aligned}
&\{\pi, ...\} = \arg\min_{\pi \in \Pi_s} L_t(\pi) \\
&\forall f_{init} \in \{f_{distil}, f_{reuse}\}, \\
&\mathbb{E}_{x \sim X_t} V(x \mid f_{init}(\pi) \rightarrow \pi_t, M_t) \geq \mathbb{E}_{\pi \sim \{\pi, ...\}, x \sim X_t} V(x \mid \pi \rightarrow \pi_t, M_t)
\end{aligned} \tag{3.5}$$

Work in initializing new policy parameters is presented in sections 4.4, 5.

Once a policy has been selected and initialized, it will conclude transfer by training of the target MDP using on-policy RL. On-policy algorithms are chosen to account for the time and safety constraints of interacting sub-optimally with the environment during exploration. M_t will be constrained, with constraints being informed from domain expertise and structural similarities with the selected source tasks.

$$\mathbb{E}_{x \sim X_t} V(x \mid \pi_t, f_c(M_t)) \geq \mathbb{E}_{x \sim X_t} V(x \mid \pi_t, M_t) \tag{3.6}$$

In all prior constraints, the expected value of policies implicitly encodes the principles of adaptive safety-constrained control via transfer as used for this research. Given the same time resources or interactions with the target, the proposed approaches should get higher valuations per unit resource. This can be represented as (1) faster speed to convergence to an optimal policy where (2) control actions have known bounds. And (3) safety constraints are explicitly defined in M_t' such that experiences within bounds are more optimized. Ultimately,

$$\arg\max_{DIST, L_t, f_{init}, f_c} \left(\mathbb{E}_{x \sim X_t} V(x \mid \pi_t^0 \rightarrow \pi_t, M_t') - \mathbb{E}_{\pi \sim \Pi_s, x \sim X_t} V(x \mid \pi, M_t) \right) \tag{3.7}$$

3.3 Methodology

At the high level, application of this research will proceed in the following ways: implementation of proposed methods in code, their validation by evaluation against benchmarks in literature and on test-beds, and investigation of their application in real-world scenarios.

The software design for this research will be interoperable with requirements for real-world deployment, and be comparable against benchmark algorithms. This includes the interface specification of OpenAI gym and Gazebo environments. Conventions for the state and action space used in process control of UAVs and HVACs will be followed.

For validation of proposed approaches, test-beds for HVAC and UAV systems will be developed where faults and system changes can be dynamically introduced. The test-beds will be consolidated under a uniform application programming interface (API) such that the software application of this work is domain agnostic.

Finally, case studies of application of methods will be done. The case studies will compare control methods standard in their domains against this research. Studies will also be done of deploying controllers based on this work on real-world systems from scratch (i.e. bootstrapping).

3.4 Speed of convergence towards target task

A speedy learning stage is a shared goal for machine learning, fault-tolerant control, and safety. An algorithm that learns fast is more sample efficient, can quickly adapt to changes in a system, and has a smaller exploratory stage where it may be prone to uncertain behaviors. By building upon related work in sub-fields of machine learning, speed is approached in a multitude of ways.

Sample efficiency can be improved by synthesizing training samples for the learning algorithm. This in turn can be achieved by learning a data-driven model $M_{t,\omega}$ of a target MDP M_t. Using an intermediate data-driven model is a representation of f_{reuse} in the problem formulation.

$$\begin{aligned} B &: \{(x,u,x',r),...\} \sim M_t \\ M_{t,\omega} &= \arg\min_{\omega} \mathbb{E}_{x,u,x',r\sim B} L((x',r), M_{t,\omega}(x,u)) \\ f_{reuse}(\pi_\theta) &= \arg\max_{\theta} \mathbb{E}_x G(x \mid \pi_\theta, M_{t,\gamma}) \end{aligned} \tag{3.8}$$

In this case, model-augmented learning can be sample efficient if (1) an accurate model of the MDP is learned, and (2) which in turn is also done sample-efficiently. Both of these are research questions to be addressed for this task. The thrust of the work will be to use fully-connected neural networks to learn models, but incorporate techniques like residual modeling with a physics model and meta-learning. Some related work on fault-tolerance in RL via sample efficiency is presented in sections 4.2 and 4.3.

When reusing a policy, a smaller exploratory stage is possible if a source task T_s is selected which is similar to the target task T_t. Policy reuse in transfer learning employs an older policy expected to perform better at the target task. The selection mechanism of policies is an open problem which may be addressed by an investigation of similarity measures ($DIST_M$, $DIST_\pi$) between tasks. Ultimately, the goal is to select the highest performing policy, and to speed up the sampling process itself by curating a small library of useful source policies Π_s. Similarity measures between tasks are reviewed in section 2.6. The focus of this work is to develop distance measures in policy space. RL algorithms used in this work use stochastic policies, therefore distance measures between probability distributions will be used as a starting point, for example the Kullback-Leibler divergence and the Wasserstein metric (section 2.6). Some preliminary work is discussed, also in sections 4.3 and 4.4.

$$\begin{aligned} B &: \{(x,u,x',r),...\} \sim M_t \\ DIST_\pi(\pi_1,\pi_2) &= f(\{p_1 : P(u \mid x,\pi_1), x,u \sim B\}, \{p_2 : P(u \mid x,\pi_2), x,u \sim B\} \end{aligned} \tag{3.9}$$

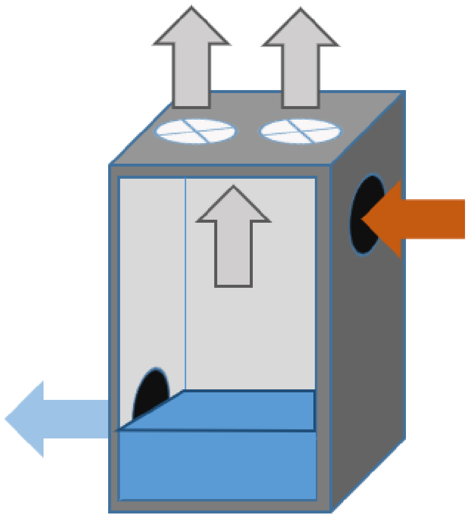
Finally, fast adaptation is possible by initializing a policy function's parameters π_t^0 such that they are likely to reach an optimum for a variety of tasks in the population $\mathscr{T}_s$. Policy distillation in transfer learning uses a set of source tasks to generate a policy for the target task. Model-agnostic meta-learning (MAML) is general approach applicable to gradient based optimization problems. It samples experiences from a set of source tasks to precondition a parameter initialization to adapt to a range of target tasks. Therefore, application of MAML on the distillation step can yield a performance advantage. However, the sampling step in MAML and the higher order gradients can be time intensive. This approach is discussed in more detail in section 4.4. Again, the structural knowledge of the set of source tasks can help selecting policies $[\pi]^{\mathbb{Z}^+}$ to distill that are expected to perform well.

$$f_{distil} = \texttt{MAML}([\pi]^{\mathbb{Z}^+}) \tag{3.10}$$

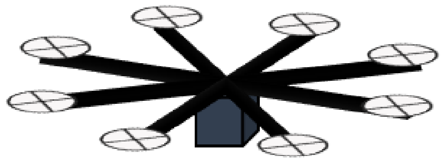
The research questions in this case are the size and method of obtaining the policy sample $[\pi]^{\mathbb{Z}^+}$, and the computational complexity of meta-learning.

3.5 Validation test-beds

While the research may stand on its own theoretically, an empirical validation can help to understand real-world concerns like random disturbances and higher order effects. Similarly, for generality, the work must be shown to apply to a range of hybrid systems. To that end the UAV and HVAC system test-beds are developed. They belong to different ends of the spectrum in their time constants, dimensionality of control actions, and



(a) HVAC cooling tower expels heat via evaporation by controlling fan speeds.



(b) Octo-rotor UAV manoeuvres by controlling propeller speeds.

Figure 3.2: Simplified schematics of the two test-beds, showing the different mechanisms of operation.

nature of exogenous variables (wind, temperature, humidity, load etc.). The test-beds will be developed with high fidelity in mind and will use physics-driven modeling. Internal faults and exogenous disturbances will be dynamically introduced to present adaptation and FTC scenarios. The test-bed development is discussed in sections 4.5 and 6.

3.6 Case studies

Similar to the above, the validation of research also requires that its viability in real world applications be examined more holistically. That means benchmarking the proposed approaches against control algorithms used as a standard in respective domains (PID, MPC) in the context of adaptive control. It also necessitates a study of the mechanism for starting a transfer learning based RL controller from scratch, involving system modeling, software deployment, and identifying opportunities for transfer. To that end, section 4.1 compares the use of RL as an adaptive control method versus model-based approaches. Finally, 4.5 investigates the design of transfer and RL in a HVAC system.

In the following pages, chapter 4 describes the preliminary studies for this approach in more detail. In that section, adaptive FTC is achieved by reasoning about the parameters of the machine learning algorithm to make learning adaptive. That is, the problems of sample inefficiency for stochastic gradient descent are addressed. On the other hand, section 5 reasons about adaptive machine learning control in the domain of the process itself. There, the RL policy is initialized and adapted as informed by system and fault identification. Finally, chapters 6, 7, and 8 conduct a case study of adaptive control on a bespoke virtual test-bed of an octo-rotor UAV.

CHAPTER 4

Data-driven adaptation of machine learning control

Machine learning algorithms iteratively learn from data. If learning is allowed to continue for the duration of operation of a control task, the machine-learned controller can adapt to systemic and environmental changes. However, the speed of adaptation can be slow. Often times not enough data is available in short enough time for the learning algorithm to converge to a (local) optimum if it continues as it is.

In this section, the machine learning process is examined to mitigate this challenge. The machine learning process is principally concerned with how data is used to update parameters of a learned function. First, section 4.1 demonstrates utility of RL as an adaptive control mechanism robust to sensor noise. Sections 4.2, 4.3, and 4.4 look at different ways of exploiting past experiences to speed up the learning process by modification of the components of the machine learning algorithm. For example, by using previously gathered measurements to learn a data-driven transition model $Tr : X \times U \rightarrow X$. Or by re-using previously learned control policies' parameters to get a head-start on the changed system, by initializing the target policy function's parameters. Finally, section 4.5 looks at a practical example of bootstrapping data measured from a classical PID controller to develop a RL controller on a data-driven model of an HVAC system.

So long as the Markov Property holds, the machine learning algorithm is concerned with optimizing the reward function. The mechanisms proposed to make RL sample efficient are agnostic to the nature of underlying systemic changes.

4.1 Reinforcement learning as parameter transfer mechanism for fault-tolerant control

Complex systems are expected to operate efficiently in diverse environments. A controller that can function within bounds despite malfunctions and degradation in the system, or changes in the environment is safer than a controller which does not respect process constraints. With the rise in automation and increasing complexity of systems, direct or timely human supervision may not always be possible. Fault Tolerant Control (FTC) seeks to guarantee stability and performance in nominal and anomalous conditions.

An MDP with a fault can be considered a new target task for homogeneous transfer of a RL problem (2.1). That is, the input (state) and output (action) space of the control policy for the MDP before and after a fault are the same. This is contrasted with new target tasks created as a result of manufacturing variations, natural degradation, and distributional shift of exogenous variables. To proceed with the investigation of data-driven approaches, it is arguably necessary to understand their benefits and drawbacks against popular approaches in relevant domain literature.

The objective of this research task is to benchmark performance of classical controls approaches against reinforcement learning in the context of task adaptation. Task adaptation my be necessitated by abrupt or transient faults, and natural system changes. Particularly, the performance under time and measurement constraints is analyzed. Approaches are to be evaluated using algorithm-specific measures like episodic rewards, and by domain specific metrics for the test-bed. In the context of the framework illustrated in figure 3.1:

$$f_{reuse} = \pi_s \tag{4.1}$$

The main contribution of this work is to benchmark RL-based controllers against MP control. We implement discrete-time MP control and propose two approaches for applying RL principles towards FTC. The controllers are tested on a model of fuel transfer system of a C-130 cargo plane where the controllers attempt to maintain a balanced fuel distribution in the presence of leaks.

4.1.1 Preliminaries

4.1.1.1 Model Predictive Control

Model Predictive Control (MPC) optimizes the current choice of action by a controller that drives the system state towards the desired state. It does this by modeling future states of the system+environment and a finite receding horizon over which to predict state trajectories and select the "best" actions . At each time step, a sequence of actions is selected up until the horizon such that a cost or distance measure to a desired goal is minimized. For example, for a system with state variables $(x_1, x_2, ..., x_N)$ and the desired state $(d_1, d_2, ..., d_N)$ the cost may be a simple Euclidean distance measure between the two vectors.

A model predictive controller operates over a set of states $x \in X$, actions $u \in U$, and a state transition model of the system $Tr : X \times U \rightarrow X$. It generates a tree of state trajectories rooted at the system's current state. The tree has a depth equal to the specified lookahead horizon for the controller. The shortest path trajectory, that is, the sequence of states and actions producing the smallest cost is chosen, and the first action in that trajectory is executed. The process repeats for each time step.

Garcia et al. (1989) discuss MPC in further detail particularly in reference to continuous systems. Abdelwahed et al. (2005) presents a case study for MPC of a hybrid system. The control algorithm for the discrete subsystem is essentially a limited breadth-first search of state space. They use a distance map, which uses the euclidean distance between the controller state and the closest goal state as the cost function. The MPC algorithm implemented for this work is adapted from Abdelwahed et al. (2005) and described in Algorithm

1.

Algorithm 1 Model Predictive Control

```
Require: system model $Tr : X \times U \to X$
Require: distance map $D : X \to \mathbb{R}$
Require: horizon $N$
 1: $x_0 \leftarrow CurrentState$
 2: Initialize state queue $Queue = \{x_0\}$
 3: Initialize state set $Visited = \{x_0\}$
 4: Initialize optimal state $Optimal \leftarrow Null$
 5: $MinDistance \leftarrow \infty$
 6: while $Queue.length > 0$ do
 7:     $state = Queue.pop()$
 8:     if $state.depth > N$ then
 9:         Break
10:     end if
11:     $newStates = state.neighbourStates \cap \neg Visited$
12:     for $x \in newStates$ do
13:         Visited.add(x)
14:         Queue.insert(x)
15:         if $D(x) < MinDistance$ then
16:             $MinDistance = D(x)$
17:             $Optimal = x$
18:         end if
19:     end for
20: end while
21: while $OptimalState.previous \neq x_0$ do
22:     $Optimal \leftarrow Optimal.previous$
23: end while
24: $action = T(Optimal.previous, \cdot) \to Optimal$
```

4.1.1.2 Q-Learning

Reinforcement learning is the learning of behaviour by an agent, or a controller, from feedback through repeated interactions with its environment. Kaelbling et al. (1996) divide RL into two broad approaches. In the first, the genetic programming approach, an agent explores different behaviours to find ones that yield better results. In the second, the dynamic programming approach, an agent estimates the value of taking individual actions from different states in the environment. The value of each state and action dictates how an agent behaves. This section explores methods belonging to the latter approach.

An agent (controller) is connected to (interacts with) the system+environment through its actions and perceptions. Actions an agent takes cause state transitions of the system in the environment. The change is perceived as a reinforcement signal from the system+environment, and the agent's measurement of the new state.

The standard RL problem can be represented as a Markov Decision Process (MDP). A MDP consists of a set of states X, a set of actions U, a reward function $R : X \times U \times X \to \mathbb{R}$ which provides reinforcement after each state change, and a state transition function $Tr : X \times U \rightarrow \Pi(X)$, which determines the probabilities of going to each state after an action. This is the model for the system operating in an environment. For the purposes of this paper, the transition function is assumed to deterministically map $x \in X$ and $u \in U$ to the next state $x' \in X$.

In a MDP, everything an agent needs from the environment is encoded in the state. The history of prior states does not affect the value or reward of the following states and actions. This is known as the Markov property.

An agent's objective is to maximize its total future reward. The value of a state is the maximum reward an agent can expect from that state in the future.

$$V(x) = \max_{u \in U}(\mathbb{E}[R(x,u,x') + \gamma V(x')]), \tag{4.2}$$

where γ is the discount factor that weighs delayed rewards against immediate reinforcement. Alternatively, an agent can learn the value of each action, the q-value, from a state:

$$Q(x,u) = \mathbb{E}[R(x,u,x') + \gamma \max_{u \in U} Q(x',u)]. \tag{4.3}$$

We use the q-value formulation for this paper. An agent can exploit either value function to derive a policy that governs its behaviour during operation. To exploit the learned values, a policy greedily or stochastically selects actions that lead to states with the highest value:

$$\begin{aligned} \pi_{greedy}(x) &= arg \max_{u \in U} Q(x,u) \\ \pi_{stochastic}(x) &= \Pi(U) \end{aligned} \tag{4.4}$$

(4.2), known as the Bellman equation, and (4.3) can be solved recursively for each state using dynamic programming. However, the complexity of the solution scales exponentially with the number of states and actions. Various approaches have been proposed to approximate the value function accurately enough with minimal expenditure of resources.

Monte Carlo (MC) methods (Sutton and Barto (2017)) alleviate the Curse of Dimensionality by approximating the value function. Instead of traversing the entirety of state-space, they generate episodes of states connected by actions. For each state in an episode, the total discounted reward received after the first occurrence of that state is stored. The value of the state is then the average total reward across episodes. Generating an episode requires a choice of action for each state. The exploratory action selection policy π_{exp} can be uniform, partially greedy with respect to the highest valued state (ε-greedy), or proportional to the relative values of actions from a state (softmax). The MC approach still requires conclusion of an episode before updating value estimates. Additionally, it may not lend itself to problems that cannot be represented as episodes with terminal or goal states.

Temporal Difference (TD) methods provide a compromise between DP and MC approaches. Like MC, they learn episodically from experience and use different action selection policies during learning to explore the state space. Like DP they do not wait for an episode to finish to update value estimates. TD updates the value function iteratively at each time step i.

$$\begin{aligned} G_i &= R_{i+1} + \gamma \max_{a \in A} Q_{i-1}(x_{i+1}, a) \\ Q_i(x_i, a_i) &= Q_{i-1}(x_i, a_i) + \alpha(G_i - Q_{i-1}(x_i, a_i)), \end{aligned} \tag{4.5}$$

where α is the learning rate for the value function. G_i, called the return, is a new estimate for the value. The value function is updated using the error between the new estimate and the existing approximation. Note that G estimates the new value by only using the immediate reward and backing up the discounted previous value of the next state. This is known as $TD(0)$. TD methods can be expanded to calculate better estimates of values by explicitly calculating rewards several steps ahead, and only backing up value estimates after that. These are known as $TD(\lambda)$ methods. For example, the return for $TD(2)$ is:

$$\begin{aligned} G_i^{(2)} &= R_{i+1} + \gamma(R_{i+2} + \gamma \max_{a \in A} Q_{i-1}(x_{i+2}, a)) \\ &= R_{i+1} + \gamma R_{i+2} + \gamma^2 \max_{a \in A} Q_{i-1}(x_{i+2}, a) \\ Q_i(x_i, a_i) &= Q_{i-1}(x_i, a_i) + \alpha(G_i^{(2)} - Q_{i-1}(x_i, a_i)) \end{aligned} \tag{4.6}$$

In the extreme case, when G is calculated for all future steps until a terminal state s_T, the TD method becomes a MC method because value estimates then depend on the returns from an entire episode.

An improvement on $TD(\lambda)$ methods is achieved by making the return G more representative of the state space. Values of actions are weighed by the agent's action selection policy. This is known as $n-$step Tree Backup (Precup (2000)). Defining $\pi_{exp}(u|x)$, x_T as the terminal time, and:

$$\begin{aligned} V_i &= \sum_u \pi_{exp}(u|x_i) Q_{i-1}(x_i, u) \\ \delta_i &= R_{i+1} + \gamma V_{i+1} - Q_{i-1}(x_i, u_i), \end{aligned} \tag{4.7}$$

where V_t is the expected value of x_t under the action selection policy π_{exp}, and δ_t is the error between the return and prior value estimate. Then the n^{th}-step return is given by:

$$G_i^{(n)} = Q_{i-1}(x_i, a_i) + \sum_{k=t}^{\min(t+n-1,T-1)} \delta_k \prod_{i=t+1}^{k} \gamma \pi_{exp}(u_i|x_i). \tag{4.8}$$

The value function can then be updated using the error $G_i^{(n)} - Q_{i-1}(x_t, u_t)$. A longer lookahead gives a more representative estimate of value. A greedy action selection policy with $n = 0$ reduces this to the $TD(0)$.

The following section presents a modified $n-$step Tree Backup algorithm tailored for adaptive fault-tolerant control.

4.1.2 Reinforcement Learning-based controller design

Temporal difference based RL methods iteratively derive a controller's behaviour by estimating the value of states and actions in a system through exploration. During operation, the controller exploits the knowledge gained through exploration by selecting actions with the highest value. This works well if the system dynamics is stationary. The agent is able to achieve optimal control by exploring over multiple episodes and iteratively converging on the value function. Conservative learning rates can be used such that the value approximation is representative of the agent's history.

When a fault occurs in a system, its dynamics change. The extant value function may not reflect the most rewarding actions in the new environment model. The agent seeks to optimize actions under the new system dynamics. The agent must, therefore, estimate a new value function by exploration every time a fault occurs. The learning process can be constrained by deadlines and sensor noise. This paper proposes design of a RL-based controller subject to the following requirements:

Adaptivity: The controller should be able to remain functional under previously unseen faults (Goldberg et al. (1993)).

Speedy convergence: The agent's behaviour should be responsive to faults as they occur.

Sparse sampling: The system model may lose reliability following a fault due to sensor noise or incorrect diagnosis. The agent should be able to calculate value estimates representative of its local state space from minimal (but sufficient) sampling.

Generalization: The agent should be able to generalize its behaviour over states not sampled in the model during learning.

The temporal difference RL approach is inherently adaptive as it frequently updates its policy from exploration. Its responsivity can be enhanced by increasing the learning rate α. In the extreme case, $\alpha = 1$

replaces the last value of an action with the new estimate at that time step. However, larger values of α may not converge the value estimate to the global optimum.

RL controllers have to balance exploratory actions to discover new optima in the value function against exploitative actions that use the action values learned so far. An exploration parameter $\varepsilon \in [0,1)$ can be set, which determines the periodicity of value updates and hence the controller's responsivity to system faults.

Hierarchical RL can be used to sample the state space at various resolutions (Lampton et al. (2010)). An agent can use a hierarchy function $H : X \rightarrow \Re^{+}$ that gives the step size of actions an agent takes to sample states. H can yield smaller time steps for states closer to goal and coarser sampling for distant states. This allows the agent to update the value function more frequently with states where finer control is required.

TD RL methods can use *function approximation* to generalize the value function over the state space instead of using tabular methods. Gradient descent is used with the error computed by the RL algorithm to update parameters of the value function provided at design time. Function bases like radial, Fourier, and polynomial can be used to generalize a variety of value functions.

Algorithm 2 implements Variable $n-$step Tree Backup for a single episode. It updates the value estimate after exploring a sequence of states up to a finite depth d of actions with varying step sizes. Value estimates for each state in the sequence depend on following states up to n steps ahead. A RL-based controller explores multiple episodes to converge on a locally optimum value for actions. During operation it exploits the value function to pick the most valuable actions as shown in (4.4).

The following subsections discuss two approaches to approximating values for a RL-based controller.

4.1.2.1 Offline Control

An offline RL-based controller derives the value function by dense and deep sampling of the state space. The learning is done once each time a new value approximation is required. The agent learns from episodes over a larger sample of states in the model. Each episode lasts till a terminal state is reached. Offline control is computationally expensive. However, by employing a partially greedy exploratory action selection policy π and a suitable choice of learning rate α, it is liable to converge to the global optimum.

4.1.2.2 Online Control

Online RL-based control interleaves operation with multiple shorter learning phases. The agent sporadically explores via limited-depth episodes starting from the controller's current state. The sample size is small and local. Each exploratory phase is comparatively inexpensive but may only converge to a local optimum.

In other words, like MPC, online RL employs a limited lookahead horizon periodically to learn the best actions. Unlike MPC, online RL remembers and iteratively builds upon previously learned values of actions.

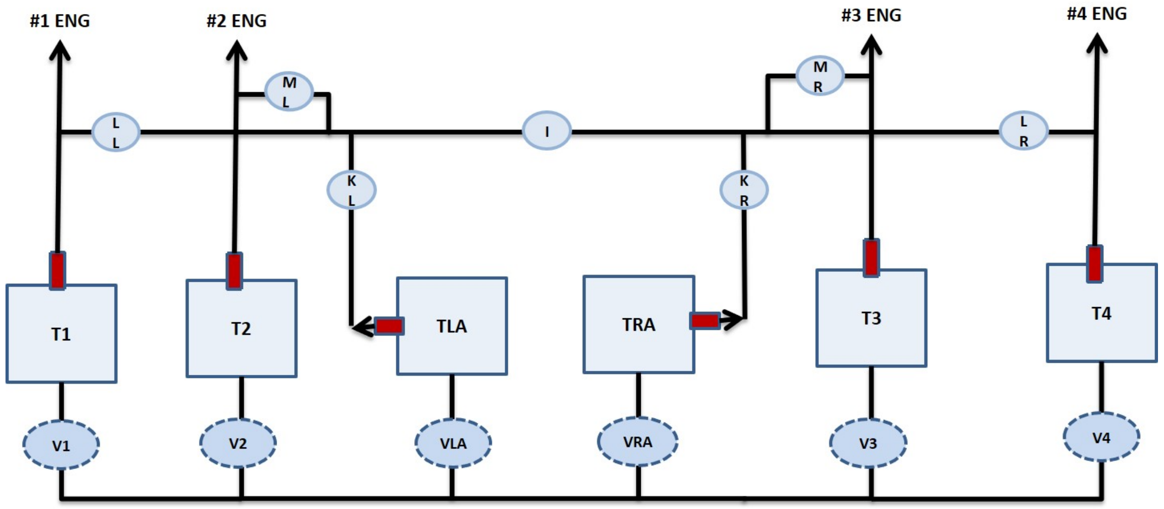


Figure 4.1: Simplified C-130 fuel system schematics. The controller manages valves (dotted ovals) and can observe fuel tank levels. Net outflow to engines via pumps (solid ovals) is controlled independently.

4.1.3 Case Study

The three control schemes (MP, Offline RL, and Online RL) are applied to a simplified version of the fuel tank system of a C-130 cargo plane (see Fig. 4.1). There are six fuel tanks. The fuel tank geometry is symmetric and split into a left/right arrangement where each side primarily feeds one of two engines. Control of fuel transfer is required to maintain aircraft center of gravity (CG). Changes in the fuel pump operating performance, plumbing, engine demand, and valve timing can affect the way each tank transfers fuel. The CG is a function of fuel quantities in the tanks, therefore, the fuel control system can restore the CG by turning transfer valves off or on to bring the system back into acceptable limits.

Fuel transfer is controlled by two components: pumps and valves. Pumps maintain a constant flow to the engines. Under nominal operating conditions, the fuel transfer controller is designed to pump fuel from outer tanks first, and then sequentially switch to inner tanks. Each tank has a transfer valve feeding into a shared conduit. When valves are opened, pressure differentials between tanks may result in fuel transfer to maintain the balance.

The state of the system is described by twelve variables: six fuel tank levels, which are continuous variables, and six valve states, that are discrete-valued. For simplicity, fuel levels are represented on a $0-100$ min-max scale. Valve positions are binary on/off variables. A controller can switch any combination of valves on or off, giving a total of 64 possible configurations fr the action space.

$$\begin{aligned} &x : \{T_1, T_2, T_{LA}, T_{RA}, T_3, T_4, V_1, V_2, V_{LA}, V_{RA}, V_3, V_4\} \\ &u : \{V_1, V_2, V_{LA}, V_{RA}, V_3, V_4\} \end{aligned} \tag{4.9}$$

Faults in the fuel tank system are leaks in fuel tanks. When a leak occurs, a tank drains additional fuel at a rate proportional to the quantity left. It is assumed that controllers have accurate system models and the fault diagnosis results are provided in a timely manner. WE make the single fault assumption, i.e., at any point in time, there is at most one fault in the system.

Sensor noise is simulated by scaling the measurements of tank levels with values derived from a Gaussian distribution centered at 1 with a standard deviation σ. The noise is assumed to be inherent to the sensors.

A trial begins with all tanks filled to capacity with a possible fault in one tank. The trial concludes when no fuel is left in tanks. For each trial, the maximum imbalance and the time integral of imbalance are used as performance metrics.

Goal states in the system are configurations where there is zero moment about the central axis. The magnitude of the moment is the imbalance in the system. The distance map D used by MP controller is a measure of imbalance.

$$D(x) = Abs(3(T_1 - T_4) + 2(T_2 - T_3) + (T_{LA} - T_{RA})) \tag{4.10}$$

RL controllers are supplied with a reward function R. The controller is rewarded for reaching low-imbalance states and for doing it fast when there is more fuel to spare.

$$\begin{aligned} R(x,x') = &\frac{T_1' + T_2' + T_{LA}' + T_{RA}' + T_3' + T_4'}{600} \\ &+ \frac{1}{1 + D(x')} \end{aligned} \tag{4.11}$$

RL controllers also hierarchically sample state space. The further a state is from goal, the more imbalanced the tanks are, therefore, the longer the action duration is at that state.

$$H(x) = 1 + Log_{10}(1 + D(x)) \tag{4.12}$$

Value function approximation is done via a linear combination $Q(x,u)$ of normalized fuel level and valve state products with a bias term. At each episode, the $Error$ is used to update weights $\vec{w}$ by way of gradient descent.

$$\begin{aligned} Q(x,u) &= \vec{w} \cdot \left\{ \frac{T_1(1+V_1)}{200}, \dots, \frac{T_4(1+V_4)}{200}, 1 \right\}^T \\ \nabla Q_w &= \left\{ \frac{T_1(1+V_1)}{200}, \dots, \frac{T_4(1+V_4)}{200}, 1 \right\}^T \\ \vec{w} &\leftarrow \vec{w} - \alpha \cdot Error \cdot \nabla Q_w \end{aligned} \tag{4.13}$$

RL controllers employ a softmax action selection policy π_{exp} during exploration (Doya et al. (2002)). The probability of actions is proportional to the values of those actions at that instant. This ensures that during learning, states which yield more reward are prioritized for traversal.

4.1.4 Results and discussion

For each of the three control schemes, thirty trials were carried out with random faults (Ahmed (2017)). Table 4.1 shows the values of the baseline parameters used. For these parameters, MP and online RL controllers sampled the same number of states at each step on average.

Name	Symbol	MP	Offline RL	Online RL
Learning rate	α	N/A	0.1	0.1
Discount	γ	N/A	0.75	0.75
Depth	d	N/A	30	5
Lookahead steps	n	N/A	10	5
Exploration rate	ε	N/A	N/A	0.4
Sampling density	ρ	1	1	0.5
Horizon	N	1	N/A	N/A

Table 4.1: Controller baseline parameters

Under baseline conditions, Figure 4.2 shows the performance of various controllers using two metrics. (1) *Maximum imbalance* is the average maximum value $D(x)$ reached during trials. (2) *Total imbalance* is the average time integral of $D(x)$ over each trial.

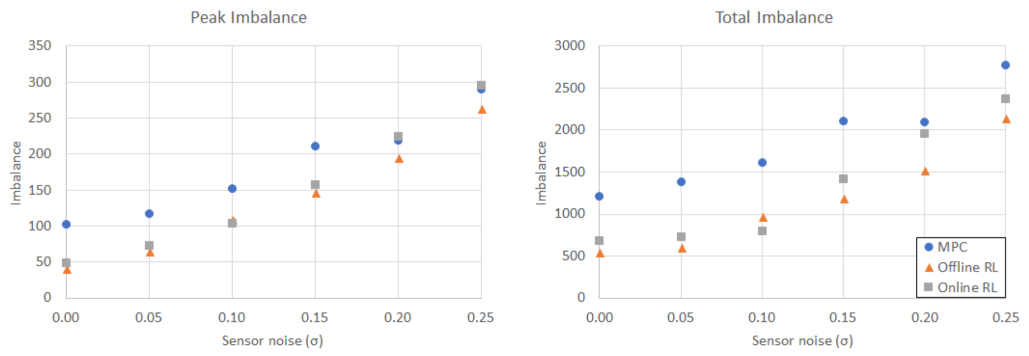


Figure 4.2: Performance of controllers under varying sensor noise. RL-based controllers are more robust to sensor noise than MPC. Lower imbalances are better.

These results are representative of how controllers sample the state space to select actions. MP control samples all reachable states within a horizon at each step. It finds the locally optimal action. The choice of each action depends on the instantaneous values associated with the states in the model. In case of sensor noise, sub-optimal actions may be chosen.

Offline RL control samples a large fraction of potential successor states once, and derives a single value function and control policy that is applied to select actions in the future. Online RL periodically samples a small, local fraction of successor states to make recurring corrections to its policy. It starts off being sub-optimal, but with each exploratory phase improves on its choice of actions till it converges to the lowest cost choices. It is possible in scenarios where response time is critical, online RL may not have time to sample enough states to converge to an acceptable policy.

Both RL methods rely on accumulated experience. Having a learning rate means that state values, and hence the derived control policy, depend on multiple measurements of the model. For Gaussian sensor noise, the average converges to the true measurement of the state variable. Therefore, the controller is influenced little by random variations in measurements. Furthermore, the choice of value function can regularize the policy derivation so it does not over-fit to noise.

RL methods are also sensitive to the form of the value approximation function, which may under-fit the true value of states or over-fit to noise. They forego some accuracy in state valuation by using approximations in exchange for the ability to interpolate values for unsampled sates. The use of neural networks as general function approximators may allow for a more accurate representation of true state values, and therefore, better control.

Another set of 30 trials was carried out with online RL and MP controllers to simulate sampling and processing constraints during operation. State sample sizes were restricted by lowering the sampling density ρ. At each step, the MP and RL controllers could only advance a fraction ρ of available actions to their neighbouring states to explore. The exploration rate was set to $\varepsilon = 0.2$ to ensure equal sample sizes for both controllers. Sensor noise was reset to $\sigma = 0$. Results are shown in Figure 4.3.

Online RL control is quick to recover from a fault and maintains smaller imbalance for the duration of the trial. By having a value function approximation, the controller estimates utilities for actions it cannot sample in the latest time step based off of prior experience. MP control, however, is constrained to choose only among actions it can observe in the model. It is, therefore, more likely to make suboptimal choices. With a decreasing sample density, MP control's performance deteriorates whereas online RL control is less sensitive to the change.

Finally, combining sensor noise and low sampling densities yielded a significant disparity in performance between the RL and MP control approaches as shown in Table 4.2.

The results show that reinforcement learning-based control is more robust. When the model is insufficiently observable, either due to sensor noise or due to sampling constraints, RL controllers are able to generalize behaviour and deliver consistently better performance than MP control. The complexity of both MP and RL approaches is linear in the number of states sampled. However, by discarding prior experience and sampling states anew at each step, model predictive control forfeits valuable information about the

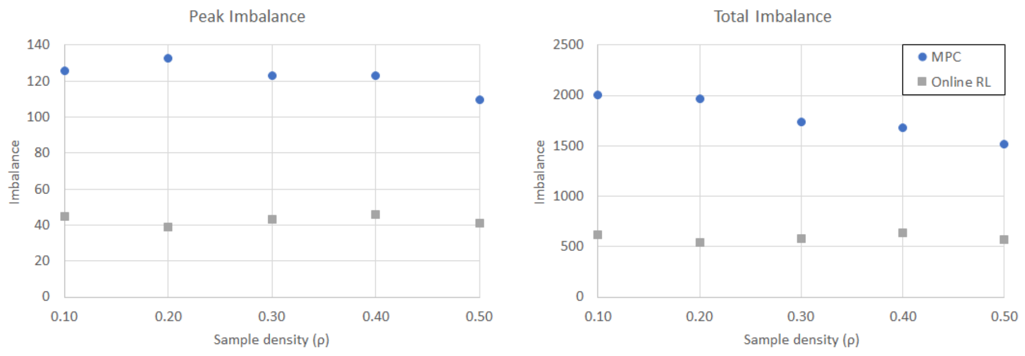


Figure 4.3: Performance of controllers under varying sample densities. RL-based controllers are more robust to smaller state samples than MPC. Lower imbalances are better.

Control	Max Imbalance	Total Imbalance
MP	147.66	1972.89
Online RL	75.17	721.57

Table 4.2: MP vs Online RL control
$(\sigma = 0.05, \rho = 0.5)$

environment that RL methods exploit.

4.1.5 Conclusion

Reinforcement learning-based control provides a competitive alternative to model predictive control, especially in situations when the system degrades over time, and faults may occur during operations. RL control's independence of explicitly defined goals allows it to operate in environments where faults may render nominal goal states unfeasible. The ability to generalize behavior over unseen states and actions, and to derive control policies from accumulated experience, make RL control the preferred candidate for system models subject to computational constraints, partially observable environments, and sensor noise. There are several venues for exploration in the choice of hyperparameters for RL based controllers which may yield further performance gains.

4.2 Fault-Tolerant Control of Degrading Systems with On-Policy Reinforcement Learning

Often times when a target task materializes due to a fault or another system change, there are constraints on the amount of data available or the time available to adapt. Similarly, the nature of the new task may need to be characterized via system identification or fault detection and isolation. For example, once a fault is detected, a FTC controller may reconfigure its parameters or bring redundant components online. These constraints can eat into the costs of orchestrating a faster computational system, additional subsystems for task characterization, and more efficient algorithms for faster adaption. It is therefore prudent to address the latter concern, as it requires minimal changes in the hardware of the system.

The objective of this work is to design transfer learning approaches, such that a reinforcement learning controller is able to efficiently adapt to a target task without the explicit knowledge characterizing it (without fault detection). Specifically, reusing extant policies and training them on new tasks by efficiently sampling their dynamics.

The following research goals are set out for this work. In brief, prior policy/policies are reused in efficient ways. Instead of the trivial approach of applying source policies on the target task until it converges or the highest performer is found, effort is made to reuse a policy already preconditioned to perform well:

Policy reuse of existing policy by pre-training it on synthesized data of the target task.

Policy sampling from multiple past policies such that the selected policy improves performance on the target task. This is addressed in section 4.3.

We propose a novel adaptive reinforcement learning control approach for fault tolerant control of degrading systems that is not preceded by a fault detection and diagnosis step. Therefore, *a priori* knowledge of faults that may occur in the system is not required. The adaptive scheme combines online and offline learning of the on-policy control method to improve exploration and sample efficiency, while guaranteeing stable learning. The offline learning phase is performed using a data-driven model of the system, which is frequently updated to track the system's operating conditions. We conduct experiments on an aircraft fuel transfer system to demonstrate the effectiveness of our approach. In the context of the approach in 3.1,

$$f_{reuse} = \texttt{PPO}(\pi_s, M_{t,\omega} \mid B) \tag{4.14}$$

In previous work, Ahmed et al. (2018), we developed a RL-based fault-tolerant controller to accommodate abrupt faults. Faults were simulated on a fuel transfer system of a C-130 aircraft. A value iteration algorithm was used with a polynomial action-value function to derive a control policy. The controller showed better robustness to Gaussian sensor noise in comparison to model-predictive control (MPC). The proposed algorithm learned a policy offline with *a priori* knowledge of faults applied to a system model.

In this work, we extend our previous work and develop an online adaptive RL control algorithm for degrading systems with the goal of extending remaining useful life of a system, or providing sufficient time to continue degraded but safe operations till the system undergoes maintenance. Online RL-based control offers several advantages. By learning from past experience, it can adapt to changing system dynamics without changing design-time parameters, unlike classic control approaches like proportional integral derivative (PID) control. Moreover, compared to MPC schemes, a complex non-linear and non-convex optimization problem does not have to be solved online. This implies that the computational power required may be significantly less for RL-based control approaches during online operation.

The main contributions of this paper are the following: (1) the use of RL to learn a fault-tolerant controller that does not require *a priori* knowledge of faults, or a fault detection and diagnosis step; and (2) the combination of online and offline policy learning to improve exploration and sample efficiency while guaranteeing stable learning. We demonstrate the effectiveness of the proposed approach by applying it to a complex non-linear system under different degrading conditions. We believe that this is a first application of an online/offline RL control scheme to FTC.

The paper is organized as follows. First, the RL principles and the learning algorithm employed in this work are described. Then, the physics-based hybrid model of the operations for an aircraft fuel transfer system is presented. We devise methods for seeding different incipient faults into the system. This is followed by a description of the FTC controller design using a mixed learning RL approach. The experiments and results are presented next to demonstrate empirically how the proposed controller adapts to different incipient fault scenarios. In the last section, we present the conclusions and directions for future work.

4.2.1 Proximal Policy Optimization

In policy gradient algorithms (Sutton et al., 2000), the policy function is parameterized $\pi_\theta(x) : x \to u$ and learned through gradient ascent on the expected returns. The magnitude of change in policy parameters from previous version θ_k to the next θ_{k+1} is governed by the step size α.

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta \mathbb{E}[V^{\pi_\theta}(x)] \tag{4.15}$$

PPO is a policy gradient algorithm that can be used for systems with either discrete or continuous action spaces (Schulman et al., 2017). PPO is an actor-critic algorithm, where the policy function $\pi_\theta(x)$ and value function $V_\theta(x)$ are both parametrized. By modeling $V^\pi(x)$, the algorithm does not need to play out multiple trajectories of a policy to estimate a state's value. PPO can be classified as an on-policy reinforcement learning method because the updates of the policy are based on the experiences collected under the up-to-date policy. Learning through policy gradient methods is challenging because they are sensitive to the choice of update step size, i.e. too small a step makes convergence extremely slow (requiring thousands of interactions with the environment), and too big a step makes them prone to divergence. Thus, researchers have sought strategies to make the step size sensitive to the learning process. This is the idea behind PPO. PPO differentiates itself from other on-policy, policy gradient algorithms like Trust Region Policy Optimization (TRPO) by having a computationally simpler estimation of the step size.

PPO includes mechanisms to constrain the update step size while using first-order optimizers, like the gradient descent method, by formulating the following objective function:

$$\begin{aligned} \theta_{k+1} &= \arg\max_{\theta} \mathbb{E}_{x,u\sim\pi_{\theta_k}}\left[L(x,u,\theta,\theta_k)\right] \text{ s.t.} \\ L(x,u,\theta,\theta_k) &= \min\Bigg[\\ &(R^{\pi_{\theta_k}}(x) - V_{\theta_k}(x)) \cdot \frac{\pi_\theta(u|x)}{\pi_{\theta_k}(u|x)}, \\ &(R^{\pi_{\theta_k}}(x) - V_{\theta_k}(x)) \cdot clip\left(\frac{\pi_\theta(u|x)}{\pi_{\theta_k}(u|x)}, 1-\varepsilon, 1+\varepsilon\right)\Bigg], \end{aligned} \tag{4.16}$$

where ε, called the penalty coefficient, is the hyper-parameter that controls how much the new policy is allowed to differ from the old one. If the probability ratio between the new policy and the old policy falls outside the range $[(1-\varepsilon),(1+\varepsilon)]$ then the clip function is applied to cut-off the update step size.

4.2.1.1 Off-policy and On-policy updates

For learning a control policy, the choice of off-policy versus on-policy updates is related to the bias-variance trade-off (Sigaud and Stulp, 2019). Off-policy RL algorithms, such as those based on Q-learning, perform policy updates by reusing samples from previously collected experiences generated by a different policy, which are typically stored in a replay buffer. Off-policy updates allow for greater exploration of states, but such updates introduce a bias, especially for time-varying systems, where the experiences used for the updates may have been generated under a different operating condition for the system. The bias in the update step can make the policy sub-optimal or even divergent. Conversely, on-policy RL algorithms allow for more stable learning. However, they are less sample efficient since more interactions with the environment are required to converge.

4.2.1.2 Offline and online updates

In this paper we propose a mixed learning scheme. We employ a data-driven model of the system. The same on-policy learning algorithm (PPO) is used to improve the current policy but the interactions are performed offline with the system's model periodically to improve the exploration process. The model is updated (re-learned) at frequent periodic intervals to reflect changes in the system as it degrades over time. Employing a model as an offline environment allows the controller more interactions to experience system behavior and feedback. Thus, it is more sample efficient.

4.2.2 Approach

The control loop is discussed in this section. This work proposes an on-policy reinforcement learning-based control loop with a mixed learning scheme. The control policy is updated based on interactions with the real system (online). Intermittent periods of offline updates are also carried based on interactions with a data-driven model of the degrading system. The model is periodically updated to account for changing conditions in the system due to degradation. The controller logic is presented in Algorithm 5, with the overview illustrated in figure 4.4.

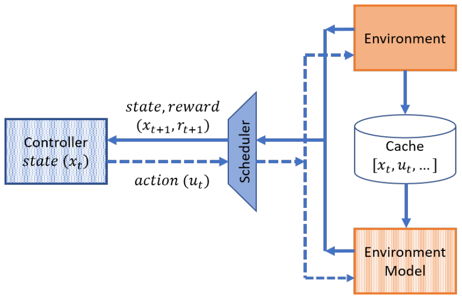


Figure 4.4: An overview of the proposed approach. The controller learns from the system and its data-driven model model to improve sample efficiency. Online control may proceed while the controller trains offline on the model in the background.

During operation, the controller for the fuel transfer system follows the derived policy function. Each interaction is cached. After t_{update} interactions, the policy function is updated online. After an interval t_{online} interactions, a data-driven model of the system is learned from cached experience. Offline, the controller updates its policy using the model for $t_{offline}$ interactions.

Using an offline learning phase addresses the problem of sample inefficiency of on-policy reinforcement learning algorithms. This is due to two reasons: (1) during operation, the size of experience of on-policy controllers is limited by the frequency of interactions with the system; and (2) on-policy controllers, by design, are restricted to applying a single policy during learning and operation. Therefore, they cannot benefit from sampling alternative behaviours.

Given the availability of an accurate model of the system, the controller is free to interact at a frequency of its choosing, bounded by two criteria: (1) that a dynamic model of the system is sufficiently accurate to derive an "optimal" policy – degradation of components in the system may cause the model to become inaccurate, so the model is re-learnt at frequent periodic intervals. And (2) computational resources.

A fully connected neural network is employed as a system model. The next state of the system solely depends on its current state and action taken. The sampling frequency of the data is high enough to capture system dynamics and there are no time delays. Furthermore, the state and action spaces are compact. This satisfies the requirements of the universal approximation theorem for neural networks (Cybenko (1989)).

4.2.3 Experiments

4.2.3.1 Environment Formulation

The fuel transfer model is considered as the controller's environment. An episode begins when the fuel tanks are at capacity, i.e., at the beginning of an aircraft flight, with the assumption that the system is operating nominally. The episode ends when total fuel in the tanks is less than the total demand from engines, implying that the pilot would have maneuvered the aircraft to land just before this situation occurred, which would cause the aircraft to crash. The environment is run for 10 intervals of $t_{online} = 512$ steps each. A single interval contains multiple episodes. After each interval, the system is degraded. For offline learning, the controller interacts with the model for an interval of $t_{offline} = 2048$ steps.

The objective of the controller is to maintain aircraft stability (i.e., center of gravity) even under degrading fault conditions in the aircraft. The reward function, defined in equation 4.17, is comprised of three variables: (1) centre of gravity (r_{cg}), (2) variance of fuel distribution r_{var}, and (3) the proportion of open valves (r_u). A centre of gravity close to the longitudinal axis of the aircraft contributes to lateral stability. Similarly, concentration of mass about the wing tips for forward-swept wings makes the plane more stable (Goraj and Zakrzewski (2005)). Finally, it is desirable to have a minimal number of valves open for fuel transfer so that fluid mass does not shift when a plane maneuvers in air. Fuel tanks were assumed to be equally spaced, unit distance apart from each other about the fuselage axis. The moment arms of fuel tanks are given by $\vec{m} = (-3, -2, -1, 1, 2, 3)$.

The reward function (equation 4.17) prioritizes a neutral centre of mass. $\odot$ denotes the Hadamard product. Only when centre of mass is close to the axis is the variance of fuel distribution rewarded. Figure 4.5 illustrates how reward changes over an episode.

$$
\begin{aligned}
r_{cg} &= \sum (\vec{x} \odot \vec{c} \odot \vec{m}) / (\vec{x} \cdot \vec{c}) \\
r_{var} &= \text{var}_{\vec{x} \odot \vec{c}} (\vec{x} \odot \vec{c} \odot \vec{m}) \\
r_u &= \bar{u} \\
r &= \left(1 - \frac{|r_{cg}|}{\max \vec{m}}\right) \cdot r_{var} - r_u
\end{aligned}
\tag{4.17}
$$

4.2.3.2 Data-driven System Model

A hyperparameter grid search was conducted to obtain the most accurate model architecture for the fault-free fuel transfer system. Training data was generated from the model for 50 episodes. Random actions were

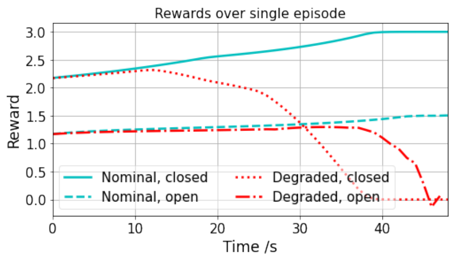


Figure 4.5: Comparison of rewards per step over an episode between nominal and degraded operation with open and closed valves. The left-most engine is degraded. This causes fuel on left tanks to drain faster. As the centre of gravity shifts to the right, the contribution of variance is scaled down.

chosen, which persisted for at most half an episode length. We noted that increasing the width of networks improved performance more than increasing depth. The best performance model had 2 hidden layers of 64 units each with Rectified Linear Unit (ReLU) activations. It was trained with an Adam optimizer learning algorithm with early stopping and a learning rate of 10^{-3}. A coefficient of determination of 0.996 was achieved with cross-validated data. Models used in the following experiments were initialized to the learned parameters from this model.

4.2.3.3 A single fault: engine degradation

For the trial, a degrading fault was introduced in the left-most engine, (E_1) with degradation factor $\vec{D_E}[E_1] = 20$. The system degraded linearly with each interval.

Figure 4.6 illustrates controller performance under progressing degradation over multiple intervals. When all valves are closed, the fuel distribution is no longer symmetric and the centre of gravity shifts to the right. Average rewards per step rapidly decline with increasing degradation. However, fuel does not freely flow between tanks, so the variance remains large, hence the high initial rewards when faults are minor. When all valves are open, fuel is able to freely move between tanks. The distribution of fuel is kept symmetric about the longitudinal axis. This also means that fuel rushes to median tanks as they drain first. This is penalized by the variance term r_{var} in the reward.

Figure 4.7 shows control actions over one episode at the highest degradation. The RL agent maintains a better balance than when all valves are open or closed at high levels of degradation. Valves open for tanks on the opposite side to equalize fuel distribution, as incentivized by r_{cg}. Of note is how tank 5's valve is primarily opened up to transfer fuel to tank 3 (Open valves are illustrated by the white spaces in the tank level plots in figure 4.7). This allows tank 6 to maintain its fuel mass and earn the reward r_{var} for a high distribution variance.

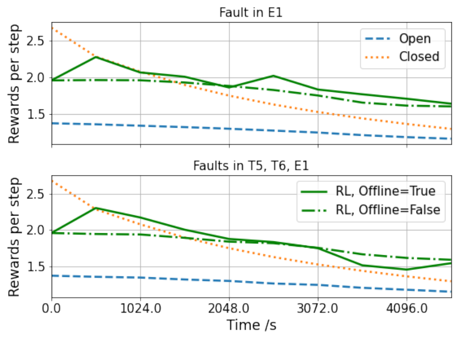


Figure 4.6: Average rewards per time step for each online interval as system progressively degrades under a single (top) and multiple (bottom) faults.

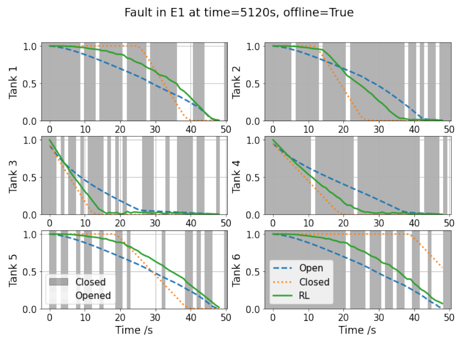


Figure 4.7: Control of fuel tanks when the leftmost engine is degraded.

4.2.3.4 Multiple faults: Engine and tank degradation

Another trial was carried out where the leftmost engine was degraded the same as before. Additionally the valves and resistances for tanks 5 and 6 were degraded with a factor $\vec{D_T}[T_5, T_6] = 20$. Figure 4.6 shows that the difference in performance between online-only and online+offline control schemes is less noticeable under multiple faults, but still outperforms baseline.

4.2.3.5 Aggregate performance

20 trials were carried out and the results averaged. For each trial, a valve and and an engine pump (chosen randomly) were assigned a degradation factor between $[10, 30]$. Figure 4.9 shows aggregate performance. Rewards earned by RL control do not show deterioration as degradation increases. RL control with offline learning consistently outperforms other approaches.

4.2.4 Conclusion

The fault-tolerant control problem has been applied to a wide range of transportation and industrial applications (Zhang and Jiang, 2008). This paper discusses an on-policy reinforcement learning-based fault-tolerant

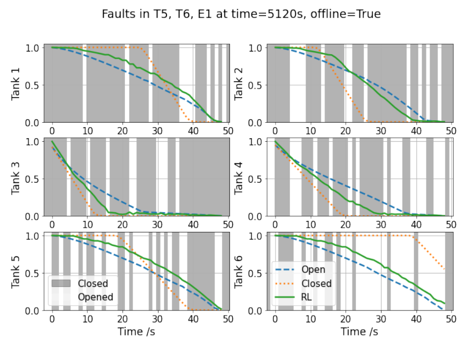


Figure 4.8: Control of fuel tanks when the leftmost engine and two right tanks are degraded.

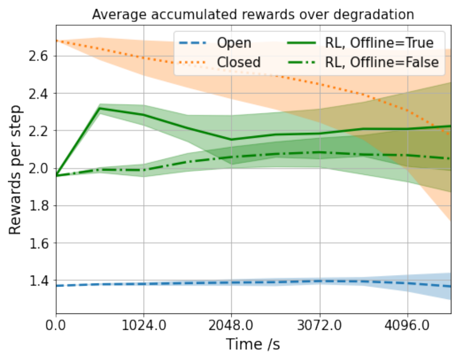


Figure 4.9: Average rewards per time step for each interval of t_{online} steps as system progressively degrades under multiple faults. A big contribution to offsets between rewards is r_u which rewards closed valves.

control scheme for incipient faults that does not require the fault detection and isolation step. Control is tested on a degrading model of an aircraft fuel transfer system. The controller is incentivized to maintain a high fuel distribution variance, centre of gravity close to longitudinal axis of the aircraft, and a minimal number of open valves. Under these constraints, RL control outperforms the baseline case when all valves are kept closed. On average, RL-based control shows a slower degradation in performance than the baseline cases (all valves closed or all valves open).

Previous methods to fault tolerance require a fault detection and isolation or performance monitoring to trigger the adaptation schemes (Li et al., 2019; Zhang and Jiang, 2008). The proposed approach does not rely on fault detection and isolation to learn control policies under degraded conditions. As the system degrades, the on-policy algorithm is able to optimize actions based on changing reward signals. In contrast to model-predictive control, an optimization problem is not solved at each time step. This points towards FTC applications where memory (for storing experiences) is abundant and the limiting factor is computational power.

There are several directions for future research. Fault-detection mechanisms can be explored, such that offline control re-learning is scheduled aperiodically whenever faults are detected. The control scheme can also be applied to the case of abrupt faults. Finally, a thorough study is merited into the choice of reward functions, model architecture, and convergence guarantees for a control policy across a general class of non-linear hybrid systems.[1]

4.3 Transfer reinforcement learning for fault-tolerant control by re-using optimal policies

The following work iterates upon this approach. Previously, a single policy was reused with the help of a stochastically learned model of the gradually changing task. Here a library of source policies was curated Π_s, learned from a population of MDPs $\mathscr{M}_s$ representing various faults. On preemption for adaptation for a new MDP M_t due to a fault, a policy heuristically determined to produce the best performance was reused. The evaluation was done on a buffer B of recent experiences collected by the running policy.

Our approach combines policy-reuse and consolidation algorithms to achieve learning speed improvement when new faults occur in the system. Our policy-reuse strategy finds a policy parameter initialization that leverages previous knowledge in the form of learned policies for past fault occurrences. We present the conditions under which our approach becomes effective and empirically demonstrate our approach on a test-bed of a 6-tank fuel transfer system of an aircraft.

The contribution of this work is to use the Jensen-Shannon metric, for probability distributions, for determining the relationship between policies. Furthermore it uses the predicted expected returns V_θ on the buffer of off-policy experiences as a heuristic for expected performance on M_t with a fault.

$$\begin{aligned} DIST_\pi(\pi_1, \pi_2) &= \texttt{JensenShannon}_{x,u\sim B}\left(P(u \mid x, \pi_1), P(u \mid x, \pi_2)\right) \\ L_t &= -\sum_{x,u\sim B} P(u \mid x) \cdot V_\theta(x, u) \\ f_{reuse} &= \arg\min_{\pi\in\Pi_s} L_t(\pi, B) \end{aligned} \tag{4.18}$$

4.3.1 Approach

The proposed FTC scheme takes over after an abrupt fault is diagnosed in the system. It maintains a set of policy parameters previously learned under different tasks. On encountering a new fault, an extant policy similar to the present conditions and expected to generate larger rewards is sampled from the set and substituted into the controller. Once initialized, the controller starts interacting with the new task and fine-tunes policy parameters according to the RL algorithm being used. The trained parameters are then consolidated in the set for future reuse.

[1]Source code for this work can be found at https://git.isis.vanderbilt.edu/ahmedi/airplanefaulttolerance/-/tree/ifac2020

4.3.1.1 Discriminating between policies

The choice to maintain a library of policies can be justified by the diverse modes of operations they cover. The policies change substantially with the learning task for each fault. The policies being randomly initialized and highly non-linear functions, their parameters cannot be used to measure similarity. Policies may also behave differently when subjected to two different tasks. So performance difference on a nominal task is not necessarily the same difference on a separate task.

Therefore, similarity between two policies π_1, π_2 is computed dynamically using probabilities of actions (p_1, p_2) extracted from a buffer $\mathscr{M}$ of recent states encountered and actions taken by the controller. One such measure is the Jensen-Shannon divergence, $JSD(\pi_1 \| \pi_2)$, Dagan et al. (1997), which is based on Kullback-Leibler (KL) divergence between two probability distributions $D(\pi_1 \| \pi_2)$. The divergence calculates the similarity between two probability distributions. In this case the distributions are the action probabilities of states in $\mathscr{M}$. In other words, the controller is asking the question: which policy in the library behaves most similarly under the same fault as the extant policy?

$$\begin{aligned}
\vec{p_{1,2}} &= \{\pi_{1,2}(x,u) : x, y \in \mathscr{M}\} \\
\vec{p_M} &= (\vec{p_1} + \vec{p_2})/2 \\
JSD(\pi_1 \| \pi_2) &= \left(D(\vec{p_1} \| \vec{p_M}) + D(\vec{p_2} \| \vec{p_M})\right)/2
\end{aligned} \tag{4.19}$$

Where π_M is the mixture of the two compared distributions. JS divergence is symmetric, unlike KL divergence. Therefore $JSD(\pi_1 \| \pi_2) \equiv JSD(\pi_2 \| \pi_1)$. Policies which produce a smaller divergence are likely to behave similarly and thus produce the same feedback. Such policies can be clustered or filtered out, to reduced the sample size in the library. Library pruning is left as a discussion for a later work.

4.3.1.2 Ranking optimal policies

Policies previously learned are ranked by how well they can be expected to perform under the new task at initialization. The policy estimated to yield this largest cumulative rewards is selected by the controller as optimal. The architecture of policy gradient algorithms like PPO is leveraged to sample optimal policies. PPO's policy is trained as a *critic* function that evaluates actions taken from states $V(x,u)$, in turn to train the *actor* function that outputs the probability of actions to take $P(u)$.

Let the current nominal task be M, and the controller policy be π. Let there be a library of policies trained on tasks (faults) previously encountered by the controller $\Pi = \{\pi_l : l \in L\}$. When a new fault occurs, the controller faces a new test task $M_t \sim \Omega$, which would have an optimal choice of π_{l^*} from Π. Let $\mathscr{M}_l$ be the buffer of experiences had a library policy π_l been used to collect observations in the aftermath of a fault. Whereas $\mathscr{M}$ is the buffer using the extant policy π. Then the selection of the optimal policy initialization is given by equation 4.20.

$$\begin{aligned} V(\mathscr{M}_l)_{\pi_l} &= \sum_{x,u \sim \mathscr{M}_l} V(x,u) \cdot \pi_l(x,u) \\ \pi_{l^*} &= \arg\max_{\pi_l \sim \Pi} V(\mathscr{M}_l)_{\pi_l} \end{aligned} \tag{4.20}$$

Equation 4.20 is used as a heuristic for comparison more than as a true approximation of value. The weighed sum emphasizes more valuable states from likelier actions over other state-action pairs in $\mathscr{M}$. It does not evaluate actions not taken by the extant policy.

It should be noted that the controller only has access to $\mathscr{M}$ under π. There is not enough time to deploy each policy in the library and collect a buffer of experiences for each. Therefore, for equation 4.20 to hold, the ordinal relationships using the substitution $V(\mathscr{M})_{\pi_l} \approx V(\mathscr{M}_l)_{\pi_l}$ for $l \in L$. Figure 4.10 demonstrates how policies trained on tasks similar to M_t will share intermediate or terminal states. Therefore the valuation of the π_l using $\mathscr{M}$ will be truer to the valuation had $\mathscr{M}_l$ been used. With a truer policy evaluation, the policy likely to perform better will be chosen. The following experiments bear these results by showing the utility of policy re-use and its limitations under faults of various similarities.

4.3.1.3 Test-bed for experiments

The RL test-bed is a six-tank fuel transfer system on the wings of an aircraft. Fuel is pumped from tanks to engines under a fixed schedule. The objective is to transfer fuel between tanks to keep fuel mass balanced about the longitudinal axis, to keep fuel mass concentrated at the extremities, and to conserve fuel mass against leaks. This is a hybrid system. The state space $x \in [0,1]^6$ constitutes of fuel levels in each tank as a fraction of their height. The action space is the status of valves on each tank $u \in \{[0..1]\}^6$. The environment is parametrized by the tank geometry, valve resistances, and engine fuel consumption rates.

The model can experience multi-modal faults. Each faulty state represents a separate task to be learned by the controller.

- Valve faults leave a single or a pair of valves unable to close fully.
- Leak faults cause fuel to leave tanks besides through engine pumps or valves.
- Pump faults cause fuel supply to engines degrade for a tank.

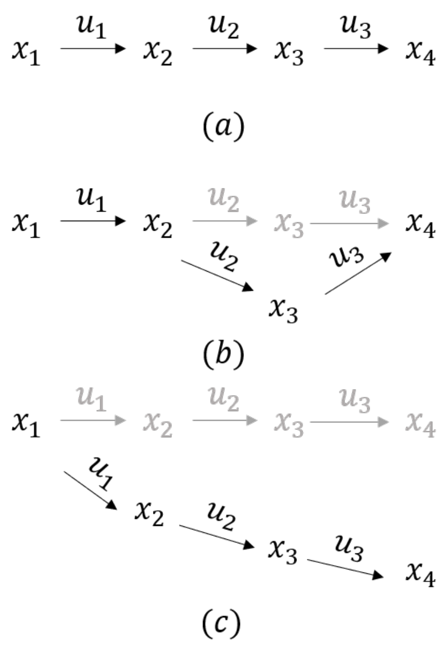


Figure 4.10: Illustration of buffered experiences using different policies under M_t. Dark sections represent stored experiences. Faint sections show the difference with the extant policy π under M_t. $(a) : \mathscr{M}$ using π. $(b) : \mathscr{M}_l$ where the task to train π_l is similar to M_t. $(c) : \mathscr{M}_l$ but where the task to train π_l is dissimilar to M_t.

The evaluation metric for training and testing is the reward function. It is made up of several sub-components, each pertaining to operating features that should be optimized:

- **centre** is the deviation of centre of gravity from the longitudinal axis. Closer to zero is better.
- **activity** is the average number of valves that are open. Closer to zero is better to prevent unnecessary mass transfer.
- **spread** is the variance of mass distribution about the longitudinal axis. Higher is better for forward-swept wings.
- **level** is the average fuel in tanks. Higher is better.
- **deficit** is the engine fuel demand unmet my pumps in tanks. Closer to zero is better.

Ultimately, the components form the reward function in equation 4.21.

$$r = \left(level + (1 - |centre|) + \frac{spread}{2} - \frac{activity}{4} \right) \cdot (1 - deficit) \tag{4.21}$$

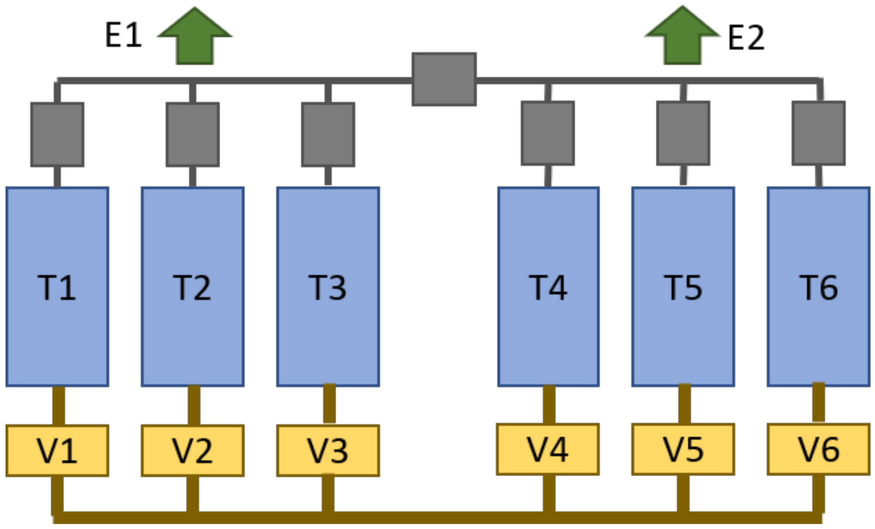


Figure 4.11: The fuel tank system. Pumps are in grey, tanks in blue, and valves in yellow.

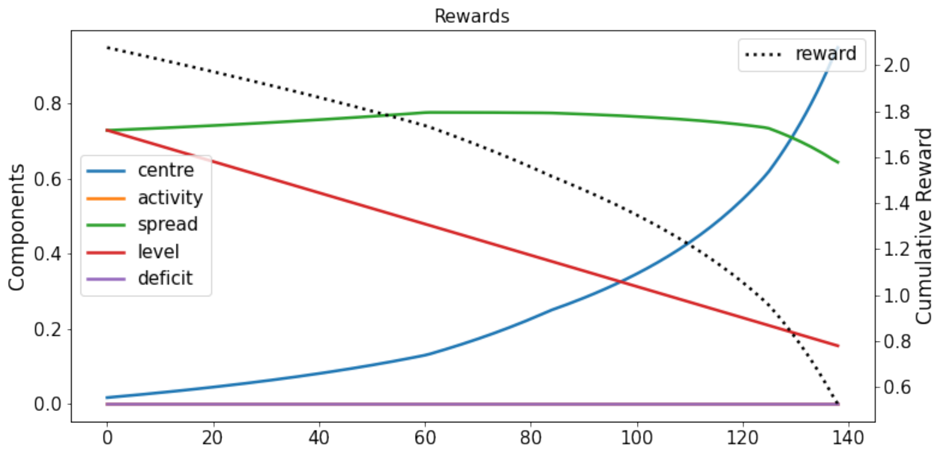


Figure 4.12: A breakdown of various components in the reward signal over a single episode. For the nominal case, components such as `deficit` and `activity` are zero.

4.3.2 Results

A library of policies trained on faults was learned. Sixteen faults were used to measure the pairwise similarities between their corresponding policies. Table 4.3 shows the four modes of faults, each with four instances, possible in the system. A fault was chosen without replacement and introduced into the environment. This created a new MDP representing the target task M_t. A proximal policy was chosen from the policies corresponding the the remaining faults, corresponding to the source tasks L. The proposed approach was evaluated using the area under curve (AuC) of reward signals over episodes of operation after a fault. As a benchmark, the performance of the same policy before the fault was charted as the “Vanilla” approach, along with the averaged performance of choosing every other policy in the library, labelled "Average". The code for these experiments is available at https://git.isis.vanderbilt.edu/ahmedi/AirplaneFaultTolerance.

Table 4.3: A description of the designation and sizes of various modes of faults.

Type	**Number**	**Description**
valves25	4	Pair of valves stuck at 25%
valves50	4	Pair of valves stuck at 50%
leak	4	Leak in one fuel tank
pumps	4	Pump stuck at 10%

4.3.2.1 Measuring proximal policies

The Jensen-Shannon metric, as introduced in section 4.3.1.1, was used to calculate distances between policies for each fault. The metric was visually compared against the cosine, Euclidean, and L1 norm metrics. Figure 4.13 shows normalized pairwise distances between policies trained on faults for each similarity metric. Of note are the lower mutual distances for faults pertaining to valves (indices 1-8) and leaks (indices 9-12). They are delineated against pump faults (indices 13-16). One explanation for this is that the fluid dynamics of valve and leaks are similar: the flow rates depend on tank pressure and resistances. Whereas the flow rates for pump faults are modeled as constant.

Alternatively, using multi-dimensional sampling over the pairwise distances, the policies were projected into Cartesian space . Figure 4.14 shows a possible arrangement in two dimensions. By all distance measures, the overlap between valve and leak policies is larger than with pump policies.

4.3.2.2 Single-mode faults

The approach was first tested by having only a single category of fault in the library and occur in the system. In this and the following subsections, the domain of faults were sampled from [`valves50, leaks, pumps`] as described in table 4.3. Results are presented in table 4.4. For the valve and leak faults, the proposed approach produces higher overall rewards. This translates into an aerodynamically preferable mass distribution and lesser number of valves open. For pump faults, our approach is the least rewarding. From figure 4.13 and 4.14, pump faults have one of the larger mutual distances inside their category and against other categories of faults. This means that pump faults' policies are significantly different from each other, and therefore the estimate in section 4.3.1.2 may have been inaccurate.

4.3.2.3 Novel Faults

To evaluate the case where the encountered fault was completely novel or the learned policies were very different, the set of faults on the system and learned policies in the library were kept disjoint. Under novel

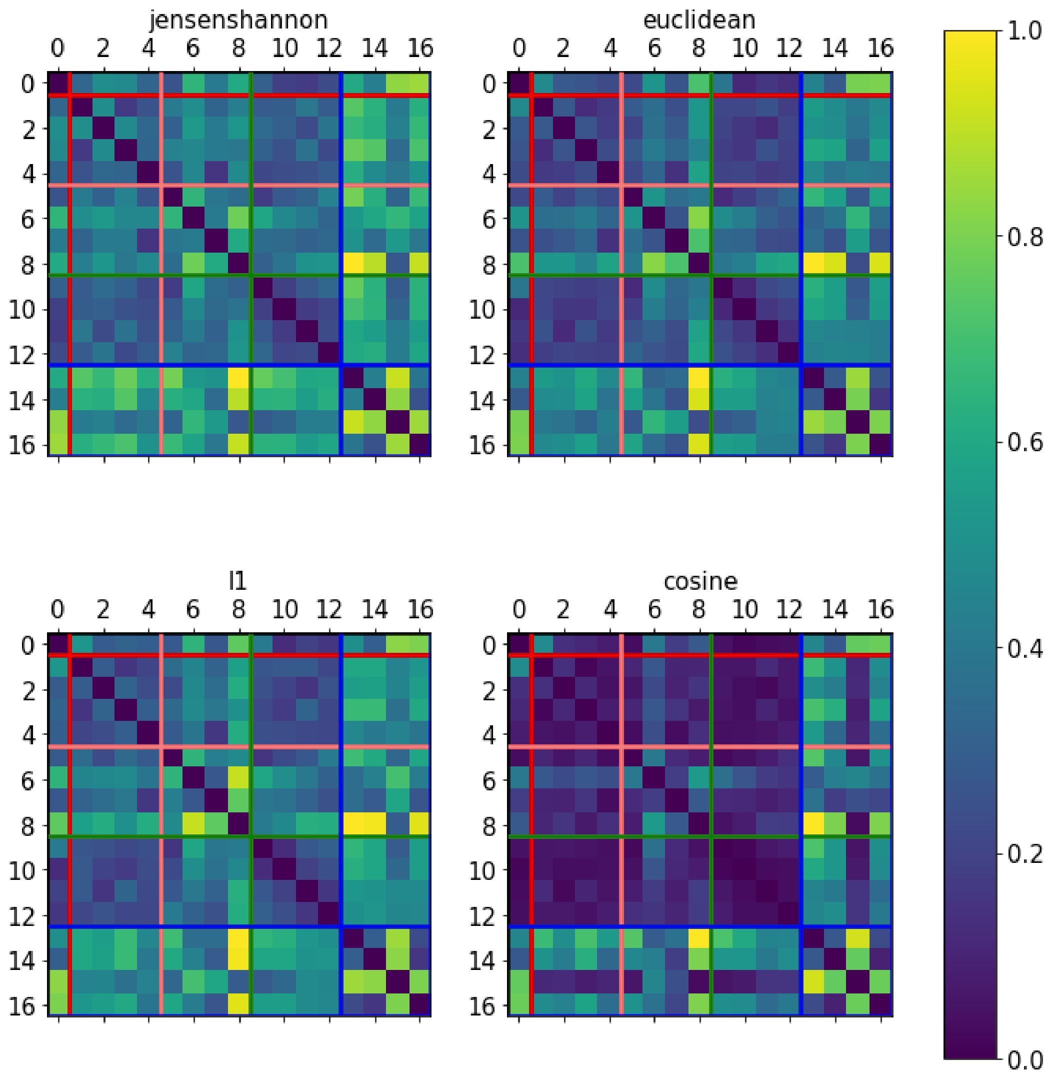


Figure 4.13: Pair-wise distances between the nominal policy (index=0) and faults (1-16) normalized to $[0, 1]$. The demarcating lines represent different fault modes as described in table 4.3.

faults, the performance was poorer. This reinforces the hypothesis in section 4.3.1.2 that when the optimal sequence of actions and states for a new task is significantly different from the buffer of experiences generated by a sub-optimal policy, the corresponding evaluation may not be useful for ranking.

4.3.3 Conclusion

In this paper we presented a use case of transfer learning for fault-tolerant control of a hybrid system. By exploiting experiences from similar faults, the controller was able to achieve better performance than using its current policy for learning the new task. In cases where the encountered fault was novel, or where the category of that fault resulted in dissimilar policies, the achieved performance often lagged behind simply learning using the extant policy.

Table 4.4: Rewards from single-mode faults where library and encountered faults belong to a single fault mode.

valves50	AuC					
	Deficit	Centre	Activity	Spread	Level	Total
Ours	**0.0325**	**262**	**28918**	**32158**	**17948**	**65735**
Vanilla	0.0432	367	29794	31867	17945	65298
Average	-	-	-	-	-	64598

leaks	AuC					
	Deficit	Centre	Activity	Spread	Level	Total
Ours	52.39	**-5.37**	**28512**	**31424**	**18013**	**64847**
Vanilla	**39.86**	-15.51	30161	31139	18008	64583
Average	-	-	-	-	-	64568

pumps	AuC					
	Deficit	Centre	Activity	Spread	Level	Total
Ours	1012	**-18.2**	32342	29429	17634	61724
Vanilla	**947**	-41.4	**29557**	**29487**	**17643**	62507
Average	-	-	-	-	-	**64568**

Table 4.5: Rewards from novel faults where library and encountered faults belong to different fault modes. Names are categories of the novel fault.

valves50	AuC					
	Deficit	Centre	Activity	Spread	Level	Total
Ours	0.073	-35.5	**28572**	**32181**	**17946**	**65919**
Vanilla	**0.049**	**1.56**	29139	31858	17940	65563
Average	-	-	-	-	-	64686

leaks	AuC					
	Deficit	Centre	Activity	Spread	Level	Total
Ours	55.78	1111	33874.25	**30892**	**18008**	63394
Vanilla	**37.7**	**995.9**	**30181.75**	30812	18007	64398
Average	-	-	-	-	-	**64460**

pumps	AuC					
	Deficit	Centre	Activity	Spread	Level	Total
Ours	203	803	30403	**30485**	17915	63860
Vanilla	**171**	**763**	**28168**	30472	**17916**	**64459**
Average	-	-	-	-	-	64030

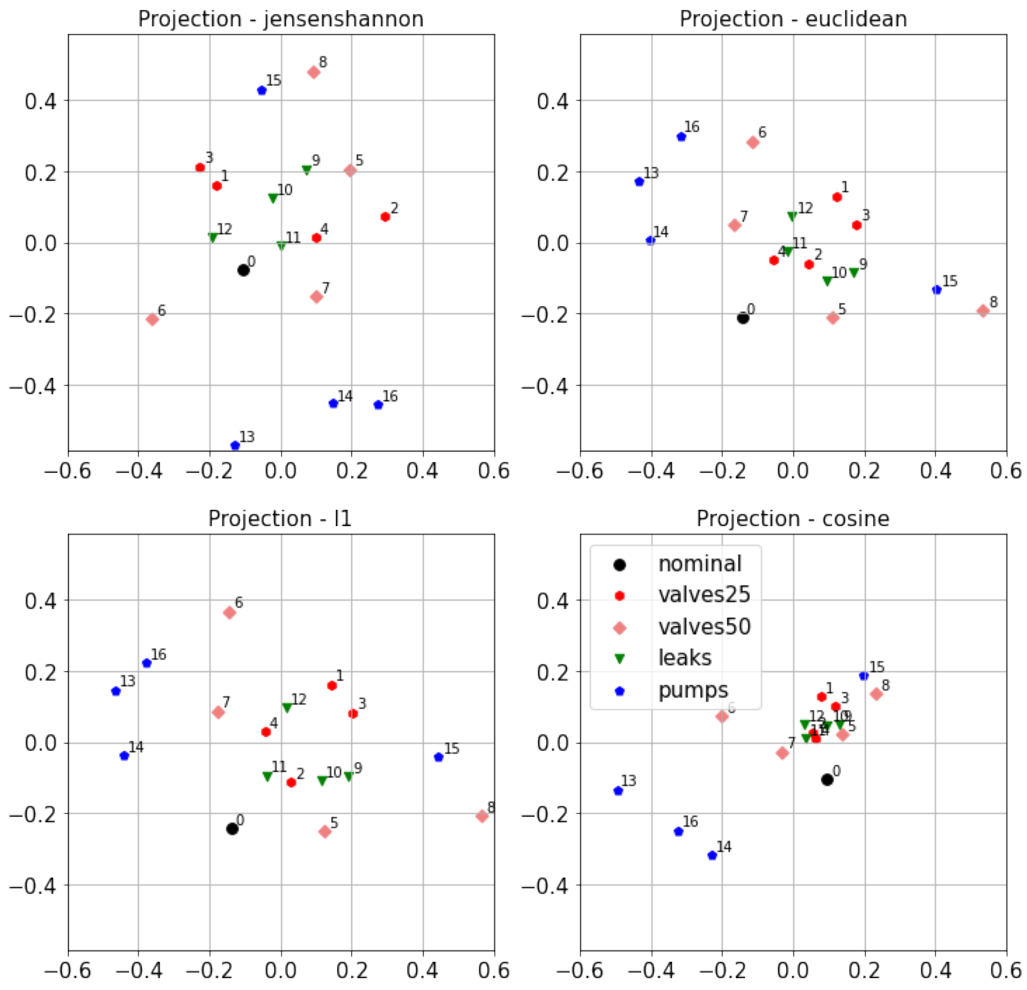


Figure 4.14: Using multi-dimensional sampling on pairwise distances between policies for four faults in each of the four fault modes on 2 principal axes.

Thus, the choice of a dynamic policy ranking metric which is robust against dissimilar policies poses an interesting problem for future work. Equally vital is the pruning of the library of policies to keep the sample size small while preserving breadth of experiences. This paper presented some preliminary solutions by way of similarity measures to discriminate policies.

4.4 Complementary Meta-Reinforcement Learning for Fault-Adaptive Control

Similar to the problem defined in section 4.3, sample efficiency is a critical aspect of timely adaption using data-driven methods. However it is possible that reusing a prior policy may lead to a sub-optimal or even a negative transfer. In this case the policy parameters may need to be modified before applying them to the transfer task. Such an initialization may be achieved by pre-training a single policy on a buffer of new experiences. But what if there are multiple candidate policies? Can experiences from a variety of tasks be distilled into an initialization for the target task?

The objective of this work is to investigate approaches to use multiple policies for transfer to a target task. The specific goals of this research are:

Meta-learning a new policy initialization π_t^0 from a library of past policies Π_s.

Minimizing chances of sub-optimal transfer and sampling time of policies by curating the library with the most "useful" policies.

In our past work Ahmed et al. (2020), we developed data-driven models to supplement experience with the real system and simulate faults. In this work, we employ meta-RL for faster adaptation of the RL algorithm to collected data samples. Our approach is not dependent on the time-consuming step of a data-driven model being learned first, however one can be used. It is assumed that a policy $\pi_{t,nominal}$ is controlling an MDP. When a fault occurs, the policy is reinitialized to adapt to that new target task M_t. Unlike MAML, a set of source tasks is not explicitly sampled for generating experiences for the meta-update. Interacting with multiple MDPs adds to computational cost. Instead, one buffer of experiences B, collected from the target MDP M_t after a fault, is evaluated on a set of source policies Π_s and the current policy $\pi_{t,nominal}$ of the target MDP. The gradients of the meta update are then with respect to those valuations, and used to update the current policy of the target MDP.

$$
\begin{aligned}
L_t &= -\sum_{x,u \sim B} P(u \mid x) \cdot V_\theta(x,u) \\
f_{distil} &= \texttt{MAML}(\Pi_s \cup \{\pi_{t,nominal}\} \mid B) \qquad (4.22)
\end{aligned}
$$

Building on prior work, the approach optionally can generate synthetic trajectories from a stochastic model of the task. To add a safety constraint on performance, the target policy is given a nominal RL update step using the same batch of experiences $\pi_{t,nominal}$. Before the policy distilled from source $f_{distil}(\pi_t^0) \rightarrow \pi_t$ is initialized on the target task, it is compared against the nominally updated target policy using a model of the MDP trained from the same buffer of experiences. Whichever policy has a higher valuation under this sanity check is used on the target MDP:

$$
\begin{aligned}
\pi_t' &= \texttt{PPO}(\pi_{t,nominal}, M_{t,\omega} \mid B) \\
\pi_t^0 &= f_{distil}(\pi_{t,nominal}, M_{t,\omega} \mid B) \\
f_c(\pi_t) &= \begin{cases} \pi_t^0, & \texttt{if } V(x \mid u \sim \pi_t, M_{t,\omega}) \geq V(x \mid u \sim \pi_t', M_{t,\omega}) \\ \pi_t', & \texttt{otherwise} \end{cases}
\end{aligned}
\tag{4.23}
$$

4.4.1 Model-Agnostic Meta Learning

Meta-learning seeks to speed up a machine learning process through introspection. Essentially, it learns how to learn. In a RL context, meta-learning seeks to quickly adapt a policy trained on one process to another.

Model-agnostic Meta Learning (MAML) Finn et al. (2017b) speeds up the optimization of any model learned through gradient updates. It does so by running an inner *introspective* loop within each iteration of a gradient update to the model's parameters, which is designated as the outer loop. In the inner loop, variants of the process are sampled as $p^i \sim P$. The current model parameters θ are then optimized by training for several interactions on each p^i using gradient ascent to yield θ^i. At the end of the inner loop, gradients on a test set of interactions are computed. In the outer loop, the update to θ is a weighed aggregate of the test gradients from the inner loop. That is, the training step for the outer loop is based on the test step of the inner loop.

Meta RL for FTC is a nascent field. Recently, Nagabandi et al. (2018) used used model-based RL for quickly adapting control to changed system dynamics. They used MAML and a recurrent network as two approaches to develop a meta-update rule for the system model parameters. In our case, however, we apply MAML towards updating the policy parameters. Alternatively, Sæmundsson et al. (2018) train a model to predict a latent representation of the system. The latent variable is fed to the agent as a conditioning variable to represent changed dynamics. Wang et al. (2016) use a recurrent neural network to train a controller on a population of related systems. The controller, being recurrent, has memory of this experience, and therefore learns an internal function to transition between systems as they change.

4.4.2 Complementary Meta-Reinforcement Learning

4.4.2.1 Importance sampling in PPO

Parameter updates at each iteration in PPO are dependent on experienced rewards under the latest policy. This is known as *on-policy* RL. This approach is sample inefficient because new trajectories of interactions need to be obtained for each version of θ. A way around this is to use *importance sampling* in the gain function. By modeling the policy as a stochastic function over actions, $\pi_\theta(x \mid u)$, the relative probabilities, known as importance ratios, of the same trajectory under different policies can be obtained. Thus, the gain function can reuse the same batch of experiences to update the current iteration of parameters θ' by weighing cumulative rewards. Equation 4.24 shows how importance sampling reuses experiences collected under θ_k for the next iterations of policy parameters $\theta_{k+i} : i \geq 0$. The learning rate is α.

$$G = \mathbb{E}_{x_0 \sim X}\left(\Pi_{t=0}^{\infty} \frac{\pi_{\theta_{k+i}}(x_t|u_t)}{\pi_{\theta_k}(x_t|u_t)}\right) J_{\pi_{\theta_k}}(x_0, u_0)$$

$$\theta_{k+i+1} = \theta_k + \alpha \cdot \nabla_{\theta_{k+i}} G \tag{4.24}$$

4.4.2.2 Problem Formulation

The problem of the controller is thus: to exploit its past experiences with different processes, and sparse interactions under new process dynamics p' to quickly converge to a locally optimal policy. The proposed approach for adaptive control operates under the framework depicted in figure 4.15. The adaption pipeline can either be preempted by fault detection, or happen periodically.

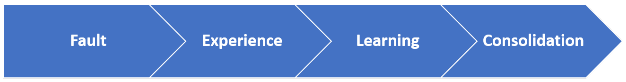


Figure 4.15

The adaption step begins with a *fault*. The fault is abrupt, causing a discontinuous change in process dynamics $p \rightarrow p'$. The MDP representing the system has changed. In the aftermath of a fault, a controller continues to interact with p' and records states, actions, and rewards in a memory buffer $\mathscr{M}$ using its current policy parameters θ_k. Once sufficient interactions t_{update} have been buffered, the controller attempts to initialize new parameters θ' from its memory, and then fine-tunes them to θ_{k+1} by interacting with the new process. Once learning is complete, the controller consolidates the newly learned policy with its prior policies. Thus, when a new fault occurs, it is able to exploit its past experience and adapt faster.

The *learning* phase consists of two stages: the meta-update using the memory, followed by iterations of any choice of a gradient-based reinforcement learning algorithm on the new process. During the meta-update, the controller uses its consolidated prior experience to initialize new policy parameters. The controller can also generate a data-driven model of the system to supplement sample inefficiency of RL. After that, the parameters are iteratively updated by the RL algorithm through interactions with the actual system.

Consolidation of knowledge happens via maintaining a complement of prior policies $\mathscr{C} = \{\theta \mid \pi_\theta\}$. The set of policies is periodically pruned to ensure that they capture diverse behavior but are small enough to evaluate within time constraints.

4.4.2.3 Policy meta-update

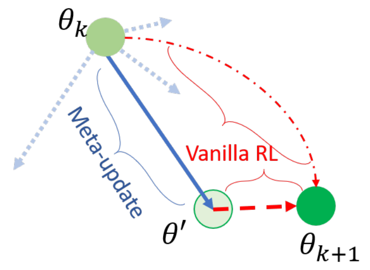


Figure 4.16: The meta-update initializes policy parameters closer to an optimum, after which RL converges faster to a solution. The meta-update depends on the aggregate gradients of policies in the complement. The gradients are calculated from samples from a data-driven process model updated from a buffer of recent experiences, or the buffer itself. The meta-update step from θ_k to θ' is described in algorithm 7.

Our approach mirrors MAML in that there is an outer update loop for the main policy parameters. It depends on the gradients of the test error on the inner loop. We diverge in our formulation of the inner loop. In MAML the inner loop samples random processes from a population $p^i \sim P$ defining the MDP. It uses those samples to derive intermediate parameters θ^i from the single starting parameter θ_k. We forego sampling processes anew to derive such intermediate parameters, and instead exploit the history of the controller's experience. In other words, MAML evaluates multiple processes on a single set of parameters. We propose to evaluate a single process on multiple sets of parameters.

Prior to the meta-update, a memory $\mathscr{M}$ of interactions under the new process is buffered. The meta-update step assumes a complement $\mathscr{C} = \{\theta \mid \pi_\theta\}$ of prior policies trained on the system under different faults. This foregoes the need of sampling an altogether new set of processes for the meta-update. The complement of polices is then trained for a few steps K_{in} to yield an updated set of meta-parameters. Finally, the test error of the meta-parameters on the process is used to update the outer loop's policy parameters.

Optionally, as a guard against a sub-optimal initialization $\theta_k \to \theta'$, θ_k is also concurrently updated using standard RL without meta learning to a baseline parameter $\theta^b_{k_{out}}$ for each iteration k_{out} of the outer update loop. Finally, the meta-learned parameters and baseline parameters are evaluated on a provided process model p_m. Whichever performs better is returned as the new initialization θ'.

Evaluating policies from $\theta^i \in \mathscr{C}$ necessitates new interactions with the changed process p'. This can be achieved by learning a data-driven model p_m of the process using $\mathscr{M}$. However, this introduces an additional computational load on the meta-update step. An alternative approach, already inherent in PPO, is to forego a model altogether and instead use importance sampling (equation 4.24) to adjust the gain with respect to θ^i.

With importance sampling, the returns already calculated on p' under θ_k stored in $\mathscr{M}$ can be weighed by the relative probabilities of actions under θ^i. This process is delineated in algorithm 7 and figure 4.17.

4.4.2.4 Population of complement

The final step of the approach is to store the newly learned parameters for future reference. The complement of policies should be populated with policies such that it maximally spans the parameter space. Policies should be different enough so that the meta-update has a greater likelihood of adapting to novel faults. The difference between policies is evaluated on the memory of interactions collected by the controller. Each policy in $\mathscr{C}$ generates a probability for actions stored in $\mathscr{M}$. KL-divergence between the probabilities is used as a metric of difference. The total divergence of each policy from the rest of the complement becomes a score of a policy's uniqueness. Given a complement size $|\mathscr{C}| \leftarrow s$, the s most unique policies are kept as new members of $\mathscr{C}$. Algorithm 8 goes through the process of selecting between the existing and newly learned policies to update $\mathscr{C}$.

4.4.3 Experiments

The algorithm was evaluated on a simulation of a fuel transfer system of an aircraft. The system is defined in greater detail in Ahmed et al. (2021). The objective is to maintain center of gravity, variance in fuel distribution, and closed valves to avoid unnecessary mass transfer. Faults can include increased valve resistances leading to low flow rates, and increased fuel consumption due engine faults.

A controller was first trained for 50,000 steps on the nominal system. At the beginning of a trial, a random fault occurred and the controller accumulated experience in memory $\mathscr{M}$. The controller then employed the meta-update step in algorithm 7 to initialize new policy parameters. Following that, the RL algorithm continued to learn on the new system. As a baseline, an RL controller was trained for $|\mathscr{C}| \times K_{in} \times K_{out}$ iterations on p_m, when p_m was provided, followed by learning on the new system p'. For all experiments, a first-order approximation of gradients $\nabla_{\theta'} G^i$ as documented in Finn et al. (2017b) is used.

First, the controller was tested with an empty complement of policies. Second, a complement of 3 policies under simulated faults on the system was generated. The complement was trained on faults in tanks 1, 3, and 5 and no engine faults. In both cases, the controller was tested on the system under random novel faults. The controller was allowed to adapt solely from buffered experiences after a fault, without learning a new system model.

Figure 4.18 shows performance with $\mathscr{C} = \{\varnothing\}$. Episodic rewards start off lower than but comparable to the baseline. They quickly recover and match baseline throughout. Of note is the low variance in episode rewards compared to the baseline. Figure 4.19 shows performance with a complement of 3 policies. The

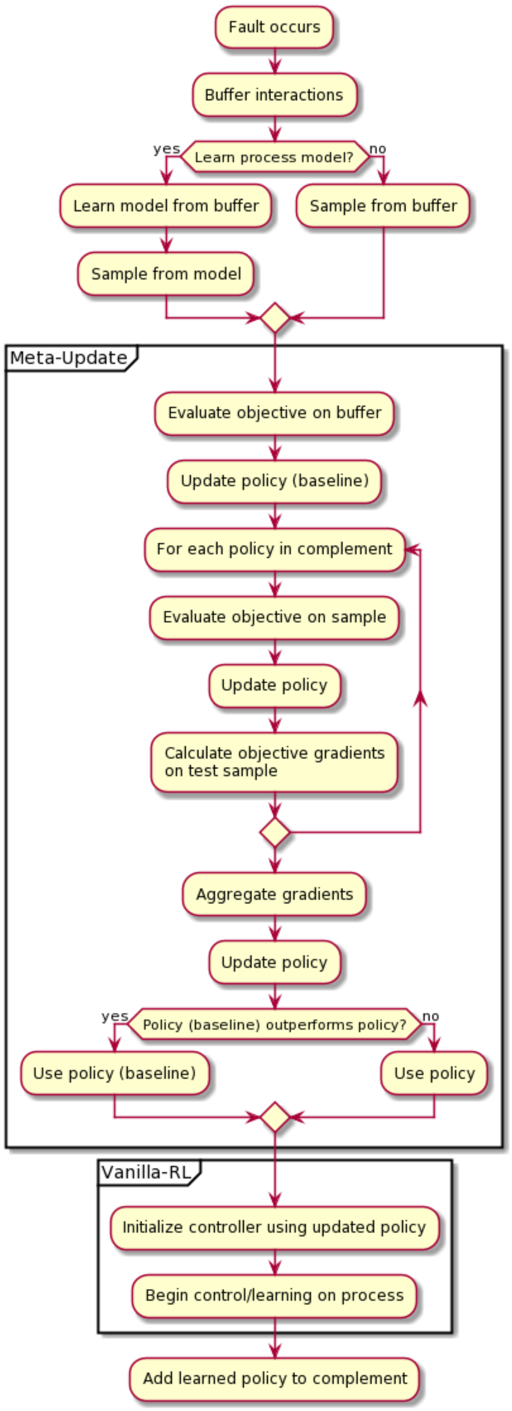

Fault occurs
Buffer interactions
Learn process model?
yes
no
Learn model from buffer
Sample from buffer
Sample from model
Meta-Update
Evaluate objective on buffer
Update policy (baseline)
For each policy in complement
Evaluate objective on sample
Update policy
Calculate objective gradients on test sample
Aggregate gradients
Update policy
Policy (baseline) outperforms policy?
yes
no
Use policy (baseline)
Use policy
Vanilla-RL
Initialize controller using updated policy
Begin control/learning on process
Add learned policy to complement

Figure 4.17: An overview of the complementary MAML algorithm in a FTC context. The meta-update step initializes the policy based on a complement of policies evaluated on the new process.

controller starts off with performance similar to the baseline, but quickly pulls ahead and converges to an optimum. The initialization using a populated complement allows the controller to converge to a solution faster.

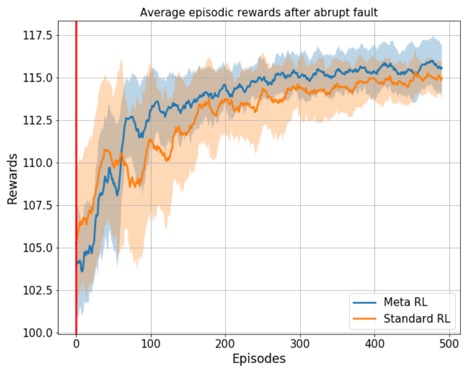


Figure 4.18: Episodic rewards after abrupt fault when there is no complement of policies available to the controller.

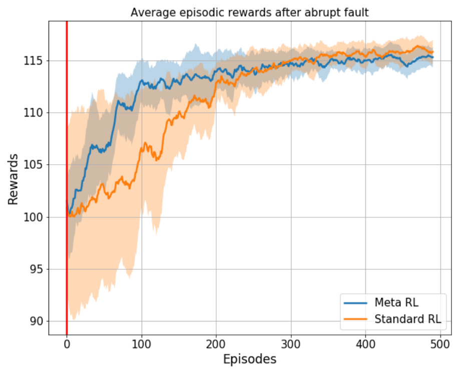


Figure 4.19: Episodic rewards after abrupt fault when there is a complement of 3 policies available to the controller.

4.4.4 Conclusion

We have proposed a meta-RL algorithm, which exploits a controller's past experience under faults to initialize parameters for a new policy under a novel abrupt fault. The meta-update can optionally use a data-driven model to mitigate sample inefficiency, or it can fall back to using importance sampling on buffered experiences to evaluate the complement under current conditions. The newly derived parameters are added to the complement if they are divergent enough from the members of the set, thus ensuring a diverse library of behaviors for faster adaption to new faults.

MAML can be sensitive to choice of model architecture, task, and hyperparameters Antoniou et al. (2018b). This merits further investigation on guarantees of convergence and optimality under faults. MAML can be further incorporated in our approach by using meta-learning to update the data-driven model itself. This should further reduce time taken to learn an updated model and the dependence on the size of the buffered data.

4.5 Bootstrapping transferable reinforcement learning controllers for HVAC domains

It can be the case that sufficient data are not available to train a data-driven controller for deployment in a real-world application. For example, RL policies rely on an interactive environment. And transfer learning approaches rely on well-defined source and target domains. Given a controls application relying on rule-based control with poorly defined dynamics, there needs to be some bootstrapping to learn the first iteration of a task which can be used for transfer.

The objective of this research is to develop a pipeline of operations to investigate utility of transfer-based control using reinforcement learning in a specific domain. As a first iteration, data-driven models of the MDP dynamics are built using available, but incomplete, descriptions of the system. Then the viability of transfer learning is investigated when comparing against benchmark controllers. In particular:

Modeling the cooling tower environment using a data-set representative of the range of possible operation.

Definition of the optimization problem and the state and action spaces. For instance the level of abstraction at which control will be exercised (low-level actuation vs high-level set-points).

Transfer of control when tasks are partitioned by exogenous conditions, sparsity of data, and the source of measurement.

Commercial buildings in the United States account for 18% of total energy consumption eia (2022b). Of that, a total of 47% is used for refrigeration, ventilation, and cooling eia (2022a). This presents an attractive target to optimize for minimal environmental and economic cost. With the proliferation of smart building technologies and the internet of things (IoT), access to data pertaining to commercial infrastructure operation has never been easier. In this work data from buildings is used to optimize energy usage for cooling and ventilation.

Heating, Ventilation, and Air Conditioning (HVAC) systems are used to regulate temperature and humidity in large buildings. An HVAC, when cooling, relies on the refrigeration cycle to transport heat from the source (living spaces) to the sink (outside environment). The heat exchange takes place using a fluid refrigerant. It evaporates by absorbing heat from the source, and condenses by expelling it to the sink. Usually water is used as an intermediary transport medium to absorb the refrigerant's heat and expel it into the environment. Water warmed in the condenser flows through a cooling tower where it loses heat via evaporation. Energy is consumed by refrigerant compressor and water pumps in the chiller, and cooling tower fans.

Optimal control of HVACs is a complex problem. There are subsystems, each with multiple control variables which have trade-offs in terms of performance. For instance, speeding up water flow through the cooling tower will result in a smaller temperature decrease but will increase the volume of water in contact with the refrigerant. Similarly speeding up fans will increase air flow which will increase evaporative cooling, but at a marginally decreasing rate. This is further compounded by the unique dynamics of each machine depending on wear-and-tear and environmental factors. A static control policy can be a good heuristic but will be suboptimal over a population of HVAC systems.

Data-driven control offers a solution. Using empirical measurements, a model of the system can be developed. This bypasses the need of complex physical representation of internal dynamics which are ultimately impertinent to the controller. Such a model can be treated as a black box and optimized over the space of control inputs. By capturing common dynamics across applications and reusing learned parameters, a data-driven controller can transfer to another application by fine-tuning on new data.

In this work, the application of a data-driven controller to a cooling tower in a HVAC system is documented. The challenges related to data collection and processing are discussed. Finally the resulting controllers are benchmarked against industry standards.

4.5.1 Approach

In this section, the overall problem and approach to a solution are described.

4.5.1.1 HVAC System Description

In this work, two mechanical draft cooling towers are analysed. A cooling tower is a terminal component of HVACs. Cooling towers expel heat from a chiller into the environment. A chiller is the central heat-exchange mechanism of a HVAC system. It uses either a vapor compression or an absorption-refrigeration cycle to extract heat into the refrigerant and generate chilled water which is supplied to a building. The hot refrigerant gas then condenses and expels its heat into another water loop. That loop passes through a cooling tower where the refrigerant's heat is dissipated into the environment.

A cooling tower primarily uses evaporation, and conduction and radiation secondarily, to get rid of excess heat from water. In a mechanical draft cooling tower, fans circulate air through a column while hot water falls under gravity. The contact between air and water leads to heat exchange. Fills can be added inside the tower column to increase contact surface area and time for more heat exchange. At the bottom of the tower cool water is collected and circulated back to extract excess heat from the chiller.

Evaporation depends mainly on three factors: temperature of water, surface area in contact with air, and partial pressure of water in air. Warmer water molecules have more kinetic energy and will escape into air faster. A larger surface area means a greater mass of water will evaporate in the same time period. Conversely, higher partial pressure of water in air (corresponding to high humidity) will reduce evaporation. Therefore dry air, or fast-blowing air such that humid air is displaced over water faster, will increase rate of evaporation.

The maximum amount of cooling possible depends on the wet-bulb temperature (T_{wb}) of ambient air. Wet-bulb temperature is the point at which air will become fully saturated with water vapor and will not be able to absorb more water. Evaporation will not be possible. Therefore the lowest temperature of water exiting from a cooling tower is bounded by the wet-bulb temperature.

Figure 4.20 illustrates the cooling tower and pertinent variables used in this work. Controllable variables in a cooling tower are the fan speeds for air flow, and condenser pump for water flow rate.

The cooling towers operate on a campus building, where each tower is attached to an 800-ton chiller. The towers operate one at a time. Each tower has two variable frequency drive fans which run in unison.

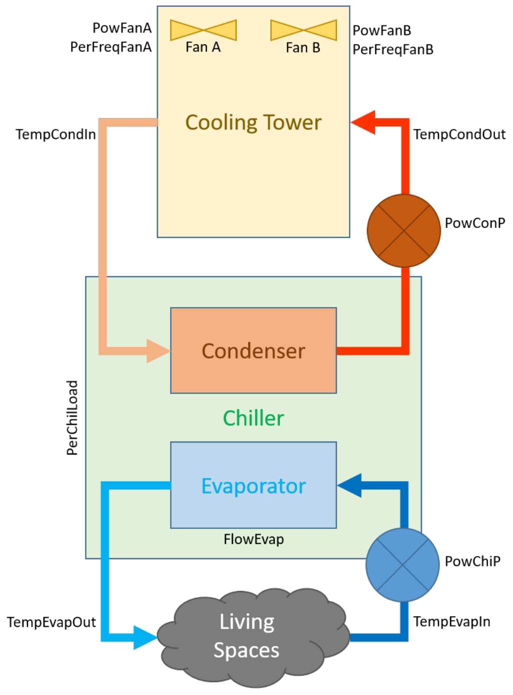


Figure 4.20: Schematic of an HVAC system showing relationships between components and measured variables.

4.5.1.2 Problem Description

The overarching goal is to train controllers that can adapt fast to a new environment, either as a result of a fault or a result of a new deployment. The control objective is to maximize temperature drop of water passing through the cooling tower ($T_{ct,i} - T_{ct,o}$), whilst keeping the fan power ($P_{ct,f}$) low. The hypothesis is that the cooler the water flowing into the chiller's evaporator unit, the more efficiently will heat be exchanged with the refrigerant. Given that the bulk of energy consumption of an HVAC is attributed to the chiller, a marginal drop in water temperature will have a multiplicative effect on net energy usage.

According to Newton's law of cooling, the rate of cooling is proportional to the instantaneous temperature differential with the surroundings. In this case the differential is relative to the differential with the wet-bulb temperature ($T_{ct,i} - T_{wb}$). If the marginal cooling with increase in fan speed is not positive, there is no utility in turning the cooling tower fan higher.

For this application, control is exercised through the temperature setpoint for water coming out of the cooling tower ($T_{ct,o}$). The internal logic of the HVAC uses the setpoint and an obfuscated PID controller to modulate fan speeds.

First, a data-driven model of the cooling tower is learned to predict exiting water temperature as a function of control variables. Then an optimal control policy is developed by exploring the control space. Finally the policy is evaluated in a data-driven environment of the cooling towers.

4.5.1.3 Data

Data for each cooling tower were collected from the HVAC system installed at the Engineering Science Buiding at Vanderbilt University. Measurements were taken at 5 minute intervals. Table 4.6 documents fields in the dataset.

4.5.1.4 Modeling Cooling Tower Temperature

From the theoretical discussion previously, and the physical models developed by Jin et al. (2007) and Cortinovis et al. (2009b), the exiting water temperature of the cooling tower $T_{w,o}$ is modeled as a function of incoming water temperature $T_{w,i}$, ambient temperature T_a, wet-bulb temperature T_{wb}, air flow rate S_a, and water flow rate $S_{w,ct}$. In this case, correlated variables are used to reflect the availability of data:

$$T_{ct,o} = f(T_{ct,i}, T_a, T_{wb}, F_{ct,a}, F_{ct,w}) \tag{4.25}$$

$$T_{ct,o} = f(T_{ct,i}, T_a, T_{wb}, D_{ct,p}) \tag{4.26}$$

Table 4.6: Original and derived fields in dataset and corresponding nomenclature.

Column Name	Variable	Description
	$F_{ct,w}$	Cooling tower water flow rate
PressDiffCond	$D_{ct,p}$	Differential pressure in condenser loop pump
	$F_{ct,a}$	Air flow rate through cooling tower
PerFreqFanA		Fan frequency as a percentage of maximum
PerFreqFanB		Fan frequency as a percentage of maximum
	F	Average fan frequency as a percentage of maximum
PerHumidity		Relative humidity
PowChi	P_{ch}	Power consumed by chiller
PowChiP	$P_{ch,p}$	Power consumed by chiller water pump
PowConP	$P_{ct,p}$	Power consumed by cooling tower water pump
Tonnage	L	System Load. Rate of heat extraction from building
PowFanA	$P_{ct,fa}$	Power consumed by fan A
PowFanB	$P_{ct,fb}$	Power consumed by fan B
PowFan	$P_{ct,f}$	Average power consumed by fans
TempAmbient	T_a	Ambient air temperature
TempCondIn	$T_{ct,i}$	Temperature of water out of the cooling tower
TempCondOut	$T_{ct,o}$	Temperature of water into the cooling tower
TempEvapIn	$T_{ch,i}$	Temperature of incoming chilled water
TempEvapOut	$T_{ch,o}$	Temperature of cooled chilled water
TempWetBulb	T_{wb}	Wet-bulb temperature
Setpoint	S	Setpoint for $T_{ct,o}$

A multi-layer perceptron (MLP) is chosen to model this function. A MLP, also known as a feed-forward neural network, is a time-invariant mapping from input features to output targets (unlike recurrent neural networks, which have temporal dependencies). A MLP can act as a universal function approximator over a compact real space Hornik (1991).

A physical model of energy rejection dQ/dt by a cooling tower, developed by Jin et al. (2007), can be written as:

$$\frac{dQ}{dt} = \frac{c_1 {F_{ct,w}}^{c_3}}{1 + c_2 \left(\frac{F_{ct,w}}{F_{ct,a}}\right)^{c_3}} (T_{ct,i} - T_{wb}) \tag{4.27}$$

Where $dQ \propto (T_{ct,o} - T_{ct,i})$, and (c_1, c_2, c_3) are learnable constants. Assuming slowly changing flow rates and ambient conditions, the solution is an exponential function of $T_{ct,i} - T_{wb}$. This can be modeled by a MLP. The model in equation 4.26 substitutes flow rates with differential pressure, and implicitly models the fan speed control logic from ambient conditions.

4.5.1.5 Reinforcement learning environment

The RL environment's dynamics are derived from the previously described model. The state vector of the environment has three categories of variables. First, independent ambient variables (T_a, T_{wb}) change regardless of control actions and describe the extraneous phenomena. Secondly, independent system variables ($D_{ct,p}, L$) change at the behest of other controllers. Finally, the dependent system variable ($T_{ct,i}$) is a result of the previous state and control action.

In consideration of the optimization objective, the model in equation 4.26 is augmented as in equation 4.28. The tonnage variable L reflects the overall load of the chiller system and the amount of heat extracted from the building. The additional outputs $P_{ct,f}, T_{ct,i}$ are used to predict the power consumption to optimize, and the water temperature into the cooling tower for the next time interval, after exchanging heat with the chiller.

$$T_{ct,o}, P_{ct,f}, T_{ct,i} = f(T_{ct,i}, T_a, T_{wb}, D_{ct,p}, L) \tag{4.28}$$

Each episode of the environment constitutes a 24 hour period divided into 5 minute intervals for a total of 288 time steps. A ticker tape of independent variables is fed to the state vector at each time step. The model is used to predict the dependent variable, and the inputs to the reward function. The model and the ticker together make up the state transition function ($x_{t+1} \leftarrow Tr(s_t, u_t)$).

Due to the stostically trained model, the outputs may not always fulfil physical constraints. In which case the environment clips outputs of the neural network model to adhere to physical laws, described in equation 4.29.

$$\begin{aligned} T_{ct,o} &= \max\left(\min(T_{ct,o}, T_{ct,i}), T_{wb}\right) \\ T_{ct,i} &= \max(T_{ct,o}, T_{ct,i}). \end{aligned} \tag{4.29}$$

The reward function optimizes for a high cooling tower efficiency $0 \leq E_{ct} \leq 1$ and a low fan power consumption, $0 \leq p_{ct_f} \leq 1$ which is the nominal power consumption $P_{ct,f}$ scaled to $[0,1]$. Equation 4.30 describes the feedback the controller receives for each action.

Table 4.7: Logic of the simple feedback "Up-Down" controller. The first two columns are the recorded changes in feedback and action. The last column is the next direction of change in action.

$(\Delta T_{ct,o})_t$	$(\Delta S)_t$	$(\Delta S)_{t+1}$
0	0	random
0	+	random
0	-	random
+	0	random
+	+	+
+	-	-
-	0	0
-	+	-
-	-	+

$$
\begin{aligned}
x &= [T_{wb}, T_a, T_{ct,i}, L, D_{ct,p}] \\
u_t &= [S] \\
x_{t+1} &= Tr(x_t, u_t) \\
E_{ct} &= \frac{T_{ct,i} - T_{ct,o}}{T_{ct,i} - T_{wb}} \\
R(x_t, u_t, x_{t+1}) &= E_{ct} - P_{ct,f}
\end{aligned}
\tag{4.30}
$$

4.5.1.6 Training Data-driven model

The extant setpoint logic for the cooling towers follows a fixed approach controller scheme, wherein $S \leftarrow T_{wb} + a$. Where a is a margin acknowledging the inefficiency of the cooling process. An approach too small will cause the fans to spin needlessly towards an unachievable cooling performance. An approach too large will leave room for improvement. To explore this, building administration instituted periods where setpoint was fixed or varied very little. The highly bi-modal nature of data can be seen in figure 4.21. The setpoint values do not capture the full breadth of system operation. Therefore the environment model's interpolation for missing values will be inaccurate.

To ameliorate data sparsity, a simple feedback controller, henceforth known as "Up-Down" controller was deployed. The controller is parametrized by the step size of the setpoint change ΔS, and the choice of feedback function which in this case was $T_{ct,o}$. Table 4.7 tabulates the logic of the controller. A positive feedback direction causes setpoint direction to maintain. A negataive feedback direction causes setpoint direction to reverse. Figure 4.21 shows the distribution of setpoints before and after deployment of the feedback controller.

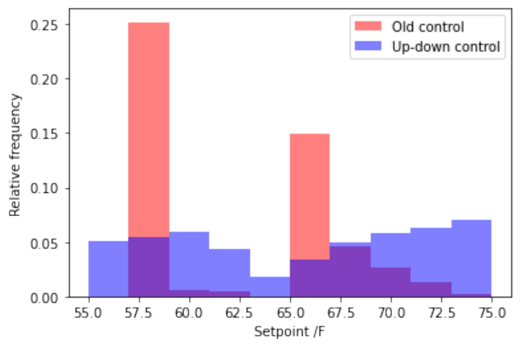


Figure 4.21: The setpoint distribution under the extant controller is highly bi-modal. An intermediate feedback controller was deployed to capture system dynamics.

4.5.2 Experiments

This section documents experiments carried out on environments learned using the data-driven models. The experiments evaluated the utility of transfer learning in different scenarios. For each transfer experiment, a RL controller trained in one environment was later trained on another. Secondly, a controller was first trained on a model of the second environment learned from 10% of the data and for 10% of the training steps, and then trained on the second environment. This was to evaluate the utility of preconditioning the controller for the new environment.

- Across equipment,
- Across ambient conditions,
- Across sparsity levels in data.

Each experiment was evaluated by bench-marking reinforcement learning performance during operation. The trained controllers were run over ten days' worth of episodes and the rewards were aggregated. The controllers for comparison used:

1. RL trained from scratch on the new environment,
2. "Up-Down" logic,
3. Fixed approach ($a = 5$),
4. Model predictive control with a 1-step horizon.

Controllers were first trained on data collected using the "Up-Down" controller for each cooling tower. Figure 4.22 shows the control behavior under identical independent state variables over a single day. Both controllers achieve high rewards per interval. However the actions taken are different. This demonstrates a knowledge gap across environments that transfer learning can solve. Figure 4.23 is the aggregate operational performance of various controllers on the transfer target: tower 2. The highest total rewards are from RL controllers trained natively on tower 2 and transferred from tower 1 to 2.

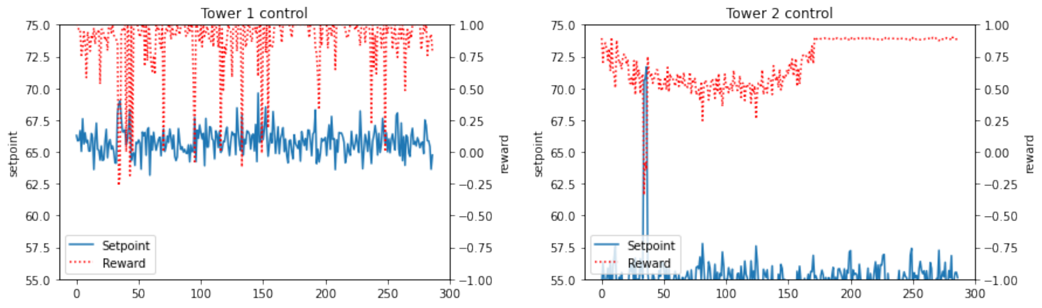


Figure 4.22: Setpoints for each cooling tower over the course of 24 hours.

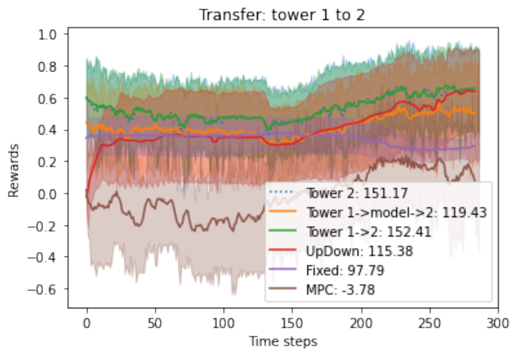
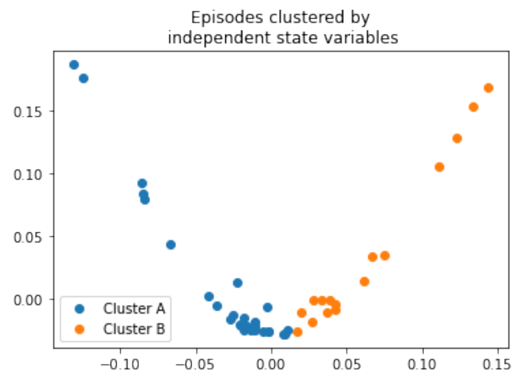


Figure 4.23: Performance of controller trained on cooling tower 1, later trained on cooling tower 2.

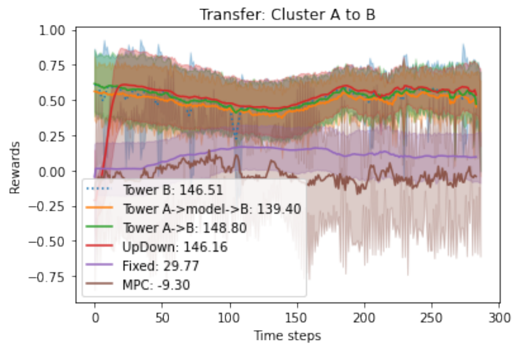
For the second set of experiments, controllers were trained on data from different operating conditions of the same cooling tower (tower 2). Figure 4.24a shows how data were put into two clusters for each controller to train on. The clusters were generated by calculating similarity measures between T_{wb}, T_a, L independent variables for each pair of days. Dynamic time warping was used to measure similarities. Then spectral clustering was used to divide episodes into two groups, A and B. The objective of the experiment was to transfer controller learned from cluster A to cluster B.

Figure 4.24b illustrates control performance over multiple episodes. The highest performing are RL controllers and the “Up-Down” controller.

The choice to use diverse setpoint data by deploying the Up-Down controller was validated by observing the quality of transfer of the data-driven environment model, and the eventual RL transfer performance. Figure 4.25 shows that the transfer from sparse to diverse setpoint data sets has the largest transfer gap left. For model transfer across towers in figure 4.25a, the transfer gap is large but is overcome due to the richness



(a) Dividing episodes for training by clustering independent state variables.

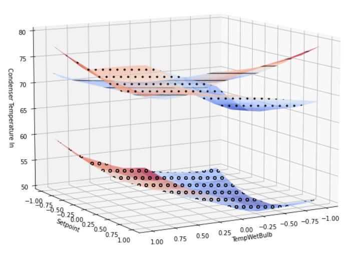


(b) Performance when transferring across state variable clusters.

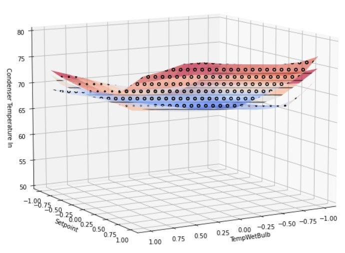
Figure 4.24: Transfer across clusters of independent state variables.

in the training data. For the transfer problem on the same tower but under different state variable distributions as shown in figure 4.25b, the transfer gap is small and easily overcome.

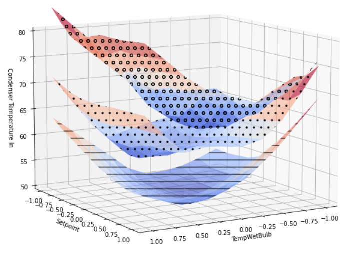
The effects of transfer gap in environment modeling manifest in the RL performance as well (figure 4.26), where the total reward difference between natively trained and transferred controllers on the target task is the highest for the case sparse to diverse data transfer.



(a) Tower 1 to 2



(b) Cluster A to B



(c) Sparse to diverse

Figure 4.25: Model transfer of $T_{ct,o}$ for different experiments. The surfaces are the predictions as the S and T_{wb} axes are varied. All other state variables are identical. The 'o' markers are the transfer source. The '/' markers are transfer target. The '.' markers are the transfer result.

Of note in all experiments is the poor performance of MPC control and fixed-approach controller. The former is explained by the data-driven model not being accurate and respecting physical constraints between temperatures as discussed earlier. Therefore the MPC controller's internal environment model my predict inaccurate states and feedback valuations which lead to suboptimal action choices. For fixed approach controllers, the fixed approach can be too ambitious, causing a power penalty, or be too lax, causing an efficiency penalty.

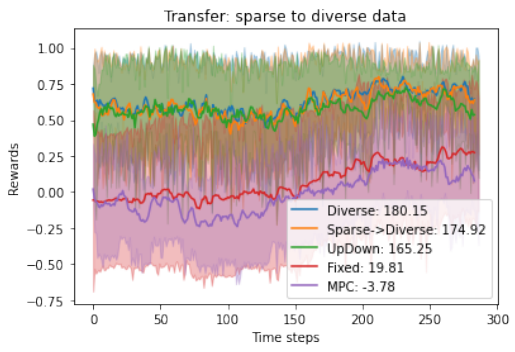


Figure 4.26

4.5.3 Conclusion

This work presented an applied approach to developing data-driven controllers for a class of HVAC systems with operational differences due to degradation and incipient faults. Challenges with data processing and modeling were presented, especially the need for representative data for modeling a data-driven controller. Finally the utility of using RL and transfer learning was demonstrated in relation to industry standard approaches like fixed-approach and model-predictive controllers. The transfer gap, in terms of model predictions and RL controller rewards, between source and target tasks was smaller when sufficient data was available for capturing environment dynamics during training. Future venues for research include codifying what pairs of tasks are considered near or far for transfer and how to adjust learning strategies to ameliorate any handicaps that it may entail.

4.6 Summary

In the preceding work, the utility of RL as an adaptive controller was demonstrated in different contexts. Two test-beds were used: a fuel-tank system, and a cooling tower in a HVAC sytem. Different methods for mitigating challenges of RL were demonstrated. First, a hierarchical RL algorithm was developed (section 4.1) that, motivated by the problem domain, discretizes the state and action spaces to be more tractable. Then, data was synthesized from data-driven models such that the machine learning process could consume real and virtual learning examples based on a scheduling algorithm (section 4.2). Following that, policy reuse and distillation approaches were examined wherein the policy was initialized such that data needed to adapt to the desired performance was reduced (sections 4.3, 4.4). Finally the case for transfer learning based on data-driven process models of a HVAC system was demonstrated in section 4.5.

In the aforementioned work, the thesis was to (a) either generate more data for the MDP such that the machine learning process can explore the problem and learn, or (b) initialize the control policy parameters close enough to an optimum such that the data requirements were reduced. However, little attention was paid to *how* the MDP changed when informing the adaptation of control. In chapter 5, adaptive control is viewed through the lens of policy transformation informed by knowledge of change in process dynamics.

Algorithm 2 Variable n-Step Tree Backup Episode

```
Require: initial state $x_0$
Require: system model $Tr : X \times U \rightarrow X$
Require: value function $Q : X \times U \rightarrow \mathbb{R}$
Require: reward function $R : X \times X \rightarrow R$
Require: hierarchy $H : X \rightarrow \Re$
Require: action selection policy $\pi : X \rightarrow \Pi(U)$
Require: parameters: $\alpha, d, \gamma, n$
 1: Get action $u_0$ from $\pi_{exp}(x_0)$
 2: $Q_0 \leftarrow Q(x_0, u_0)$
 3: $T \leftarrow \infty, t \leftarrow 0, \tau \leftarrow 0$
 4: while $i < d$ and $\tau < T - 1$ do
 5:     if $i < T$ then:
 6:         Take action $u_t$ for $H(x_t)$ steps
 7:         $x_{i+1} \leftarrow T(x_i, u_i)$
 8:         $R_i \leftarrow R(x_i, u_i, x_{i+1})$
 9:         if $x_{i+1}$ is in goal states then
10:             $T \leftarrow t + 1$
11:             $\delta_i \leftarrow R_i - Q_i$
12:         else:
13:             $\delta_i \leftarrow R_t + \gamma \sum_u \pi_{exp}(u \mid x_{i+1}) Q(x_{i+1}, u) - Q_i$
14:             $u_{i+1} \leftarrow \pi_{exp}(x_{i+1})$
15:             $Q_{i+1} \leftarrow Q(x_{i+1}, u_{i+1})$
16:             $\pi_{i+1} \leftarrow \pi_{exp}(u_{i+1} \mid x_{i+1})$
17:         end if
18:     end if
19:     $\tau \leftarrow i - n + 1$
20:     if $\tau \geq 0$ then
21:         $E \leftarrow 1$
22:         $G \leftarrow Q_i$
23:         for $k = \tau, ..., \min(\tau + n - 1, T - 1)$ do
24:             $G \leftarrow G + E \cdot \delta_k$
25:             $E \leftarrow \gamma E \cdot \pi_{k+1}$
26:         end for
27:         $Error \leftarrow Q(s_\tau, a_\tau) - G$
28:         Update $Q(s_\tau, a_\tau)$ with $\alpha \cdot Error$
29:     end if
30:     $i \leftarrow i + 1$
31: end while
```

Algorithm 3 Offline RL Control

```
1: $d \leftarrow \infty$
2: for $x \in X$ do
3:     Call Episode
4: end for
5: while True do
6:     $x_0 \leftarrow CurrentState$
7:     $action = arg\max_{u \in U} Q(x_0, u)$
8: end while
```

Algorithm 4 Online RL Control

```
Require: exploration rate $\varepsilon$
Require: sampling density $\rho$
 1: while True do
 2:     $x_0 \leftarrow CurrentState$
 3:     if $RandomNumber < \varepsilon$ then
 4:         $neighbours \leftarrow \{T(x_0, u) : u \in U\}$
 5:         for $state \in neighbours$ do
 6:             if $RandomNumber < \rho$ then
 7:                 Call Episode
 8:             end if
 9:         end for
10:     end if
11:     $action = arg\max_{u \in A} Q(x_0, u)$
12: end while
```

Algorithm 5 Proposed reinforcement learning-based control loop.

```
Require: $t_{online}$                                   ▷ Duration of online control
Require: $t_{offline}$                                  ▷ Duration of offline learning
Require: tanks, engines degradation factors
Require: offline $\in$ True, False
 1: initialize policy function $\pi$, system model
 2: loop
 3:     degrade system
 4:     for $t = 0$ to $t_{online}$ do
 5:         Apply control action $u = \pi(x)$ from state
 6:         Cache experience (states, actions, rewards)
 7:         if  mod $t, t_{update} = 0$ then
 8:             Update policy $\pi$ given system
 9:         end if
10:     end for
11:     if offline=True then
12:         Update system model from cache
13:         Clear cache
14:         for $t = 0$ to $t_{offline}$ do
15:             Apply control action $u = \pi(x)$
16:             if  mod $t, t_{update} = 0$ then
17:                 update policy $\pi$ given system model
18:             end if
19:         end for
20:     end if
21: end loop
```

Algorithm 6 Model-agnostic meta-learning

```
Require: parameters $\theta_k$, MDPs $P$, learning rates $\alpha_{in}, \alpha_{out}$, iterations $K_{in}, K_{out}$
  Set $\theta' \leftarrow \theta_k$
  for $k_{out} = 1$ to $K_{out}$ do
      Sample MDPs $p^i \sim P$
      for all $p^i$ do
          Set $\theta^i \leftarrow \theta'$
          for $k_{in} = 1$ to $K_{in}$ do
              Sample training trajectories $\mathscr{M}^i$ from $p^i$
              Calculate gain function from $\mathscr{M}^i$
              Update $\theta^i \leftarrow \theta^i + \alpha_{in} \cdot \nabla_{\theta^i} G$
          end for
          Sample test trajectories from $p^i$
          Calculate test gain $G^i$ on sample;
      end for
      Update $\theta' \leftarrow \theta' + \alpha_{out} \cdot \Sigma_i \nabla_{\theta'} G^i$
  end for
  Return $\theta'$
```

Algorithm 7 Complementary meta-RL

```
Require: parameters $\theta_k$, memory $\mathscr{M}$, learning rates $\alpha_{in}, \alpha_{out}$, iterations $K_{in}, K_{out}$
Require: (optional) policy complement $\mathscr{C} = \{\varnothing\}$, process model $p_m = \varnothing$
  if $p_m \neq \varnothing$ then
      Update $p_m$ from $\mathscr{M}$
      Sample trajectories from $p_m$, using policy
  else
      Sample trajectories from $\mathscr{M}$, discarding policy
  end if
  Set meta-updated params $\theta' \leftarrow \theta_k$
  Set baseline params $\theta^b \leftarrow \theta_k$
  for $k_{out} = 1$ to $K_{out}$ do
      Calculate gain from $\mathscr{M}$
      Update $\theta^b$ using $\alpha_{out}$
      for all $\theta^i$ in $\mathscr{C}$ do
          Sample trajectories and calculate gain
          Update $\theta^i$ using $\alpha_{in}$
          Calculate test gain $G^i$
      end for
      Update $\theta' \leftarrow \theta' + \alpha_{out} \cdot \Sigma_i \nabla_{\theta'} G^i$
  end for
  Calculate $J_{\pi_{\theta'}}, J_{\pi_{\theta^b}}$ from $p_m$
  if $J_{\pi_{\theta'}} < J_{\pi_{\theta^b}}$ then
      Return $\theta' \leftarrow \theta^b$
  else
      Return$\theta'$
  end if
```

Algorithm 8 Populating complement of policies

```
Require: policy complement $\mathscr{C}$, complement size $s$, memory $\mathscr{M}$
  Initialize divergence matrix $D = [0]^{|C| \times |C|}$
  for $\theta_1, \theta_2$ in Permute($\mathscr{C}$) do
      Action probabilities $p_1, p_2 = \pi_{\theta_1}(\mathscr{M}), \pi_{\theta_2}(\mathscr{M})$
      KL-Divergence $d = \Sigma p_1 \cdot \log(p2/p1)$
      $D[\theta_1, \theta_2] = d$
      Sum each row of $D$ for total divergence $D_T^{|C| \times 1}$
      Most divergent parameters $\mathscr{C} \leftarrow$ Sort($\mathscr{C}$) by $D_T$
      Return First $s$ parameters from $\mathscr{C}$
  end for
```

CHAPTER 5

Model+data-based adaptation via policy transformation

The generality of machine learning is its strength and its challenge. ML approaches are concerned with optimizing a reward/cost function evaluated for a function's parameters to be learned over some data. The previous section discussed complementing the data generation and parameter initialization processes. While the approaches were evaluated over test-beds that experienced disturbances and faults, the nature of those changes did not figure into the adaptive algorithm themselves.

This chapter addresses exactly that. Consider, for example, the fuel-tank system demonstrated before. The flow rate through valves is related to the pressure in the vessel. If a fault occurs, such as pressure loss, the controller has to extend valve open states for a correspondingly longer duration to transfer the same fuel mass. An identified fault, along with a model of process dynamics, can be used to reason about control adaptation and fault tolerance.

This is what happens in Model-Predictive Control. The caveat being that MPC solves the optimization problem recurrently, incurring additional computational load. RL instead *memorizes* optimal actions by solving the Bellman equation and encoding results in the value and policy functions. Can the knowledge of faults and process dynamics be used to adapt a RL control policy? The following sections demonstrate exactly that.

5.1 Adaptation for sample efficient transfer in reinforcement learning control of parameter-varying systems.

Transfer learning in control seeks to reuse the knowledge gained from previously executed tasks to speed up or improve performance on related, target tasks. This is an especially attractive proposition in applications where the relearning process is time constrained and the training samples are sparse. For example, consider data-driven fault-tolerant control, as surveyed by Li et al. (2022b), where a controller needs to adapt quickly and sufficiently well to a changed environment.

However, the questions underlying transfer are: *what knowledge from past experiences do we transfer, and how do we transfer it well?* Poorly related tasks may cause negative transfer when conventional transfer learning methods are applied, especially if the relationship between tasks is not considered in the transfer process Wang et al. (2019b); Cao et al. (2010). Fundamentally, the performance of transfer learning between tasks is dependent on the relationships between tasks. Tasks that are similar will have similar control policies, therefore, they will need lesser time and data for their policies to adapt to a changed environment.

Thus far, a substantial body of work has addressed transfer in the data and algorithm space, which we describe in section 5.1.2. Task similarity measures have been developed from the statistical properties of measurements across tasks. Stochastic algorithms have been proposed (sections 5.1.2.1, 5.1.2.2) that leverage model architectures, machine learning hyperparameters, and control policy parameter update rules to achieve faster, jump-start, or asymptotically higher performance improvement on the target task, all the while being agnostic to the underlying process dynamics.

This work makes contributions in a related direction. We address transfer in the space of process dynamics. While our approach is applicable to a broad class of dynamical systems, we demonstrate strong theoretical results for systems with linear, time-invariant dynamics. By modeling the relationships between source and target task dynamics as linear transformations, we develop a transformation of the source policy to transfer to the target task. We demonstrate conditions where the transformation will produce behavior optimal in the least squares sense. Generally, for approximately identified target tasks or locally optimal source policies, the transformation may be used as a policy initialization to get a jump-start improvement, prior to further optimization.

The following section describes the transfer learning problem for this work in the context of reinforcement learning. Following that, in section 5.1.2, we review different approaches to transfer. Section 5.1.3 motivates cases for invariance of optimal policies and cases for the derivation of policy transforms. Finally, experiments are done on dynamical systems using stochastic and classical control approaches to demonstrate these concepts.

5.1.1 The transfer learning control problem

In this work, we discuss the transfer of control across systems modeled as Markov Decision Processes (MDPs). Control of an MDP is a sequential decision-making problem. The control task $T \in \mathscr{T}$, is characterized by the process dynamics $P : X \in mathbbR^n \times U \in mathbbR^m \rightarrow X$, which map the current state x_i to the next state x_{i+1}, given an input u_i. Each state transition is assigned a reward based on the state and the action taken to reach there $r : X \times U \rightarrow \mathbb{R}$. An episode is a sequence of interactions until a terminal or goal state is reached. The objective for T is to derive a policy $\pi_T : X \rightarrow U$, such that each action picked maximizes expected future returns $\mathbb{E}[G(x_i)]$. The return $G(x_i) = \sum_{j=i}^{\infty} \gamma^{j-i} r(x_i, \pi_T(x_i))$ is the discounted sum of rewards starting at x_i following some policy to a terminal state. The discount factor $\gamma \in (0, 1]$ prioritizes the immediacy of feedback. Expected returns under an optimal policy are known as its value, $V(x) = \max_\pi \mathbb{E}[G(x)]$. Then, $\pi_T(x) \leftarrow \arg\max_u V(x)$. That is, under an optimal policy, the sequence of states traversed until the terminal state is optimal.

The transfer problem is summarized as follows. A task T consists of the process dynamics P and the reward function r. Given a target task T_t and a population of source tasks $\mathscr{T}_s \in \mathscr{T}$, find a source task $T_s \in \mathscr{T}_s$

and a transfer mechanism, such that the performance of its policy fine-tuned on T_t. This provides an optimal solution for T_t. In other words,

$$T_s : \max_{T \in \mathscr{T}_s} G(x \sim T_t \mid \pi_{T \to T_t}) \tag{5.1}$$

T_s and T_t may differ on process dynamics and reward, e.g., due to a fault or a changed control objective. Both affect evaluated returns G. The objective in equation 5.1 can only be achieved retroactively, when candidate source tasks have been evaluated on the target task. However, exploring the effectiveness of transfer of each source task may violate time and safety constraints in specific applications. Therefore, our challenge is to *preemptively* select a favorable source task and transfer mechanism to address the time to convergence and the sample efficiency problem.

Practically, this applies to cases where a single MDP's state transitions are changed due to faults, degradation, and disturbances in the environment. While the objective remains still to tolerate the change and manipulate the MDP as in the source case. For the remainder of this work, subscripts $*_s, *_t$ refer to source and target parameters, respectively. The goal is to achieve high returns on the target task. This may be done by either picking a source task liable to transfer well or by tweaking the transfer process, such that the source policy converges swiftly to an "optimum" on the target task. Prior related work has addressed both of these approaches.

5.1.2 Related Work

The aforementioned goal of transfer learning for control overlaps with multi-task learning Zhang and Yang (2021), reward shaping Brys et al. (2015b), and few-shot learning Wang et al. (2020b). The body of research is divided into two broad categories: (1) optimizing transfer learning informed by task similarity, and (2) optimizing transfer once a source task is picked.

5.1.2.1 Selecting Similar Tasks for Transfer

A number of similarity measures have been used for transfer learning task similarity for classification and regression problems. Work by Zhou et al. (2021) builds on Model-Agnostic Meta-Learning (MAML) Finn et al. (2017a) to develop task similarity-aware MAML. They represent task similarity as Euclidean distance between parameters of models trained on sampled tasks. Clusters are made for similar tasks, and the cluster closest to the target task is used during meta-initialization. Fernandes and Cardoso (2019) propose a general transfer approach, where task similarity is used to regularize the transferred model. Wang et al. (2019a) formalize the intuition that similar tasks have similar performances, and use the performance gap as a regularizer for transfer learning. Euclidean distance between model coefficients is used again. Alternatively, Zhang et al. (2017) hypothesize that, given samples, two (classification) tasks are similar if the accuracy on the source task and the target tasks is high. Lu et al. (2017) propose a reconstruction classifier, which attempts to reconstruct samples from the target task using samples from the source task. The assumption is that samples from $T_t \cup \mathscr{T}_s$ lie on a subspace that can be modeled as combinations of each other. The sparser the coefficients for reconstruction, and the lower the error, the higher the similarity. On the other hand, Shui et al. (2019) looks at the utility of explicitly including task similarity measures in the learning process. They use $\mathscr{H}$ divergence and Wasserstein distance between probability distributions in adversarial multi-task learning for classification tasks. Recently, Alam et al. (2022) develop physics-guided models to develop an initial reinforcement-learned control policy under inaccurate rewards, which in turn is transferred to the target process. The initialization of the source policy reduces the sample size needed to adapt to the target task.

5.1.2.2 Optimization of the Learning Process

Another class of approaches has addressed the learning process itself, once a source task is selected. Such methods are not only restricted to transfer learning; they apply to learning algorithms in general. Relevant work in this area touches on meta-learning Hospedales et al. (2021), neural architecture design Du et al. (2019); Luo et al. (2019), and hyperparameter tuning Yogatama and Mann (2014); Paul et al. (2019).

In recent work in classical control, Chakraborty et al. (2020) propose the design of a transferable controller via system identification. They stimulate the target process with tailored exploratory actions from the same initial state to identify relationships with the source process. The source policy can then be transformed to the identified target process.

5.1.3 Policy Transfer via Transformation

In this section, we derive an adaptation mechanism for transferring an optimal source policy π_s to a target process. The adaptation is a policy transformation appended to π_s after the relationship between P_s and P_t is identified. We demonstrate this for a category of tasks where the source policy is as close to optimal on the target task in the least squares sense.

Assumption 1. *For this work, we assume adaaptation via homogeneous transfer between $T_s = (P_s, r), T_t = (P_t, r)$. The state and action spaces of source and target processes are the same. The reward function between tasks does not change. The process dynamics are related by a time-invariant parametric change which affects state transitions.*

Assumption 2. *Reward is a time-invariant function of state only.*

Remark. This is a strong assumption, but one which is realizable in some cases. For instance, where the cost of actions is dwarfed by the incentive to drive the state variables to a certain point. In the real world, applying actions costs energy, which is a common optimization objective. In such cases, as a proxy for the energy cost of actions, the energy reserves of the process (fuel) may be appended to the state vector.

Based on assumption 2, we propose theorem 1.

Theorem 1. *Let P_s, P_t be two processes with identical state and action spaces, and reward functions. Let π_s, π_t be the optimal policies on P_s, P_t respectively. Then $s_s(x_i) = \{x_i, x_{i+1}, x_\infty : x_{i+1} \leftarrow P_s(x_i, \pi_s(x_i))\}$ is the optimal sequence of states obtained from x_i on P_s by following an optimal policy. Then, the optimal sequence of states on P_t by following the optimal policy for P_t, will be the same. $s_s(x_i) = s_t(x_i) = \{x_i, x_{i+1}, x_\infty : x_{i+1} \leftarrow P_t(x_i, \pi_t(x_i))\}$*

The optimal policy π - the control law - is derived to reach the optimal set of states. It does so by picking actions traversing states with the highest value. If assumption 2 holds, in equation 5.2, the summation of encountered rewards is over a function of state only. Meaning, the value of a state depends solely on the states encountered whilst following an optimal policy, and not the actions taken. Therefore, the $\arg\max_\pi$ operation searching over policy space will find whatever actions are needed to bring about the same encountered states. This implies that the valuation of a state does not change if the process dynamics change (equation 5.3).

$$\begin{aligned} & r : X \to \mathbb{R} \implies \nabla_u r = 0 \\ & \pi : \arg\max_\pi \mathbb{E}_{x_i \sim X} \sum_{j=i}^{\infty} \gamma^{j-i} r\left(P(x_i, \pi(x_i))\right) && (5.2) \\ & \nabla_P V(x) = 0 && (5.3) \end{aligned}$$

If the process dynamics change, $P_s \to P_t$, the response to the previously optimal policy changes as well. No longer may old actions lead to states with the highest value. However, the same set of states optimal under P_s are optimal under P_t. Therefore if π_t can bring about the same state change in P_t as π_s does in P_s, π_t can be guaranteed to be optimal. This comes with one caveat: that P_t is controllable for the same set of states (Sussmann and Jurdjevic, 1972).

Assumption 3. *For each state $x_i \in s_s(x_i)$, there exists an action $u_t \leftarrow \pi_t(x_i)$ such that it will drive P_t to the same state transition as $P_s(x, \pi_s(x))$.*

For illustration, consider a path following task for a vehicle. The state comprises the vehicle's position. The reward penalizes distance from the target only. It does not consider the energy spent by steering actions.

Trivially, the optimum of the value function resides at the reference position for the vehicle. Consider this process encounters a disturbance, for example slippery surface or steering degradation. The dynamics have changed. However, under the reward function, the optimum of the value surface is still the reference position, so long as an adapted action can drive the vehicle to reach that state. However, if the reward were a function of action too, such as energy consumed for large steering inputs, then the optimum state would be different. Since, another position may exist where the energy savings are larger than the penalty of missing the reference position.

For the remainder of this section, an algorithm is developed that relies on system identification to apply a transformation to an optimal source policy π_s, such that the resulting policy π_t will be optimal on a linear target process P_t.

Assumption 4. *The process dynamics are linear and time-invariant.*

Remark. A linear system is where $\dot{x} = \partial P / \partial t$ can be characterized by $\dot{x} = Ax + Bu$, where $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}$ are constant matrices. A is the response to internal state, and B is the response to external inputs. The measured output of the process $y \in \mathbb{R}^l$ is given as $y = Cx + Du$, where $C \in \mathbb{R}^{l \times n}, D \in \mathbb{R}^{l \times m}$. We assume that the process is observable, and therefore use the internal state x. For experiments preempting future work, nonlinear but piece-wise linear systems are also demonstrated with this approach.

The change in process is taken as a parametric fault, manifesting in a change of the A, B matrices. For example, a parametric fault can be a failure of an actuator leading to a 0 element in B, or a dissipitave effect in A, causing the state to decay faster. We assume $F_A : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}, F_B : \mathbb{R}^{n \times m} \to \mathbb{R}^{n \times m}$ represent transformations of A_s, B_s under such a fault. Then the nominal (source) and faulty (target) process dyanamics can be represented in equations 5.4, 5.5 respectively. The optimal control policy π_s already has optimized inputs to change states to optimal positions. Since the optimal states remain invariant, we want a policy π_t such that the change of state under the target process, $\dot{x}_t$ is the same as that under the source process, $\dot{x}_s$.

$$\dot{x}_s = A_s x + B_s u_s \tag{5.4}$$

$$\dot{x}_t = F_A A_s x + F_B B_s u_t \tag{5.5}$$

If $\dot{x}_t = \dot{x}_s$, then,

$$\implies u_t = (F_B B_s)^{-1}(I - F_A) A_s x + (F_B B_s)^{-1} B_s u_s \tag{5.6}$$

$$\implies u_t = \left((F_B B_s)^{-1}(I - F_A) A_s + (F_B B_s)^{-1} B_s \pi_s \right) x \tag{5.7}$$

Equation 5.6 represents a multiplicative transformation of the source policy by $(F_B B_s)^{-1} B_s$, representing

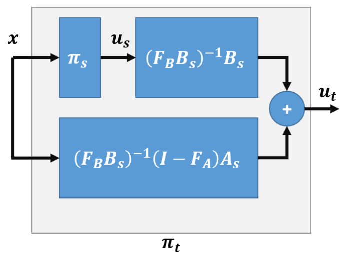


Figure 5.1: A schematic of the policy transformation. The nominal action u_s is transformed as u_t to bring about the same change in state in the target system that was considered optimal by π_s in the source system. The source policy can be any control algorithm for e.g. LQR, MPC, RL etc.

a change in the input's effect on state dynamics. And an additive correction by $(F_BB_s)^{-1}(I-F_A)A_s$, representing the changed internal dynamics of the system. Equation 5.7 factors out $u_s \leftarrow \pi_s x$ to get an equivalent representation for u_t. Figure 5.1 depicts how these transformations can be appended in series and parallel, respectively to an existing policy function π_s.

The transformation of state dynamics, F_A is not required to be invertible. The only functions needing inversion are the action dynamics B_s and their transformation F_B. In cases where a perfect inverse does not exist, the Moore-Penrose Pseudo-inverse may be used as the closest approximation in the least squares sense. The approximation error of the pseudo-inverse may be a measure of the suitability of the transformed policy.

The transformation will change the policy's response to each state. The image of the target policy function may differ from the source policy, $\pi_t[X] \neq \pi_s[X]$. If the target policy's image is a subset of the source's image, or if actions are unconstrained in the process, this will not be notable. For cases where actions have to be constrained to $\pi_s[X]$, they can be scaled down. For linear systems, this will not affect the eventual dynamics. That is, a sequence of actions scaled down by a factor of $k \in \mathbb{R}^+$ over k intervals will result in the same change in state as the unscaled action applied for one interval.

Assumption 3 can be verified for linear processes. Given A, B, a process is controllable to any state if the controllability matrix $R \in \mathbb{R}^{n \times nm}$ has full row rank.

$$\texttt{rank}(R) = \texttt{rank}\left([B, AB, ..., A^{n-1}B]\right) = n \tag{5.8}$$

A particular state x is reachable in a process if it is in the image of R. The domain of R is formed from the vector of stacked actions $\in \mathbb{R}^{nm}$. This may be used to verify that P_t is controllable to the optimal sequence of states $s_s(x)$ which is invariant from P_s.

A_s, B_s, F_A, F_B are learned through system identification strategies. For the case where process dynamics

Algorithm 9 Policy transformation via system identification.

```
Require: $\pi_s, P_s, P_t$
Require: Data buffer $\mathscr{D}_s, \mathscr{D}_t$
1: Collect $(x_i, u_i, x_{i+1})$ into $\mathscr{D}_s$ using $P_s, \pi_s$
2: Collect $(x_i, u_i, x_{i+1})$ into $\mathscr{D}_t$ using $P_t, \pi_s$
3: Use equations 5.12, 5.11 to learn $A_s, B_s, F_A, F_B$
4: Use equation 5.7 to get $\pi_t$
5: Return $\pi_t$
```

$\partial P(x,u)/\partial t$ and their transformations F_A, F_B can be linearized in the region of interest, they can be learned from measured data by solving a least squares problem:

$$\begin{aligned} x_{i+1} &= P(x_i, u_i) \\ &= x_i + \int_{t=i}^{t=i+1} \dot{x}_i \partial t \\ &\approx x_i + \delta t \cdot \dot{x}_i \end{aligned} \tag{5.9}$$

Where δt is a sampling interval. For linear systems, P can be approximated as a linear transformation,

$$\dot{x}_i = Ax_i + Bu_i$$

$$x_{i+1} \approx P \cdot [x_i, u_i]^T \tag{5.10}$$

$$= [1 + \delta t A, \delta t B][x_i, u_i]^T \tag{5.11}$$

Given $\mathscr{X}$, $\mathscr{X}^+$, $\mathscr{U}$ are measurements of state, action, and next state, the approximated process in equation 5.10 can be obtained via solving the system of equations:

$$\mathscr{X} = [x_0, x_1, ...], \mathscr{X}^+ = [x_1, x_2, ...], \mathscr{U} = [u_1, u_2, ...]$$

$$P = \left[\mathscr{X}^+ \cdot [\mathscr{X}; \mathscr{U}]^T\right] \cdot \left([\mathscr{X}; \mathscr{U}] \cdot [\mathscr{X}; \mathscr{U}]^T\right)^{-1} \tag{5.12}$$

Then, P can be decomposed and solved for A, B as between equations 5.10 and 5.11. If A_s, A_t are known, then $F_A = A_t \cdot A_s^{-1}$. Similarly, for F_B, B_s, B_t. Since B is not necessarily square, $F_B = (B_t \cdot B_s^T) \cdot (B_s \cdot B_s^T)^{-1}$. The whole approach is outlined in algorithm 9.

For non-linear transformations of the linear system matrices A, B, non-linear basis functions, such as neural networks can be used to approximate the target transformation. Since $F_B B_s$ requires inversion, a monotonic constraint should be put on their learned models (by constraining hidden layer weights to be positive, for example). Or, probabilistic models which learn a posterior distribution of a variable, given the output of a function Ardizzone et al. (2018) can be used, from which the likely inverse of the function can be sampled.

In summary, if the reward is a function only of the state, and if optimal states in P_s are reachable in P_t, then optimal change in the state remains invariant. Thus π_t can be derived as a transformation of π_s in terms of P_s and P_t to bring about the same change in state to optimize the target task. For example, this may be true for trajectory-following systems, where the trajectory the vehicle is required to follow is fixed. After faults and disturbances, an existing policy has to be transformed to accommodate the fault or disturbance, but the rewards associated with the optimal trajectory are invariant.

The worst-case computational cost of our approach is lower than that of approaches that evaluate states and actions *anew* on the target task, such as traditional RL and MPC approaches. Our approach relies on system identification to transform π_s, which is already known. As an illustration, we consider deterministic, continuous MDPs, assuming unique actions lead to unique states. Identifying an arbitrary system P requires sampling each state transition once to learn that mapping $x_{i+1} \leftarrow P(x_i, u_i)$. The complexity of identification then is the space $X \in \mathbb{R}^n \times U \in \mathbb{R}^m$. The Bellman equation (Bellman, 1966), which is foundational to RL and MPC approaches, traverses all possible state *trajectories* to evaluate states, where state transitions may be traversed multiple times, thus giving a higher computational complexity. Therefore, learning the target dynamics to transform the source policy is computationally cheaper than evaluating states in the target task. Even when a source RL policy is re-used and fine-tuned on the target task, it may need to sample every state trajectory in the worst case. The source policy has learned to drive toward valuable states (i.e., states that are close to the specified trajectory) under P_s. It needs to re-sample the transformed trajectories under P_t to re-evaluate each state. The number of trajectories needing revision will measure, and depend on, the similarity between P_s and P_t.

5.2 Experiments

We demonstrate our approach with examples from linear and non-linear systems. Our comparison studies, evaluate our approach against the Linear Quadratic Regulator (LQR) (Lavretsky and Wise, 2013), Model-Predictive Control (MPC), and RL using Proximal Policy Optimization (PPO) algorithm. The reward r and cost c, where $r = -c$, are specified as quadratic functions of state x with weights Q, where Q is a diagonal matrix. The function minimizes cost and maximizes reward around $x = \hat{0}$, which corresponds to the desired state vector. However, the system can be driven to some other point x_0 by substituting $x \leftarrow x - x_0$ without a loss of generality. For comparing against LQR, we introduce a small action weight $R = 10^{-5}$ in the reward, but that does not affect other approaches. Similarly, to accommodate LQR, the optimization assumes unconstrained actions. However, during testing, the actions are clipped to $[-1,1]$ for each time interval.

$$c = -r = x^T Qx + u^T Ru \tag{5.13}$$

A simple one-dimensional temperature (x) regulation system is first used as a test bed, where positive and negative actions control a heating or cooling element (u). System parameters are set as $a = -0.1, b = 1, Q = I$. Faults represent a change in conductivity a, and a reversal in action polarity b, such that the nominal action of increasing heat will now cool the system.

$$\begin{aligned} & x, u, a, b, F_A, F_B \in \mathbb{R} \\ & \dot{x}_{temp} = ax + bu \end{aligned} \tag{5.14}$$

A higher dimensional, but linear, spring-mass system is described in equation 5.15, with $Q = I$, and actions u as forces. The dynamics are governed by the mass $m = 1$, spring constant $k = 10$, and dynamic friction $k_f = 0.2$.

$$\begin{aligned} & x \in \mathbb{R}^2,\ u \in \mathbb{R},\ k, k_f, m \in \mathbb{R}^+, F_A, F_B \in \mathbb{R}^{2\times 2} \\ & \dot{x}_{spring} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & \frac{k_f}{m} \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \end{aligned} \tag{5.15}$$

A more complex non-linear, continuous system is a pendulum in equation 5.16, with $Q = I$, and actions u as torques. The dynamics are governed by the mass $m = 0.1$, pendulum length $l = 1$, gravitational acceleration $g = 10$, and dynamic friction $k_f = 0.02$.

$$\begin{aligned} & x \in \mathbb{R}^2,\ u \in \mathbb{R},\ g, l, k_f, m \in \mathbb{R}^+, F_A, F_B \in \mathbb{R}^{2\times 2} \\ & \dot{x}_{pendulum} = \begin{bmatrix} 0 & 1 \\ -\frac{g\sin(\cdot)}{l} & -\frac{k_f}{ml^2} \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u \end{aligned} \tag{5.16}$$

Finally, a cart pole system is used to demonstrate a more complex non-linear case study. The state vector x comprises the angle of the pole x_1, its angular velocity x_2, the position of the cart x_3, and the cart's velocityx_4.

Actions u are forces applied to the cart. The system is parameterized using the cart and pole masses, $m_c = 0.5, m_p = 0.1$, the length of the pole $l = 1$, gravitational acceleration $g = 10$, and the coefficient of friction $k_f = 0.01$. The state weights for reward are $Q = [[1,0,0,0],[0,0.1,0,0],[0,0,10^{-5},0],[0,0,0,0.1]], R = 10^{-5}$. For the reinforcement learning controller, at each time step when the pole is upright gains a constant reward of 1. The state equations are factored into terms that are functions of x and u, and $F_A, F_B \in \mathbb{R}^{4\times 4}$ are applied as disturbances.

$$\begin{aligned}
\dot{x_1} &= x_2\\
\dot{x_2} &= \frac{g}{l}\sin x_1 - \frac{k_f x_2}{ml^2} + \frac{\dot{x_4}\cos x_1}{l}\\
\dot{x_3} &= x_4\\
\dot{x_4} &= \frac{m_p \sin x_1 \left(g\cos x_1 - l\dot{x_1}^2\right) + u(t)}{m_c + m_p - m_p\cos^2 x_1}
\end{aligned} \tag{5.17}$$

Experiments were carried out by obtaining a source policy π_s on the nominal process P_s. Then, a fault, denoted by a parametric change in the state equations was introduced. The change in the system dynamics was modeled by estimating F_A, a positive definite matrix, and F_B, a negative definite matrix. A buffer of measurements $\mathscr{D}_t$ is collected to estimate the target process P_t. The transformed policy π_t is derived from π_s by approximating both P_s and P_t as linear systems about the buffer.

We evaluate *jumpstart improvement*, *asymptotic improvement*, and *time to threshold*. These metrics describe the short- and long-term advantages of our approach and its computational complexity. Jumpstart improvement is the immediate difference in rewards when a policy interacts with a new task. Asymptotic improvement is the limit of accumulated rewards as the policy continues to learn. And time to threshold is the time taken for accumulated rewards to reach an acceptable level of performance.

Two sets of experiments were carried out. First, LQR and MPC were used in conjunction with our transformation applied after the parametric change to the system. Since these methods are model-based and deterministic, hyperparameter selection, system identification, and policy function formulation used in RL were not applied to these two methods. The matrices F_A, F_B are computed directly, given the parameter change value was known.

The results are tabulated in Table 5.1. As both Tables 5.1a and 5.1b show, the episodic rewards from π_t (our transformation) were within a standard deviation of, if not better than, the benchmark π_{lqr} and π_{mpc}.

In the second set of experiments, RL was applied (Table 5.2). For RL, π_s was obtained by running the PPO algorithm until the episodic rewards converged. The policy transformation applied to π_s was further

	Temperature	Spring	Pendulum	Cartpole
π_s on P_t	-51123.05 ± 17246	-233.23 ± 53	-271.66 ± 2	40.4 ± 3
π_{lqr} on P_t	$\mathbf{-904.00 \pm 1624}$	$\mathbf{-2.33 \pm 2}$	$\mathbf{-7.15 \pm 8}$	$\mathbf{500 \pm 0}$
π_t on P_t	$\mathbf{-904.01 \pm 1624}$	-4.22 ± 5	-7.61 ± 8.2	$\mathbf{500 \pm 0}$

(a) Applying transformation to LQR. π_t is compared with the policy π_{lqr}, derived directly on P_t. LQR policy is derived on the system linearized about equilibrium state.

	Temperature	Spring	Pendulum	Cartpole
π_s on P_t	-36478.06 ± 15043	-208.92 ± 67	-23.53 ± 24	79 ± 16
π_{mpc} on P_t	-1138.50 ± 2075	$\mathbf{-2.82 \pm 3}$	$\mathbf{-7.29 \pm 8}$	155 ± 45
π_t on P_t	$\mathbf{-533.23 \pm 954}$	-4.56 ± 6	-7.60 ± 8	$\mathbf{168 \pm 48}$

(b) Applying transformation to MPC. π_t is compared with the policy π_{mpc}, derived directly on P_t with a receding horizon of 5 time steps.

Table 5.1: Mean episodic rewards using LQR and MPC, compared with π_t.

fine-tuned using gradient descent. We represented the transformed policy as π_t, the source policy fine-tuned on P_t as π_s^*, the policy fine-tuned with transformation parameters as π_t^+, and the one excluding parameters as π_t^-. In the first subset of experiments, the F_A, F_B were known *a priori* (Table 5.2a). Then, the same experiments were repeated, where the transformations F_A, F_B had to be estimated from measured data (Table 5.2b). For both cases, $\mathcal{D}_s, \mathcal{D}_t$ used at most five episodes, amounting to no more than $2,500$ interactions with the system. This contrasts with the time steps taken by RL to converge to a policy.

Figure 5.2 shows the effect the fault had on accumulated rewards, and how our policy transformation caused a jumpstart improvement as the RL policy tunes control after the system has changed. A parametric fault was introduced once RL converged to a policy on P_s. This caused an abrupt fall in the rewards associated with P_t. Using RL iteratively to learn on P_t was slow, and sometimes unable to converge to a policy that worked to maintain the trajectory.

However, the policy transformation led to a jump-start improvement in the rewards. For simple linear systems such as temperature, the transformation was instantly optimal. For non-linear systems like the cart pole system, there was a smaller jump-start improvement, which led to a faster time to convergence. For both sets of experiments involving RL, the results show that the transformed source policy, derived from the identified target system, achieved comparable, if not better, performance than the source policy fine-tuned directly on the target task.

5.2.1 Conclusion

Our results demonstrate several key points. Our approach gets a jump-start improvement in performance after a parametric fault. Second, during RL, if not already converged, rewards convergence faster to specified thresholds (Figure 5.2). Third, when knowledge of transformation and dynamics are known, the source policy's transformation produced results that were similar to LQR and MPC being trained on P_t (Tables

	Temperature	Spring	Pendulum	Cartpole
π_s on P_t	-51123.05 ± 17246	-295.39 ± 70	-271.66 ± 2	56.8 ± 15.16
π_s^* on P_t	-1076.92 ± 1597	-281.33 ± 70	-271.66 ± 2	486.1 ± 23.33
π_t on P_t	$\mathbf{-904.02 \pm 1624}$	-2.50 ± 3	-7.24 ± 8	$\mathbf{500.00 \pm 0}$
π_t^- on P_t	$\mathbf{-904.02 \pm 1624}$	$\mathbf{-2.46 \pm 3}$	$\mathbf{-7.16 \pm 8}$	$\mathbf{500.00 \pm 0}$
π_t^+ on P_t	$\mathbf{-904.02 \pm 1624}$	-2.48 ± 3.14	-7.17 ± 8	$\mathbf{500.00 \pm 0}$

(a) Policy transformation with known F_A, F_B.

	Temperature	Spring	Pendulum	Cartpole
π_s on P_t	-51123.05 ± 17246	-295.39 ± 70	-271.66 ± 2	56.8 ± 15.16
π_s^* on P_t	-1076.92 ± 1597	-281.33 ± 70	-271.66 ± 2	486.1 ± 23.33
π_t on P_t	$\mathbf{-904.02 \pm 1624}$	-171.15 ± 175	-19.84 ± 7	156.5 ± 88.67
π_t^- on P_t	-904.33 ± 1624	$\mathbf{-110.60 \pm 169}$	-12.79 ± 8	$\mathbf{500.00 \pm 0}$
π_t^+ on P_t	$\mathbf{-904.02 \pm 1624}$	-134.11 ± 168.62	$\mathbf{-10.96 \pm 8}$	493.8 ± 18.6

(b) Policy transformation with unknown F_A, F_B. They are approximated using a buffer of measurements for 5 episodes of π_s on P_t.

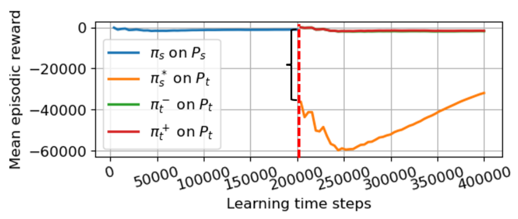
Table 5.2: Mean episodic rewards using RL.

5.1, 5.2a). However, unlike MPC, an optimization problem does not need to be solved recurrently when applying control, which reduces computational complexity. Finally, when knowledge of the transformation and dynamics are not sufficiently accurate because of the accuracy in the parameter estimation or the existence of nonlinear dynamics in the system, an approximate transformation using measured samples and fine-tuning using RL gives similar, albeit marginally lesser, results (table 5.2b).

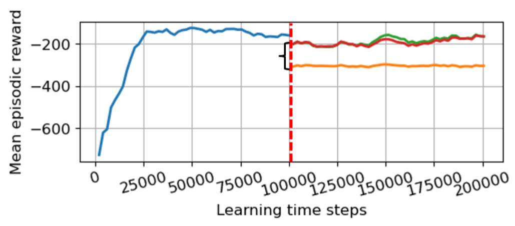
Therefore, our approach may lend itself as an initialization strategy for data-driven controllers to mitigate sample inefficiency. After the adaptation step, the controller can proceed to use online reinforcement learning methods to fine-tune the control policy.

We looked at transforming control policies by reasoning about task dynamics as a means of adaptive control, instead of the statistical properties of parameters, which is the approach adopted in machine learning. Our main contribution was the transformation of a nominal control policy that leverages system identification. It is applicable to a host of control algorithms, and tasks where the objective function is agnostic to actions. The transformation is such that a source policy would transfer positively on the target process with a higher sample efficiency than reinforcement learning.

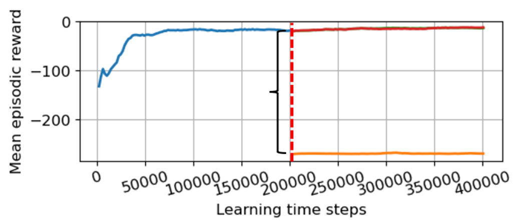
There are several interesting directions for future research. First, using the error in policy transformation, which may use pseudo-inverses, as a measure and guarantee of the quality of the transfer. Secondly, extending the transformation to a broader class of MDPs with non-linear disturbances. Thirdly, using parameter estimation, by relying on fault identification, instead of using system identification methods to further reduce the data samples to adapt to a new task.



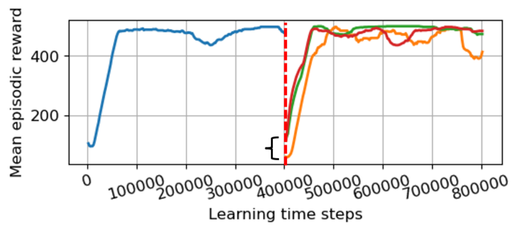
(a) Temperature system.



(b) Spring system.



(c) Pendulum system.



(d) Cartpole system.

Figure 5.2: Performance of RL trained on the source task π_s (blue), transformed for the target task π_t, and later fine-tuned on target task with (red) and without (green) fine-tuning the transformation too, π_t^+/π_t^-. Performance is compared against continuing to learn π_s to learn the target task (orange). The vertical line represents a fault occurrence.

5.3 Conditional policy transformation

Robust, reliable control of real-world systems requires the control policy to adapt to faults and disturbances that can be represented as parametric changes in the dynamic system model. Classic control methods, such as model-predictive control, rely on modeling such phenomena to be able to account for their effects (Wang et al., 2015). However, accurate modeling and parameter estimation of physical systems, especially after fault occurrences, is a complex problem. Therefore, designing controllers that can adapt to faulty behaviors is also a difficult task (Liu and Li, 2019). Online machine learning (ML) approaches may mitigate these complexities to some extent by learning a data-driven policy from the operational data of the system. However, ML approaches come with their own challenges. These include sample inefficiency, stochasticity, and explainability, among others (Paleyes et al., 2020).

How well a data-driven controller accommodates changes in the system dynamics can only be ascertained in hindsight: after it has operated in the modified environment and has collected sufficient data to compute metrics that evaluate its current performance. To adapt, a data-driven controller needs to interact with the environment in the first place. This creates the need for addressing the trade-off between the speed of adaptation versus the certainty in the quality of adaptation. In other words, fast convergence of the controller to required performance specifications is important to maintain the safe operation of the system.

The main contributions of this paper are 1) a physics-based policy transformation for sample efficiency improvement, and 2) a measure of optimality for the derived transformation towards guaranteeing a monotonic improvement in on-policy RL methods. First, we define a model-based policy transformation inspired by system identification techniques that combined with on-policy RL methods can quasi-optimally adapt to faults in non-linear systems. The proposed transformation is derived from a linear approximation of the state equation model of the system and faults are represented as changes in the physical parameters of the system. The aim of the transformation is to provide a jump-start improvement toward achieving optimal performance after fault occurrence. Second, we introduce a conditional step, where the quality of an initial policy transformation, derived using a model-based approach, is quantified before the transformed policy is employed for fault adaptation. Our approach is similar to meta-learning, in that the policy transformation is a parameter initialization step, which can lead to a *jump-start* improvement on a new task. The rest of the paper is organized as follows. Section 5.3.1 describes related work on the topic. Section 5.3.2 presents the problem to be solved and the respective proposed solution. In section 5.3.3 we validate the proposal through non-linear cases studies. Finally, we present the conclusions and future work in section 5.3.4.

5.3.1 Related work

5.3.1.1 Classical adaptive control

Åström and Wittenmark (2013) define an adaptive controller as one "with adjustable parameters and a mechanism for adjusting the parameters". Adaptive control can be viewed as a nested control loop: the inner loop actuates the system, and the outer loop manipulates the operating parameters of the inner loop. Methods such as adaptive model-predictive control (MPC), periodically estimate the process parameters. The inner predictive control loop applies model-based control. Other approaches like linear quadratic regulators (LQR), linearize the process dynamics at different operating points. The inner loop then solves a sequence of linear problems to obtain an optimal action.

Adaptive Proportional Integral Derivative (PID) controllers Anderson et al. (1988); Tjokro and Shah (1985) have a storied history, where controller gains are tuned from feedback signals. More recent work has involved adaptive MPC (Adetola et al., 2009; Pereida and Schoellig, 2018; Chowdhary et al., 2013) to estimate process model parameters under uncertainties for fast dynamics.

5.3.1.2 Adaptive machine learning

Adaptive classical controls literature parallels meta-reinforcement learning (Schweighofer and Doya, 2003; Santoro et al., 2016) in the field of machine learning. The inner loop is the *base learner*, which optimizes a single optimal control problem. The outer loop is the *meta learner*, which updates the controller parameters so that they perform well across control problems.

For example, recent work by Richards et al. (2021) uses *offline* meta-learning to pre-train an adaptive controller using data-driven simulations. The controller can actuate a drone to follow trajectories under wind disturbances.

5.3.2 Conditional policy adaptation

We develop the details of the conditional policy adaptation approach in this section. For the proposed approach, $F_\pi : \Pi \rightarrow \Pi$ is the adaptive outer loop that tunes parameters of the inner control loop, i.e., the reinforcement-learned policy π. We hypothesize that the adaptation step will speed up RL convergence because it initializes policy parameters close to an optimum. Our approach comprises the following elements:

1. A fault signal that triggers the policy adaptation algorithm for the changed, target task T_t;
2. A buffer $\mathscr{D}$ of states and actions measured from the extant controller's interaction with the target (faulty) system;
3. A time period between the signal and policy adaptation, where the system is deviating from a nominal state trajectory, but it does not lead to a catastrophic change in the system. In other words, we assume that if a transformed policy is deployed in this interval, the system behavior may be restored to its specified trajectory. We also assume that the parameter associated with the fault can be estimated in this time period; and
4. A quantification measure that is correlated with the expected returns of the transformed policy on the target task.

5.3.2.1 Assumptions & Limitations

We make the following assumptions:

Assumption 5. *Reward is time invariant and action-independent.*

For example, for popular RL systems like Cartpole, Acrobot, Car Racing, the reward is not a function of actions, only the eventual states. But for Pendulum, Mountain Car, Bipedal Walker, and Lunar Lander, the reward function penalizes the magnitude of actions. For other environments, such as those defined by the MuJuCo engine, a `ctrl_cost` parameter is provided to weigh the importance of "effort" - action magnitudes - over "performance" (state). For our real-world applications with unmanned vehicles, we can include the state of charge as a state variable, and have action magnitudes limited to safe bounds, as a proxy for action regularization in the reward function.

Assumption 6. *A fault diagnoser is available.*

The policy adaptation is preempted by a signal detecting whether system dynamics have changed (Qin (2012); Frisk et al. (2017)). Then, using system identification approaches, the target process model can be learned.

Assumption 7. *The controlled system is locally linear.*

For the ideal case of a linear system, an optimal source policy, and a parametric fault represented as a linear transformation of A, B matrices, our proposed transformation alone will be optimal. For the case of continuous non-linear systems, local linearity may be assumed in some cases. We define those conditions as follows. Given that P_n, P_l are the nonlinear process and its linear approximation $X \times U \to X$, and $r : X \times X \to \mathbb{R}$ is the time-invariant, state-only reward function, then the optimal policy between the process and its approximation will remain invariant so long as the following condition is met:

$$\forall x, u \; \texttt{sign} \nabla_x r(P_n(x,u)) = \texttt{sign} \nabla_x r(P_l(x,u)) \tag{5.18}$$

That is, a control policy derived on the linear approximation will be optimal on the original non-linear process. This is because there are no optima on the reward function surface under states traversed by the nonlinear process and not under its linear approximation. Therefore the ordinal relationship between states' rewards (and thus value, which is sum of future rewards) will remain invariant. And therefore the optimal policy will be identical.

This derives from the concept of monotone preferences in microeconomics (Mas-Colell et al., 1995) and resulting research on policy invariance by Ng et al. (1999). From the former, namely, that the ordinal properties of the utility value function, therefore control policy, are preserved under a monotonic transformation.

Given,

$$V(x_i) = \max_{u_i \in U} r(x_i) + \gamma V(x_{i+1}) \tag{5.19}$$

$$\pi(x_i) = u_i = \arg\max_{u \in U} V(x_{i+1}) \tag{5.20}$$

Then, if V_t is a monotone transformation of V_s,

$$V_s(x_1) < V_s(x_2) \iff V_t(x_1) < V_t(x_2) \tag{5.21}$$

$$\implies \arg\max_u V_t(x) = \arg\max_u V_s(x) \tag{5.22}$$

$$\implies \pi_s = \pi_t \ \forall x \in X \tag{5.23}$$

This is true, since an optimal policy samples actions with optimal value. And $\arg\min$ and $\arg\max$ operations, determining optimal sampling, are invariant under a monotone transform. As a corollary, we establish a condition on a value function representing a monotonic transform. For an optimal policy to remain invariant:

$$\lim_{\delta x \to 0} \texttt{sign} \nabla_x V_s(x + \delta x) = \texttt{sign} \nabla_x V_t(x + \delta x) \tag{5.24}$$

Since V is a sum over rewards, the critical component determining optimal policies as the range and the domain of the reward function $r : X \to \mathbb{R}$. Tasks $T \in \mathscr{T}$ may be related by a transformation of their dynamics, thus changing the path across the reward surface charted by a policy, thus possibly changing valuation of states. Expanding equation 5.24,

$$\texttt{sign} \nabla_{x,u} r(P_s(x,u)) = \texttt{sign} \nabla_{x,u} r(P_t(x,u)) \tag{5.25}$$

Linearization of dynamics should preserve ordinality of states with respect to the reward function. That is, for any change $\delta x, \delta u$ whatever direction of change in r under P_s is seen under P_t. That is, there is no optimum on one of r_s, r_t and not the other, when going from $x \to x + \delta x, u \to u + \delta u$. Figure 5.3 shows an example of this case, where a linear approximation of a higher-order system keeps the optimal policy invariant.

As an illustrative example, we use a linear spring-mass system with a two-dimensional state space and a one-dimensional action space, where the objective is to reach the rest position and have zero velocity in the shortest possible time. The dynamic model of this system is presented in the next section.

5.3.2.2 Condition for policy transformation

Policy transformation may be applied after fault occurrence. For faults of small magnitude that do not cause significant changes in performance, the transformation step is ineffective. We introduce a measure for gauging the sub-optimality of a source policy π_s after a fault occurs in the system. We leverage the actor-critic

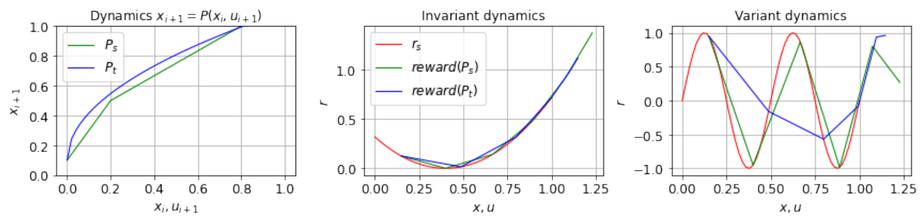


Figure 5.3: An optimal policy is invariant under a reward function with different dynamics when the there are no unique optima between state-action transitions between P_s and P_t. The first figure depicts two process dynamics, where one can be assumed to be a piecewise linear approximation of the other. The following figures show traversal across the cost surface by each system's dynamics, given an initial state. Under one cost function, the ordering of states by reward is preserved. Whereas in the other case, the existence of a unique optimum does not preserve ordinality.

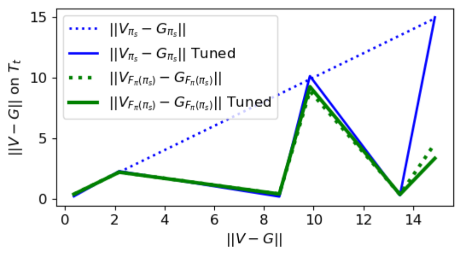
architecture, i.e., General Policy Iteration, along with a model-based approach to estimate the fault parameter and derive the policy transformation.

After a fault occurs, a small buffer $\mathscr{D}$ of system interactions for the target behavior, T_t is generated, and the expected returns under π_s are computed as, G_{π_s,T_t}. We use the comparison of returns with the value function as a performance metric, instead of the rewards directly. This helps us leverage the knowledge about policy optimality from the value function, which remains invariant from T_s to T_t. For an optimal policy for a task, expected returns estimate the value of each state. Therefore, the difference between expected returns and value measures the optimality of a policy.

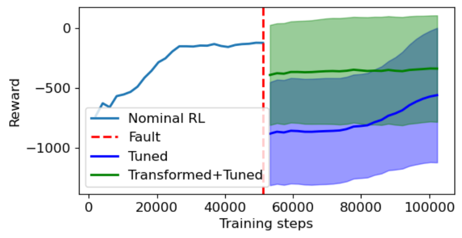
$$\varepsilon_G = ||V_s(x) - G_{\pi_s,T_t}(x)|| \tag{5.26}$$

Figure 5.4a shows how ε_G, the error in returns, gets smaller when a policy transformation and/or tuning are applied once the fault is detected and estimated. However, for values of ε_G that are already small, the transformation's effect is also small. This step can be computationally expensive, and, therefore, deriving F_π can be foregone if a condition about the magnitude of fault exceeding a threshold is not met. (For specific problems, this threshold can be determined empirically). In this case, ε_G of up to ≈ 2 yields no improvement when a transformation is applied.

The greater the difference in the above equation, the further from optimal the current policy π_s is in the target task, T_t. Therefore, the gradient of this error provides a measure of the improvement in the adaptation step when switching from T_s to T_t.



(a) A plot of faults, ordered by $\varepsilon_G = ||V_s - G_{\pi_s,T_t}||$, comparing the difference between value and returns using a policy transformation $F_\pi(\pi_s)$, with and without further tuning using RL.



(b) Performance of policy transformation over faults of varying magnitudes introduced after nominal training has finished.

Figure 5.4: Policy transformation on the linear spring-mass system.

$$\Delta\varepsilon_G = ||V_s(x) - G_{\pi_s,T_t}(x)|| - ||V_s(x) - G_{F_\pi(\pi_s),T_t}(x)|| \tag{5.27}$$

However, equation 5.27 is a *retroactive* measure, since returns under the new policy must be computed after interfacing π_t with T_t. In the following section, we derive a measure for the quality of transformation that is dependent only on $\mathscr{D}$.

5.3.2.3 Quality of policy transformation

Theorem 2. *If a policy is optimal, $\partial V(x_{i+1})/\partial\pi(x_i)$ will be zero, and $\partial^2 V(x_{i+1})/\partial\pi(x_i)^2$ will be negative semi-definite.*

Proof. An optimal policy π takes actions to states with the highest value. If a policy is optimal, any change in the policy will lead to a next state, $x_{i+1} = P(x_i, \pi(x_i))$ with an equal or lower value. □

In our problem formulation, we preserve the optimal state trajectories between the source and the target tasks. However, an error in the optimal policy on the target task, regardless of source, will produce deviations in the state trajectory. Since the accumulated value of states is sensitive to the state trajectories, a small policy error will lead to sub-optimal state trajectories, whose deviations will increase with time.

In an ideal case, when the system changes due to a fault, the process dynamics of the updated model taking into account the fault magnitude will be accurate. This also requires that the matrix products in equation 5.7 are invertible. However, if either one of these conditions is untrue, the adapted policy π_t derived by applying the transformation will be inaccurate. Therefore, the expected returns under π_t may not be maximized, which means that π_t is not optimal.

We consider the two cases that introduce such a policy error: (1) where the target dynamics $F_A A$, $F_B B$ have an approximation error, and (2) cases where the transformations or the system matrices are not invertible.

We use the observation in theorem 2 to quantify the quality of the transformed policy π_t. We estimate, for each state, how the value function on the source task V_s changes with respect to the transformed policy on the target task.

A zero gradient implies the source value function is at an optimum (or an inflection point) for the target task. The second partial derivative test can be used to determine if the optimum is a local maximum. A non-zero gradient implies that the state is not at an optimum under π_t. Figure 5.5 plots the gradient magnitudes for tasks of varying faults. The nominal task with the smallest ε_G has gradients mostly distributed around 0. Tasks with increasing faults start exhibiting some states where the value function is not at an optimum. Thus, after a policy transformation, gradients too large may be cause for pause before applying the new policy.

$$\varepsilon_V = \nabla_{\pi_t(x_i)} V_s(x_{i+1}) = \nabla_{x_{i+1}} V_s(x_{i+1}) \cdot \nabla_{\pi_t(x_i)} x_{i+1} \tag{5.28}$$

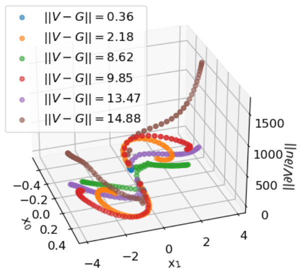
$$\begin{aligned} \nabla_{\pi_t(x_i)} x_{i+1} &= \nabla_{\pi_s(x_i)} x_{i+1} \cdot (\nabla_{\pi_s(x_i)} \pi_t(x_i))^{-1} \\ &= \delta t B_s ((F_B B_s)^{-1} B_s)^{-1}, \end{aligned} \tag{5.29}$$

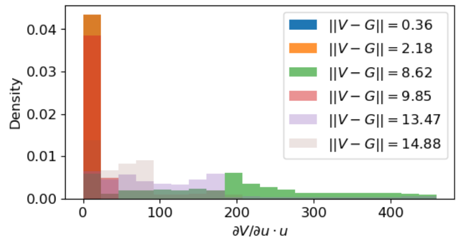
where δt is the time interval over which the change in equation 5.29 is being measured. This is a design-time constant and may be ignored for comparison.

Actor-critic reinforcement learning algorithms approximate value and policy functions for control tasks. Most implementations of such algorithms use neural networks as function approximators with automatic differentiation. This trivially gives us $\nabla_x V, \nabla_x \pi_t$.

Figure 5.6 shows how, for a linear system, $\Delta\varepsilon_G$ can equal ε_G. That is, the potential for improvement in error in returns is realized. The spike in value gradients is for the fault where the input does not apply any force to the system.

If the policy transformation is approximate, then the uncertainty in the target policy will be larger. The policy transformation requires the inversion of the $B_t = F_B B_s$ product. If a matrix M is not invertible, a Moore-Penrose pseudo-inverse, M^+ is employed. The error of the pseudo-inverse can also quantify the accuracy of the inversion. For an invertible matrix, the pseudo-inverse is the same as the inverse. Therefore, how close M^+M is to identity will be a measure of the invertibility of the matrix. We use the matrix norm as the inversion error, ε_i, as shown in equation 5.30.





(a) $\partial V/\partial u$ for a spring mass system, plotted against the two state variables x_0, x_1, representing position and velocity. Most states in the disturbed system show no change in value. However, the values associated with some of the states are sensitive to the magnitude of the fault.

(b) $\partial V/\partial u \cdot \delta u$ for a system with increasing fault magnitudes, considering the estimated change in value due to $\delta u = \pi_s(x) - \pi_t(x)$. Larger fault magnitudes lead to a larger change from the nominal state value calculated on T_s.

Figure 5.5: Plotting magnitude of ε_V for tasks with increasing fault magnitudes.

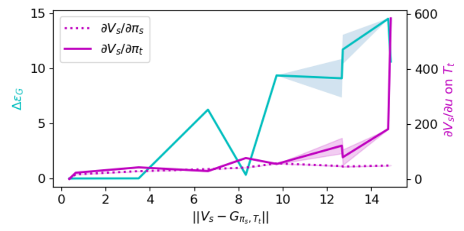


Figure 5.6: Improvement in error in returns compared against sensitivity of value function to change in policy. The gradient norms $\partial V/\partial \pi$ are computed over a buffer of states on the target task T_t.

$$\varepsilon_i(M) = ||M^+ M - I|| \tag{5.30}$$

Furthermore, the transformation step estimation of the fault magnitude, which is a parameter of the A and B matrices. For this work, we approximate A_s, B_s from the nominal system, and update the A_t, B_t matrices, by estimating the fault parameter from state transition samples generated in $\mathscr{D}$. For parameter estimation, we employ a nonlinear least squares estimator that assumes a parabolic error surface and then apply a gradient descent method to minimize the mean square error in system behavior (Bregon et al., 2011). The error in the system identification is represented as the prediction error of the model dynamics. Where, $P_{t'}$ is the learned model, the error in dynamics ε_D will be:

Algorithm 10 Conditional policy adaptation.

```
Require: $T_t$
Require: $T_s, \pi_s$, Data buffer $\mathscr{D}$
1: Collect $(x_i, u_i, x_{i+1})$ into $\mathscr{D}$ using $\pi_s$
2: Check whether to transform using equation 5.26
3: Learn $T_t$ dynamics, $F_A, F_B$
4: Use equation 5.7 to get $\pi_t$
5: Predict suitability of $\pi_t$ using equations 5.31, 5.30, 5.28
6: Return $\pi_t$
```

$$\varepsilon_D = \mathbb{E}_{x_i, u_i, x_{i+1} \sim \mathscr{D}} ||P_{t'}(x_i, u_i) - x_{i+1}|| \tag{5.31}$$

5.3.2.4 Conditional policy adaptation

Combining the two sources of errors gives a decision mechanism for avoiding negative transfer to the target task. Whereas ε_V represents the change in value due to a change in system dynamics, $\varepsilon_i, \varepsilon_D$ represent the uncertainty in the measurement of change itself.

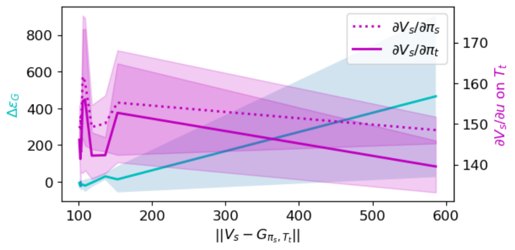
The effect of errors in task approximation on the optimality of the transform is used to conditionally update the source policy when the process changes. This is intended to prevent any negative adaptation from taking place. During such intervals when the policy adaptation is withheld, the extant learning algorithm can iteratively improve on the task as usual. The whole approach is outlined in algorithm 10.

5.3.3 Experiments

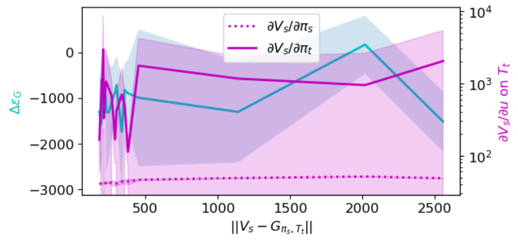
This work is demonstrated by several experiments conducted on versions of a nonlinear system. A nominal RL policy is trained, which is then adapted on tasks with increasing fault magnitudes, represented by ε_G. We use the "Lunar Lander" environment used in reinforcement learning and optimal control literature, where $x \in \mathbb{R}^8, u \in \mathbb{R}^2$. The objective is to land upright, with minimal residual velocity. The system is controlled via a main rocket to slow descent, and side thrusters to tilt the lander to direct thrust. Faults in the system are modeled as imbalances in the side thrusters. Specifically, the right thruster experiences a loss of effectiveness from $0-25\%$. First, we re-purpose the Lunar Lander system into a "Lander" environment. The Lander version of the environment exhibits continuous nonlinear dynamics. That is, collision physics with the ground and lander legs are not simulated. This is to ensure that the system can be modeled as a continuous system in the last leg of its flight. And system behavior, though nonlinear, can be locally linearized to apply the policy transformation as an approximation. The transformation is approximate because the target task dynamics, represented by A_t, B_t, are estimated from a small buffer of interactions.

Figure 5.7a shows improvement, $\Delta\varepsilon_G$, plotted against error in returns, ε_G, and the magnitudes of the gradients of the value function before the transformation is fine-tuned further using an online RL approach. Like the ideal, linear case in 5.6, a larger error in returns signifies a larger potential for improvement. With small ε_G, negative improvement is observed. A small enough ε_G may get over-corrected by a policy transformation that has errors due to the approximation in fault parameter estimation, and, therefore, the A_t and B_t matrices. Whereas for larger ε_G, while $\varepsilon_D, \varepsilon_i$ exist, the approximate policy transformation may still provide a positive jumpstart that leads to faster convergence to optimality.

Next, we demonstrate the necessity of a local linearity assumption. We use the separate Lunar Lander environment with collision physics enabled. Such mechanics adds discontinuities in state variables in the final leg of the flight when contact with the ground is made. They cannot be modeled accurately by a linear model. Figure 5.7b shows the results of policy transformation before any further RL is used for tuning. There are several notable differences from the ideal, linear case. $\Delta\varepsilon_G$ is negative, indicating a negative adaptation step. This is reinforced by the relatively larger gradient magnitude of the value function under the adapted policy π_t. Figures 5.8b support this by showing negative transfer after transformation. Invariably, re-using the extant policy and fine-tuning it yields better returns on that target task.

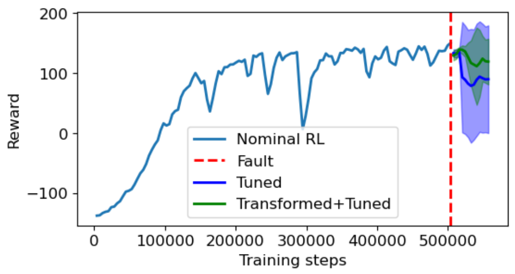


(a) Lander system.

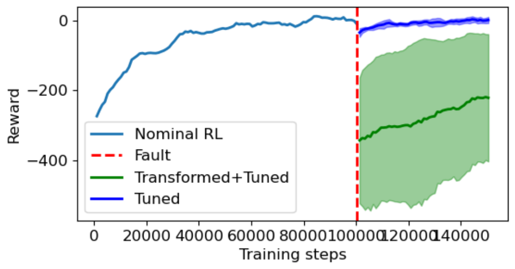


(b) Lunar Lander system. Note the logarithmic scale.

Figure 5.7: Comparing change in error in returns $\Delta\varepsilon_G$ with estimated gradient magnitudes $\partial V(x_{i+1})/\partial \pi_t(x_i)$ and $\partial V(x_{i+1})/\partial \pi_s(x_i)$. Where $x_i \in \mathscr{D}$.



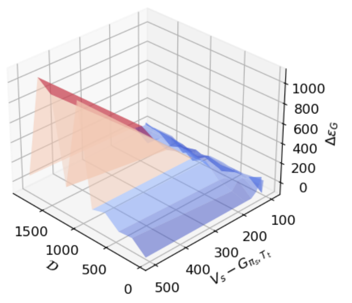
(a) Lander system.



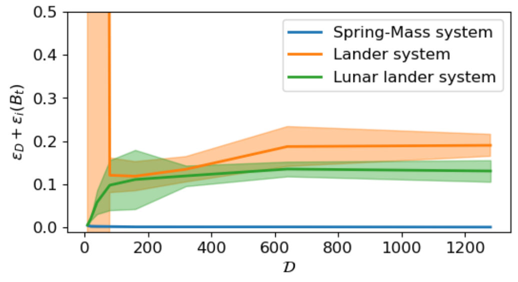
(b) Lunar lander system

Figure 5.8: Effect of policy transformation on the two lander systems. For a locally linear system, the adaptation step is, on average, able to mitigate faults in the form of a jumpstart improvement on the faulty T_t.

The amount of interactions with T_t, stored in $\mathscr{D}$, is considered a design-time parameter. We investigate the effect of varying buffer size on the accuracy of the model, and the resulting performance of the transformed policy. Figure 5.9 illustrates the effect of varying buffer size on the eventual improvement $\Delta\varepsilon_G$ and the approximation error ε_D on the Lander system. Larger buffer sizes *and* larger ε_G are related to a larger improvement. This is explained in figure 5.9b. It shows how dynamics in smaller buffer sizes may be captured by a linear model, giving a small ε_D. However, the small size itself fails to capture the change in system dynamics after the fault occurs. Whereas a large buffer, despite capturing the behavior of a non-linear system provides a much larger error when approximated by a linear model. This gives a larger ε_D. Regardless of the larger approximation error, under a larger return error, the approximate policy transformation is able to get a jump-start over reusing the nominal policy.



(a) The effect of the buffer size on tasks. For tasks that are similar, where ε_G is smaller, the data-driven policy transformation may have errors larger than the change an ideal transformation would require. This causes negative adaptation for smaller values of ε_G.



(b) Approximation error due to system dynamics, ε_D, and the invertibility of B_t, $\varepsilon_i(B_t)$, as the buffer size is increased.

Figure 5.9: The effect of buffer size $\mathscr{D}$ on policy transformation.

5.3.4 Discussion

In this work, we have an explicit dynamic system model for the source task, T_s. However, when a fault occurs in the system, we need to estimate the fault parameter. We linearize and update the A, B matrices of the model, derive and apply the transform, and make sure it provides a jumpstart. Then, we use online RL to converge to a better solution. We present two measures for quantifying adaptation. One measure, ε_G, relates to the viability of an adaptive step in the first place. The second measure, ε_V relates to the adapted policy being close to an optimum. The two measures rely on a small buffer of interactions with the new system. They are also readily calculable given the theoretical and software architecture of actor-critic RL algorithms.

We observed that systems with locally linear characteristics tend to perform well with policy transformation, which our two measures agreed with. We also observed that non-linearities may cause policy transformation to cause negative transfer, which our measures also agreed with.

Notably, the policy transformation relation in equation 5.7 does not enforce a linearity assumption, except when the arguments are constant matrices. It can be refactored as $(B_t)^{-1}\left[(F_{x,s}(x,\pi_s(x))-A_s x\right]$. Indeed, F_A may be approximated by neural networks to capture non-linear dynamics. The function $B_t = F_B B_s : U \rightarrow X$, however, needs to be invertible. This restricts the choice of its representation. Our future work will examine the use of different basis functions for capturing target task dynamics more accurately.

Furthermore, there is potential for improving the subsequent online RL. ε_V, relating to the optimality of the value function under the transformed policy, may be used to condition the learning rate for faster convergence on the new task.

5.4 Summary

In this chapter, model-based adaptation via policy transformation was presented. The strongest results are for systems that are linear, where the policy transformation represents an instantaneous adaptation step towards optimal control. In systems where local linearity is assumed, the transformation acts as an initialization step for the policy function, akin to model-agnostic meta-learning. The transformation initializes policy parameters closer to the optimum on the target task. Unlike MAML, the initialization step is motivated from knowledge of process dynamics.

This approach is also conditional on the reward function being a function of state only. This holds true for many pedagogical systems popular in RL and controls literature. For example, for popular RL systems like Cartpole, Acrobot, Car Racing, the reward is not a function of actions, only the eventual states. But for Pendulum, Mountain Car, Bipedal Walker, and Lunar Lander, the reward function penalizes the magnitude of actions. For yet other environments, such as those defined by the MuJuCo engine, a `ctrl_cost` parameter is provided to weigh the importance of “effort” - action magnitudes - over “performance” (state). For real-world applications, such as with UAVs, we can include the state of charge as a state variable, and have action magnitudes limited to safe bounds, as a proxy for action regularization in the reward function.

The following section uses RL for control of a more complex system. A high-fidelity UAV testbed is developed and coupled with RL under disturbances and faults to demonstrate the utility of this work.

CHAPTER 6

UAV testbed for simulation and validation of research

The versatility of Unmanned Aerial Systems (UASs), especially, multi-rotor systems,is leading to wide adoption for civilian applications (González-Jorge et al., 2017). With the advances in battery technology, computational power, and communications architecture, vehicle autonomy is becoming a tractable problem. Efforts have been made to weigh safety against autonomy to meet legal and technological constraints (Vidyadharan et al., 2017). Similarly, the advent of deep learning approaches in the controls domain has yielded novel approaches to addressing UAS control problems (Jagannath et al., 2021). The utility of a virtual simulation environment for validating new approaches towards UAS autonomy is arguably higher than ever.

UASs are complex electro-mechanical systems, where control actuation occurs in the order of milliseconds. They are put under stressful maneuvers and undergo natural component degradation over the course of their operation. Faults can occur in motors, sensors, and batteries. Testing fault-tolerant control on a physical system presents its own safety and cost problems. A virtual test-bed, therefore, can serve as a low-cost alternative to developing and testing fault-tolerant control applications before hardware-in-the-loop testing and actual deployment.

Octo-rotor UASs present additional redundancy with motor failures, but they cost more and are used in more performance-critical domains. There is a body of work on control applications of fixed-wing and quad-rotor UAVs. In this work, we propose a virtual simulation environment for modeling octo-rotor operations, especially in the presence of faults. The environment is built using a simulation application, which is popular with robotics applications.

The rest of the section is organized as follows. We review extant simulation environments, UAS control, and machine learning approaches for UAS control. Section 6.4 presents the test-bed architecture. Finally section 6.8 demonstrates application of reinforcement learning for fault tolerant attitude control of a octo-rotor.

There has been a some work done on simulation and control of UAVs in the context of fault-tolerant controlFourlas and Karras (2021). Classical controls approaches have benefited from tools developed in domain-specific environments like `MATLAB`Schacht-Rodríguez et al. (2018). The recent prevalence of data-driven control approaches intersects with machine learning. There dynamic, interpreted languages like Python are popular, and have a monumental repository of software libraries that support networking, graphics, optimization, and machine learning. This work leverages this infrastructure to provide a user-friendly programmable interface which is inter-operable with other software research in this field.

6.1 Software simulation environments

Existing UAV simulation frameworks are surveyed in Hentati et al. (2018). A unified interface and simulation framework geared for reinforcement learning based tasks was presented in the OpenAI Gym library Brockman et al. (2016). The simulation "environment" is a Python class, which has methods to step through actions and reset to an initial state. Method arguments and return values are standardized in the OpenAI Gym library, which makes such test-beds convenient to mix and match with control algorithms written with that interface in mind.

GymFC Koch et al. (2019) is framework using Gazebo Koenig and Howard (2004) to create UAV models and run UAV simulations. The user-facing interface is compatible with OpenAI Gym. Gazebo is a general purpose 3D physics-based simulation engine written in C++ that is used for robotics applications. GymFC uses Gazebo's built-in plugins to feed Pulse Width Modulated (PWM) signals to actuators for quadrotor control. Custom plugins written in C++ communicate states and actions to and from the Python interface over network ports. The plugins then simulate the dynamics of the vehicle, finally letting the built-in physics engine integrate through the forces and torques on the UAV. Alternative simulation platforms include `MuJuCo` Todorov et al. (2012), written in C++, and `ml-agents`Juliani et al. (2018), written in C#. However, with the integration of existing flight control software (Ardupilot) as plugins into Gazebo, it is favorably positioned for simulation and control research in this domain.

6.2 Multi-rotor control applications

The need for a programmable, customizable simulation testbed for UAV applications is evident from recent work involving a multitude of vehicles with varying configurations, geometries, and faultsHentati et al. (2018); Xian et al. (2017). An overview of linear, non-linear and other control approaches for autonomy in quadrotors is discussed in Zulu et al. (2014). Razmi and Afshinfar (2019) demonstrates sliding mode quadrotor attitude control with a neural network to learn sliding mode parameters. Razmi and Afshinfar (2019) presents co-axial octo-rotor control using PID/PD inner/outer loops. Marks et al. (2012) presents fault-tolerant control of an octo-rotor using a control reallocation scheme that accounts of actuator saturation, after a fault is detected. The simulation is able to recover attitude from failures of up to 4 motors. Saied et al. (2015) proposes FTC of co-axial octo-rotor using residual based fault detection, which then exploits built-in co-axial motor redundancy to accommodate faults by increasing motor speeds.

Recent work has used reinforcement learning-based control to achieve fault-tolerance in UAVs. Li and Xu (2020) employs General Policy Iteration and an inverse dynamics approach to learn value functions over vehicle state and control actions, and then takes valuable actions greedily. It is simulated on an octo-rotor UAV. In a similar vein, Fei et al. (2020) discusses RL-based quadrotor FTC under actuator and sensor faults

Table 6.1: Symbols reference

Symbol	Units	Description
A		Control allocation matrix $\mathbb{R}^{6\times p}$
$b, \hat{b}$		Body reference frame, axes
D		Controller-prescribed forces and torques $\mathbb{R}^{6\times 1}$
F	N	Force
g	ms^{-2}	gravitational acceleration
J	kgm^2	rotational inertia
k_T	Ns^{-2}	Propeller thrust constant
k_{EMF}, k_τ	N	Thrust and torque constants
M	Nm	Moment
$n, \hat{n}$		Inertial reference frame, axes
$\hat{\omega}$	s^{-1}	Angular rate of the vehicle
$\hat{\Phi}$		Orientation of the vehicle
Ω	s^{-1}	Rotation speed of propeller
p		Number of propellers
$\hat{r}$	m	Position vector of the vehicle
v	ms^{-1}	Velocity
$f_{\hat{r}}, f_{\hat{\Phi}}$	s^{-1}	Frequency of position, attitude control

via cyber attacks. It demonstrates that a RL policy trained on a model can be transferred to actual vehicle for fault recovery. System agnostic UAV control using RL is presented in Pi et al. (2021). A policy learns forces and torques needed to follow a reference trajectory. They are then individually allocated to quad- and hexa-rotor platforms.

6.3 Utility of a software framework

In this work, we present a framework for the simulation and control of an octorotor UAV digital twin platform using reinforcement learning. The framework allows introduction of motor faults and simulation of electronic control of the platform, such as battery effects, dynamically during the simulation. The Gazebo simulation environment is used purely as a physics engine to translate the effect of the electrical state of the system into forces and torques.

6.4 The Test-bed

6.4.1 Architecture

The simulation test-bed includes two virtual processes that communicate via a network protocol. This work builds on previous work on the `GymFC` test-bed for quad-rotors (Koch et al., 2019). The control process includes a python interface, and the simulation process is a Gazebo server modeling the physics of octo-rotor flight. Different from `GymFC`, the net forces and torques acting on the body are calculated in python. This test-bed also implements a physics simulation of the vehicle with a python back-end for validation against Gazebo.

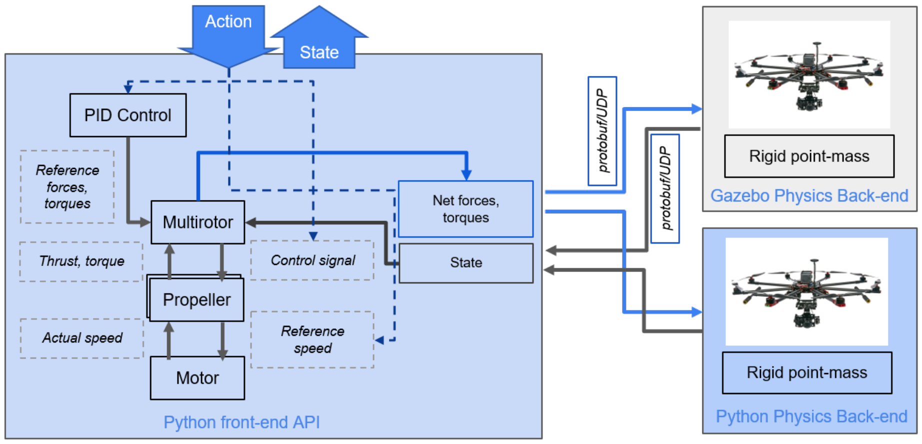
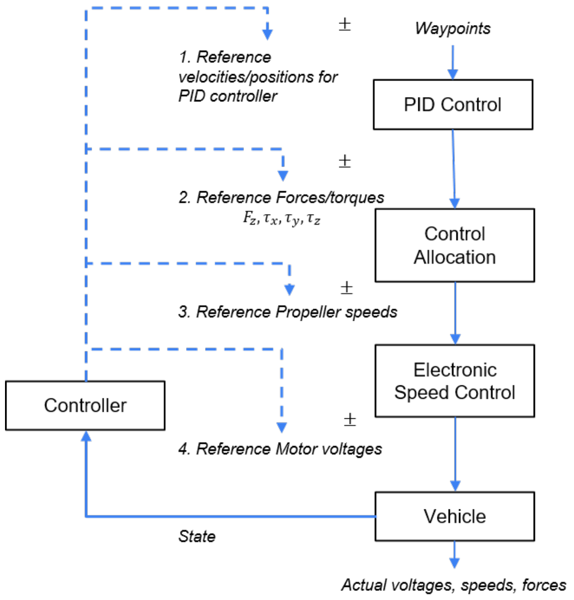


Figure 6.1: System architecture. The python-only code base is able to function with/out a Gazebo back-end for faster design iteration. Systems simulation is object oriented and mirrors compartmentalization in a physical vehicle.



Waypoints
±
1. Reference velocities/positions for PID controller
PID Control
±
2. Reference Forces/torques
$F_z, \tau_x, \tau_y, \tau_z$
Control Allocation
±
3. Reference Propeller speeds
Electronic Speed Control
±
4. Reference Motor voltages
Controller
Vehicle
State
Actual voltages, speeds, forces

Figure 6.2: The control logic flow for a UAV. In the standard approach, the position reference is converted by the PID controller into prescribed dynamics needed to follow that reference. The vehicle then converts the desired dynamics into propeller speeds which will generate them. The electronic speed controller actuates motors based on the speed signals, resulting in motion. Finally, the vehicle reports its resulting state vector back to the controller.

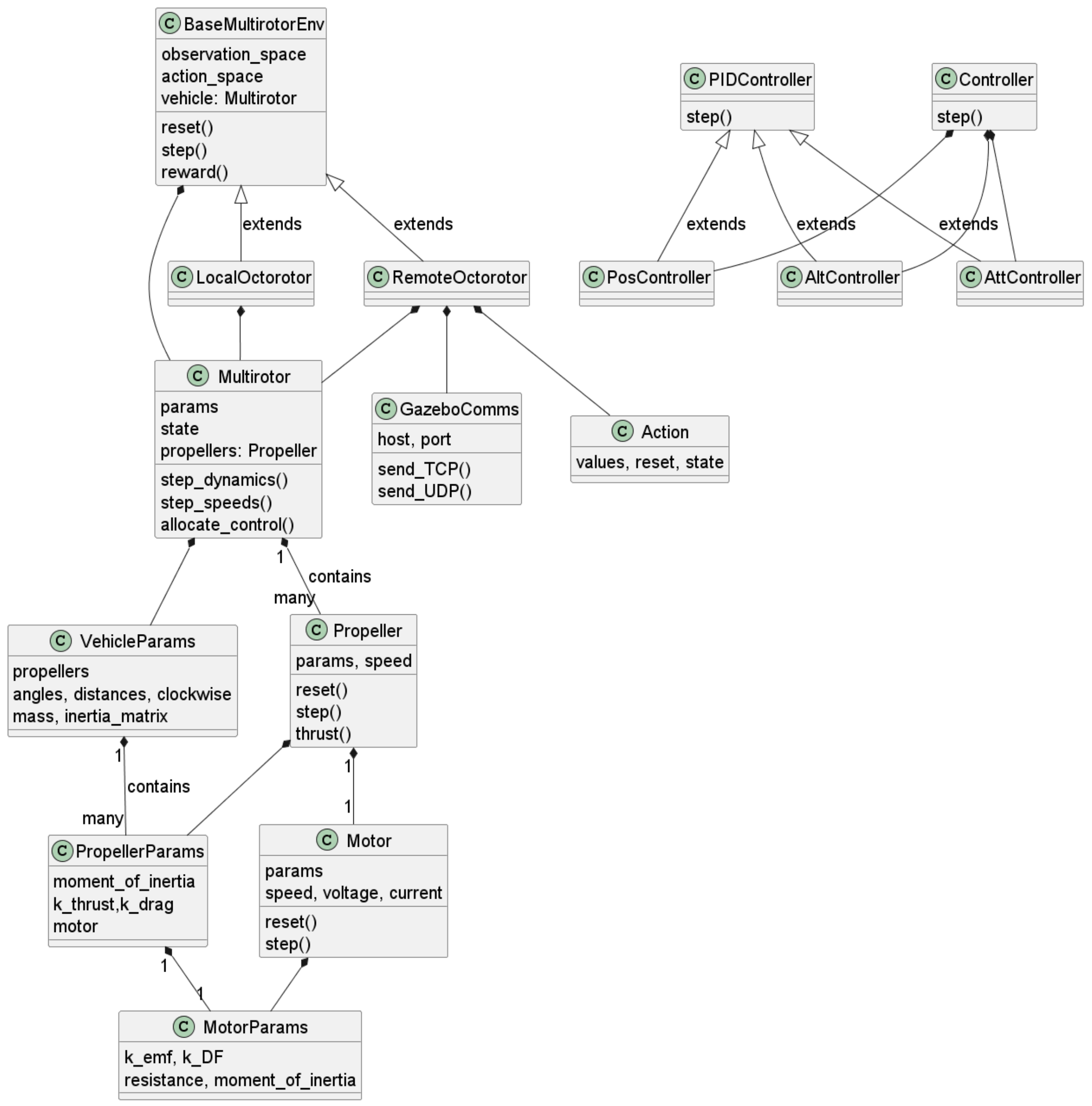


Figure 6.3: A class diagram of the exposed programming interface showing the object-oriented approach. A highlight of attributes and methods is provided.

The environment is modeled as a Markov Decision Process. It has a state space $\mathscr{X} \in \mathbb{R}^{12}$ that includes position, velocity, orientation, and angular rate. The action space is either the net forces and torques acting on the center of mass, $\mathscr{U} \in \mathbb{R}^6$, or the speeds, or control signals to individual motors, $\mathscr{U} \in \mathbb{R}^p$. $p = 8$ for an octo-rotor. Additionally the reward $R : \mathscr{X} \times \mathscr{U} \times \mathscr{X} \to \mathbb{R}$ is implemented as a function depending on the objective of a particular experiment.

We demonstrate this framework by constructing a Tarot T-18 octo-rotor model. Physical simulation of the octo-rotor is done in the Gazebo simulation application. The Gazebo application runs a physics engine to simulate forces and torques on models. Plugins written in C++ can be embedded into the server, specific models and sensors, to manipulate the simulation.

An environment plugin runs a thread synchronously to the simulation progression to monitor for control actions. When it receives the action vector from the control process, it publishes it for Gazebo to pass on to the plugin embedded in the octo-rotor model at the next simulation update. The model plugin converts the motor speeds, using the propeller properties, into forces and torques acting on the octo-rotor model.

The simulation occasionally requires a reset. The reset signal is encoded as flag accompanied by the initial state post-reset. Upon receipt, the environment plugin places the octo-rotor at the location and orientation specified in the code.

In addition to the Tarot T-18 Gazebo test-bed, a python-only back-end is also presented. It is able to model and simulate different vehicle geometries and parameters. For this configuration, the control and simulation of the vehicle occurs in the same python process.

Gazebo uses the Google Protocol Buffer format (`protobuf`) for serializing messages over the network using TCP/IP sockets. Communications between Gazebo entities (models, sensors, etc.) are published to "topics" that other entities can subscribe to. Messages between the control and simulation processes are serialized the same way, using a `protobuf` message. The message object defaults to passing a 6-dimensional array of net forces and torques acting on the vehicle center of mass. To reduce latency and network overhead, instead of TCP/IP, User Datagram Protocol (UDP) is used.

The octo-rotor simulation is exposed as an OpenAI Gym environment for programmatic control in python. The environment necessarily implements the following interfaces:

The environment object is characterized by its `observation_space` and `action_space` attributes, and its `step`, `reward`, and `reset` methods. The attributes define the state space as a twelve dimensional vector. Action space can be an array of different values, depending on the nature of control, as seen in figures 6.1 amd 6.2, for example: motor voltages and reference attitude. The object-oriented environment setup ensures that any stage of the control flow can be modified.

To adhere to the OpenAI Gym specification, each step of the simulation response returns four objects:

```
environment = Octorotor()
state = environment.reset()
time = 0
abrupt_fault(environment)
while True:
    time = time + 1
    # determine action from controller
    action = function(state)
    state, reward, done, diagnostics =
        environment.step(action)
    incipient_fault(environment, time)
```

Figure 6.4: Pseudo code illustrating the programming interface.

1. `state` is an array of observed state values.
2. `reward` is a number representing feedback from the last action.
3. `done` is a boolean signalling whether the simulation episode has ended.
4. `diagnostics` is a python dictionary populated with optional information.

The implementation of reward, episode duration, and diagnostics signals is dependent on the specific simulation case.

6.4.2 Physical modeling

This section describes the modeling process of components of the vehicle. Starting from battery voltage, and going up via motor revolutions, resulting propeller thrust, vehicle dynamics, and eventual control. The mathematical representations herein borrow from prior work by Schacht-Rodríguez et al. (2018); Charles Tytler (2017).

Given the propeller geometry parameters, the thrust is modeled by numerically solving the equation for thrust and propeller induced velocity Stevens et al. (2015)(sec 8.2). An alternate approach, relating the thrust constant and propeller velocity can be used as well (6.1).

$$T = k_T \cdot \Omega^2 \tag{6.1}$$

Torque about the yaw axis is modeled as aerodynamic drag, which is equal to the net BLDC motor torque τ_{BLDC} near hover conditions. Given the drag coefficient k_d,

$$\tau = k_d \cdot \Omega^2 \tag{6.2}$$

The environment models brushless direct current (BLDC) motors independently as sub-objects. Each motor is parameterized by the back electromotive force constant, k_e, and the internal resistance R_{BLDC}. The torque constant k_τ is equal to k_e for an ideal square-wave BLDC. k_τ determines the driving moment of the motor. Dissipative moments are governed by the dynamic friction constant k_{DF}, and the aerodynamic drag constant k_d. Finally, the net moments τ and the moment of inertia J of the motor determine how fast the rotor spins. The state of each motor is its angular velocity Ω, applied voltage v_{BLDC}, and drawn current i_{BLDC}. The dynamics are given by equation 6.3.

$$\begin{aligned}
i_{BLDC} &= (v_{BLDC} - k_e \cdot \Omega)/R_{BLDC} \\
\tau_{BLDC} &= k_\tau \cdot i_{BLDC} \\
\tau &= \tau_{BLDC} - k_{DF} \cdot \Omega - k_d \cdot \Omega^2 \\
\frac{d\Omega}{dt} &= \tau/J
\end{aligned} \tag{6.3}$$

The vehicle is modeled as a rigid body. That is, there is no relative motion in the body frame attached to the vehicle. The assumptions and conventions are documented in section 6.5.

6.5 Dynamics of multi-rotor UAVs

6.5.1 Position and Orientation representation

A multi-rotor UAV is modeled as a rigid body with six degrees of freedom. A rigid body has a constant mass distribution relative to its center of gravity. The six degrees of freedom are the three spatial coordinates x, y, z and the three Euler angles ϕ, θ, ψ.

Two reference frames are used for representing the state of the body:

- Inertial, nominal reference frame n is the static frame of reference where the axes are aligned with arbitrary, global directions. They are represented as $[\hat{x}, \hat{y}, \hat{z}]$.
- Body-fixed reference frame b has the axes aligned with respect to the center of gravity of the rigid body in motion. They are represented as $[\hat{b}_1, \hat{b}_2, \hat{b}_3]$.

Orientation of a body in the inertial reference frame follows the Tait-Bryan angles convention. That is, rotation is specified as yaw (ψ), pitch (θ), and roll (ϕ) in that order. The order of rotations matters. Starting from the inertial frame, yaw ψ is rotation $R(\psi)$ of the body frame about the inertial z axis. Pitch θ is rotation about the y axis after the first rotation $R(\theta)\cdot R(\psi)$. And roll ϕ is the final rotation about the x axis of the frame after the prior two rotations. The final product is the body reference frame.

Given the position in the inertial frame $\hat{r}^n = [x,y,z]^T$ and the position in the body frame $\hat{r}^b = [b_1,b_2,b_3]^T$, the rotation matrix from the inertial to body reference frames R_n^b is defined as:

$$
\begin{aligned}
\hat{r}^b &= R(\phi)\cdot R(\theta)\cdot R(\psi)\cdot \hat{r}^n \\
\hat{r}^b &= R_n^b \hat{r}^n \\
\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} &= \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ -c\phi s\psi + s\phi s\theta c\psi & c\phi c\psi + s\phi s\theta s\psi & s\psi c\theta \\ s\phi s\psi + c\phi s\theta c\psi & -s\phi c\psi + c\phi s\theta s\psi & c\psi c\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
\end{aligned}
\tag{6.4}
$$

By convention, the right handed coordinate system is followed. The direction of the positive x axis in the body frame (b_1) is considered "forward" orientation. The positive y axis (b_2) is "left" and the positive z axis (b_3) is up. For rotations about each axis, positive rotation is counter-clockwise, looking at the positive rotation axis coming out of the page.

For the body frame with an angular velocity about its axes with respect to the inertial frame $\hat{\omega} = [\omega_x, \omega_y, \omega_z]$, the time-derivative of a vector $\hat{\mathscr{V}}^b = \hat{\mathscr{V}}\cdot\hat{b}$ in a rotating body frame is given by the Coriolis theorem:

$$
\begin{aligned}
\frac{d\hat{\mathscr{V}}^b}{dt} &= \frac{d\hat{\mathscr{V}}}{dt}\cdot\hat{b} + \frac{d\hat{b}}{dt}\cdot\hat{\mathscr{V}} \\
&= \frac{d\hat{\mathscr{V}}}{dt}\cdot\hat{b} + \hat{\omega}\times\hat{\mathscr{V}}^b \\
&= \frac{d\hat{\mathscr{V}}}{dt}\cdot\hat{b} + \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}\hat{\mathscr{V}}
\end{aligned}
\tag{6.5}
$$

6.5.2 The state of the vehicle

Tracking the motion of the vehicle requires tracking multiple variables. The variables can be divided into translational and their rotational analogues.

- $\hat{r}^n = [x,y,z]$ are the navigation coordinates.

- $\hat{v^b} = [\dot{x}, \dot{y}, \dot{z}]$ is the velocity of the vehicle along the body frame axes.
- $\hat{\Phi} = [\phi, \theta, \psi]$ is the orientation of the body reference frame b in Euler angles (roll, pitch, yaw) with reference to the nominal reference frame n.
- $\hat{\omega} = [\omega_x, \omega_y, \omega_z]$ is the roll rate of the three body frame axes with reference to the nominal frame.

The state variables above are affected by the forces and torques acting on the body.

- $\hat{F}^b = [F_{b_1}, F_{b_2}, F_{b_3}]$ are the net forces along the three body frame axes, where $\hat{F}^b = R_n^b \hat{F}^n$.
- $\hat{M}^b = [M_{b_1}, M_{b_2}, M_{b_3}]$ are the moments along the three body axes, where $\hat{M}^b = R_n^b \hat{M}^n$.

6.5.3 Equations of motion

The equations of motion relate the state variables the the dynamics of the system. The equations of motion are derived from Newton's second law of motion. Where, for translational variables:

$$\hat{F}^b = m\frac{d\hat{v}^b}{dt} = m(\dot{\hat{v}}^b + \hat{\omega} \times \hat{v}^b) \tag{6.6}$$

For rotational variables a similar analogue exists. Given M^b is the rotating body-frame moment, ω is the angular rate measured in the body frame (subject to the Coriolis effect), and I is the moment of inertia:

$$\begin{aligned} \hat{M}^b &= \hat{I}\frac{d\hat{\omega}}{dt} \\ \hat{M}^b &= \hat{I}\dot{\hat{\omega}} + \hat{\omega} \times \hat{I}\hat{\omega} \end{aligned} \tag{6.7}$$

6.5.4 Linear variables

The linear state variables ($\hat{r}^n$, $\hat{v}^b$) are determined primarily by the forces acting on the vehicle. The force of gravity $\hat{F_g}^n = [0, 0, -mg]$ in the nominal reference frame. In the body frame it becomes $\hat{F_g}^b = R_n^b \hat{F}^n$. The net force of the p propellers in the body frame is $\hat{F_p}^b = [0, 0, \sum_i^p T_i]$. Thus, the total force acting on the center of mass is $\hat{F}^b = \hat{F_p}^b + \hat{F_g}^b$. Thus solving for acceleration $\dot{\hat{v}}^b$ and equating with the acceleration acting on the body frame $\hat{F}^b/m$ yields the rate of change of velocity $\hat{v}^b$ in the body frame.

Using R_b^n and the current body frame velocity $\hat{v}^b$ gives the rate of change of position r^n in the inertial frame.

6.5.5 Angular variables

The angular state variables ($\hat{\omega}$, $\hat{\Phi}$) are governed by the moments acting on the body. The moments due to thrust acting on the propeller arm for each propeller are $\hat{M}^b_T = \sum_i^p r_i \times T_i$. The yaw moments about b_3 due to the rotation of the motor are given by $\hat{M}^b_\tau = \sum_i^p \tau_i$. The total moments about the body are $\hat{M}^b = \hat{M}^b_T + \hat{M}^b_\tau$. Solving equation 6.7 for $\hat{\omega}$, and substituting these moments, yields the rate of change of angular rate $\dot{\hat{\omega}}$ in the body frame.

The rate of change of orientation $\dot{\hat{\Phi}} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]$ is related to the angular rate $\hat{\omega}$. The angular rate is the instantaneous rotation rate of the body axes. Whereas each angle of orientation is defined in its own reference frame during a sequence of ordered rotations from the inertial axes to the body frame (yaw, pitch, roll). Solving equation 6.8 gives the rate of change of orientation.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R(\phi) \cdot R(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{6.8}$$

The rates of changes of the 12 (linear and angular) state variables can be integrated to track the state of the vehicle.

6.6 Control

A cascaded PID controller, as depicted in figure 6.5, is implemented for position and attitude tracking. A supervisory position PID controller tracks measured lateral velocities and outputs the required pitch and roll needed to reach a specified way-point. A lower-level attitude controller then tracks the pitch, roll, and yaw velocities and outputs the required torques. In parallel, a PID controller tracks vertical velocity and outputs the thrust needed. The eventual output of the cascaded PID setup is the prescribed thrust, and the roll, pitch, and yaw torques. These prescribed dynamics are then allocated via control mixing to the motors.

The attitude controllers run at a higher frequency than the position controller. This is so that the lower level attitude controller is able to finish tracking its orientation setpoint, and then achieve a change in position which will be tracked by the higher level position controller.

6.6.1 Optimal PID control parameters

Each vehicle configuration has its own optimal controller. A hyperparameter search (Akiba et al., 2019) is conducted over the space of PID parameters to obtain the best parameter set. The objective is the last position vector relative to the origin, which, without any loss in generality, is assumed to be the position reference.

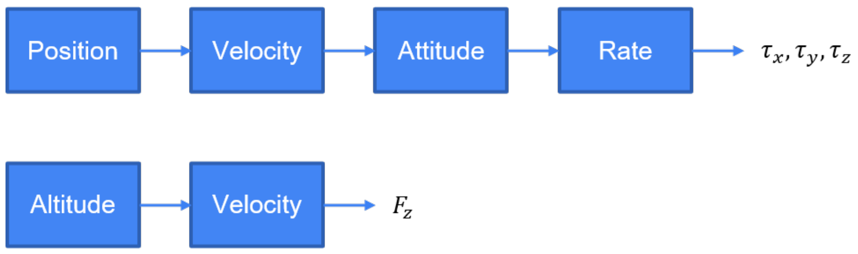


Figure 6.5: Architecture of the cascaded PID controller.

6.6.2 Control Mixing

Control mixing is the procedure of relating the net dynamics (forces and moments) about the vehicle body D, to the rotational speeds of the propellers $\hat{\Omega}$. Due to geometry, the lateral forces on the body from the propeller are zero, thus $F_{b_2} = F_{b_3} = 0$. Moments about b_1 and b_2 are determined by the moment arm r_{arm} and thrust $r_{arm} \times T$. Yaw moments about b_3 are determined by the torque equation 6.2. Putting these relationships together yields the control allocation matrix, and a system of equations linear in Ω^2. Thus, the prescribed dynamics D from the controller can be converted to the prescribed propeller speeds Ω for each motor:

$$\begin{bmatrix} F_{b_1} \\ F_{b_2} \\ F_{b_3} \\ M_{b_1} \\ M_{b_2} \\ M_{b_3} \end{bmatrix} = \underset{6\times 1}{D} = \underset{6\times p}{A} \cdot \underset{p\times 1}{\hat{\Omega}^2}$$

$$\hat{\Omega} = \sqrt{A^{-1} \cdot D} \tag{6.9}$$

6.7 Faults and disturbances

Faults are principally modeled as brush-less direct current (BLDC) motor failures in the system. Motors are controlled via a voltage signal, which modulates propeller speed. The failures lead to modification of motor dynamics. Such changes are modeled by changes in motor parameters like the resistance R_{BLDC} and the electromotive force constant k_EMF. Faults can be abrupt or incipient. Abrupt faults may occur by a step change in motor parameters during the simulation loop. Incipient faults are modeled by using a time dependent function to modify motor parameters. Since the simulation progresses by manually looping over each time step, both classes of faults can be introduced in the loop, as shown in figure 6.4.

Environmental disturbances are modeled as wind. Wind is represented as a vector field of forces which acts on the vehicle during flight. However, the architecture of this simulation is agnostic to the nature of the forces. Any category of external forces, once represented as net forces and torques on the center of mass, can be injected into the logic flow of the simulation (figure 6.1).

6.8 Experiments with nominal PID control

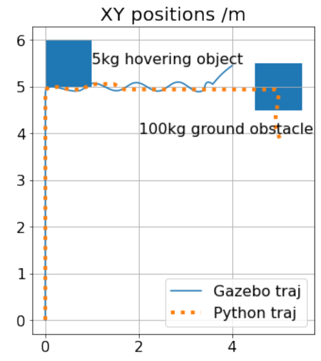
Several trials were carried out to demonstrate the feasibility and applicability of the test-bed. To start, a Tarot T18 model was simulated using the python test-bed. Trials were run over a preset trajectory with different environmental disturbances and internal faults. For each trial, state and diagnostic measurements were collected and plotted. Figure 6.8 shows the results. The same trajectory under nominal conditions is compared under wind conditions and a motor failure. Wind was modeled as a vector field in the x, y-plane with a peak magnitude and period, $w = 2, t_w = 4$. The force field was added to the net forces acting on the vehicle due to thrust, before being sent to the back-end for simulation. Motor faults were modeled as a saturation of motor speed to an upper limit Ω_{max} (equation 6.10).

$$\begin{aligned} \hat{F}_{air} &= [w \cdot \sin\left(\frac{t}{t_w} \cdot 2\pi\right), 0, 0] \\ \Omega &= \min(\Omega, \Omega_{max}) \end{aligned} \tag{6.10}$$

Figures 6.8a and 6.8b show a stable trajectory following vehicle under nominal and slightly windy conditions. Of note is how in the latter case, the velocities experienced by the vehicle are higher as it tries to counteract the wind disturbances. The trajectory is segmented into waypoints which are provided as the vehicle approaches the prior waypoint. A higher segmentation causes the references to change, thus causing the allocated dynamics to change fast as well.

Allocation errors are the difference between controller prescribed dynamics, D, and the net forces and torques actually acting on the vehicle. Allocation errors are close to zero in all cases except for an abrupt motor fault. Under high wind conditions (6.8c), the vehicle is able to adequately allocate control, but the controller itself fails to adequately prescribe dynamics. The thrust and torque set-points from the cascaded PID controller are sub-optimal, but they are satisfied by the vehicle regardless. Conversely, under an abrupt motor fault (figure 6.8d), the vehicle is unable to meet the thrust and torque set-points allocated by the controller.

To exhibit the utility of physics engine back-end, a trial was carried out with obstacles. Gazebo has collision physics available out of the box, the results were measured by the framework front-end (Figure 6.6). The measurements tracked each other until the moment of collision. Due to no networking overhead and



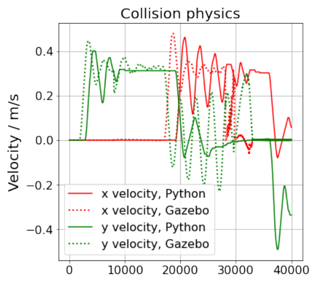


Figure 6.6: Trajectory and velocity plots of python- and Gazebo- back-ends. Collision physics in gazebo allow for simulation of complex scenarios involving environmental disturbances. However, both back ends yield similar simulation results under identical conditions.

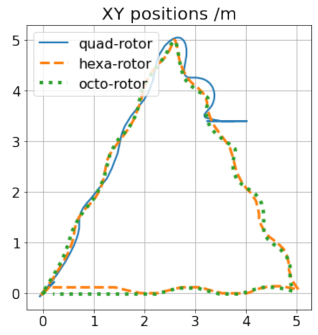
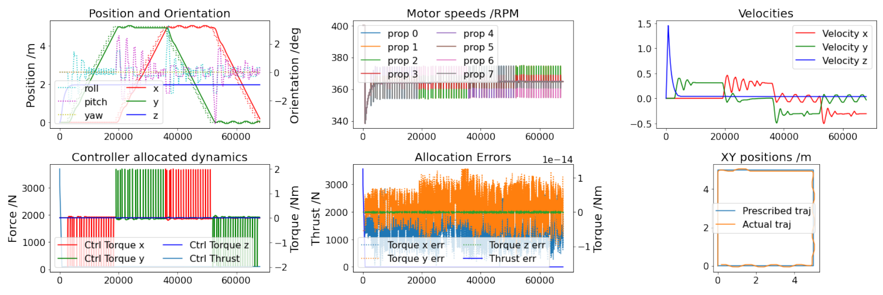


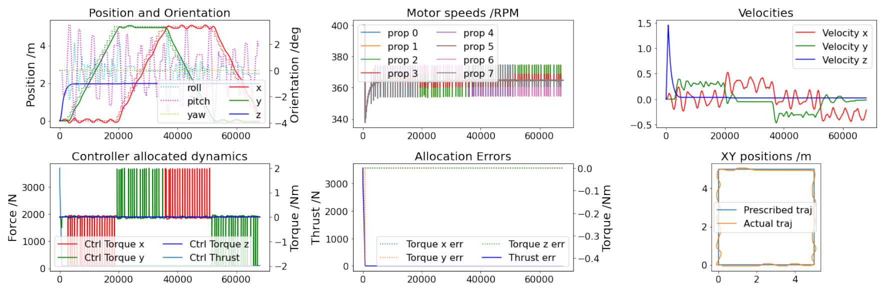
Figure 6.7: Python-only simulation of different UAV configurations under constant wind force, and identical PID parameters.

simplified physics, the python back-end runs faster but it is only applicable for a small number of scenarios. This presents a research use case for quickly prototyping a base-line data-driven controller, which can then be used as a source for transferring to more rigorous control tasks requiring a higher level of fidelity.

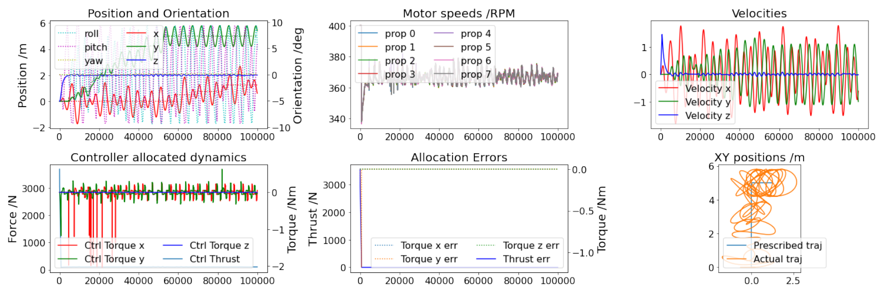
Finally, to demonstrate the ability of the software to simulate different vehicle types by using the same code-base, a trial using the python back-end was carried out with quad-, hexa-, and octo-rotors. The vehicles had different physical characteristics but were actuated by the same PID controller under identical wind disturbances. Figure 6.7 illustrates vehicle behavior under environmental disturbances. Unsurprisingly, a quad-rotor is the first to fail at the trajectory task. The four rotors provide a lesser ability to counteract wind forces during control allocation. For example, the hexa-rotor had 50% more mass and total thrust, and a 100% bigger arm span. This meant it was more effectively able to control its attitude to direct thrust.



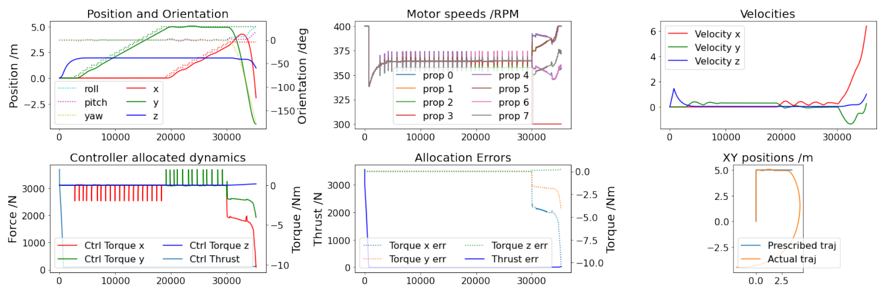
(a) Nominal conditions



(b) Stable flight in low wind conditions, but with some deviations.



(c) Unstable flight in high wind conditions.



(d) Unstable flight due to motor failure at 30s.

Figure 6.8: Simulation of an octo-rotor Tarot T-18 testbed under various conditions. For each condition, a set of system and component variables are plotted. Additionally, the control allocated propeller speeds, and allocation errors are also displayed. The object-oriented code organization allows for quick modification of the environmental conditions. All time (x) axes are in milliseconds.

CHAPTER 7

A Reinforcement Learning Approach for Adaptive Supervisory Control of UAVs Under Disturbances

In this work, we present an approach to supervisory reinforcement learning control for unmanned aerial vehicles (UAVs). UAVs are dynamic systems where control decisions have to be made in the order of milliseconds. We formulate a supervisory control architecture that interleaves with extant embedded control and demonstrates robustness to environmental disturbances in the form of adverse wind conditions. We run case studies with a Tarot T-18 Octorotor to demonstrate the effectiveness of our approach and compare it against a classic cascade control architecture used in most vehicles. While the results show the performance difference is marginal for nominal operations, substantial performance improvement is obtained with the supervisory RL approach under unseen wind conditions.

7.1 Introduction

Advances in technology have increased the feasibility of deploying UAVs across many different applications that include transportation, surveillance, nature research, extreme sports, professional photography, and surveying (Cohn et al., 2017; IV et al., 2006; Nelson et al., 2019). With the increasingly important role of UAVs in automation and human-adjacent applications, the cost of failure is commensurately higher too. Yet, UAVs are being operated in increasingly stressful and adverse conditions (Dhulipalla et al., 2022; Grishin et al., 2020; Ahmed et al., 2023). This has increased the need for flight control that is robust to environmental and other disturbances.

In our review of related work in section 7.2, we note that there is a a disconnect between theoretical research with one-off applications, proprietary offerings, and pragmatic open-source robust controllers that are easy to adopt for civilian use. Ebeid, et al. Ebeid et al. (2017) have reviewed popular open-source flight control software and determined that the preeminent actors in the market are Ardupilot and PX4. Both software applications include embedded low-level controllers running onboard the vehicle. However, as we demonstrate later in this paper, such controllers are not sufficiently robust to environmental disturbances, such as crosswind speeds with drag forces that exceed $5N$. Our solution to this problem is to introduce data-driven reinforcement learning-based supervisory control to make the overall control robust to environmental disturbances, such as crosswinds. However, the computational complexity of deep RL networks grows quadratically as the number of parameters in the deep learning network and the number of data points required to learn the RL-based supervisory controller Bianco et al. (2018). The flight computer for UAVs may not have the ability to consistently run control inference at the desired frequency due to limited processing and memory capacity. Thus, any data-intensive machine learning approach must account for the constraints of the infrastructure it is running on.

In this paper, we propose a supervisory reinforcement learning (RL) approach, which develops a path modification framework that is designed to operate in conjunction with the standard autopilot control software to accommodate wind disturbances that may occur during flight. The rest of this paper is structured as follows. Section 7.2 discusses related work in the field of UAV and adaptive control. The reinforcement learning problem is introduced in Section 7.3. Section 7.4 documents features of the UAV control logic, and our problem formulation for adaptive supervisory RL control. Section 7.5 describes the proposed supervisory control solution, along with the Ardupilot control logic that we have not encountered in peer-reviewed literature to date. Finally, we study the importance of hyperparameter optimization in developing the proposed supervisory RL approach and empirically demonstrate its effectiveness for known and unknown adverse wind conditions.

7.2 Related work

UAVs are dynamic systems that operate on small time constants (order of $10 - 1000$ Hz) and are susceptible to internal and external disturbances during flight. For example, moderate environmental changes and system faults (e.g., loss of a motor) can affect system behavior and destabilize UAV flight. Therefore, it is imperative to develop controllers for UAVs that can adapt to disturbances and faults. Aastrom and Wittenmark Åström and Wittenmark (2013) define an adaptive controller as one "with adjustable parameters" that adapts to dynamic changes in a process. An approach to adaptive control uses a nested loop: the *inner loop* actuates the system, and the *outer loop* manipulates the operating parameters of the inner loop. Methods such as adaptive model-predictive control (MPC), periodically estimate the process parameters (Adetola et al., 2009). Other approaches, like linear quadratic regulators (LQR) solve a sequence of quadratic equations with a single positive-definite solution for controllable systems, to obtain an optimal action based on the linearization of the system at different operating points (Bemporad et al., 2002). Prior work on adaptive control in UAVs adopt classical and data-driven approaches.

7.2.1 Classical adaptive control

Adaptive PID control (Anderson et al., 1988; Tjokro and Shah, 1985) tunes the controller gains based on measured variables that characterize the system behavior. This makes it easier for the controller to track reference signals. However, this comes at the cost of adaptivity because the disturbances that occur in the system must be known and accounted for ahead of time. Work has been done to make control adaptive by tuning PID gains when a fault is detected (Sadeghzadeh et al., 2012). Similarly, Pounds, et al (Pounds et al., 2012) analyzed the effects of abrupt changes in a UAV's payload and the bounds within which the PID control can adapt. In case of multiple actuator failures, cascaded PID controllers on an octo-rotor can be used to reallocate control to redundant rotors (Marks et al., 2012). There is additional literature on quadrotor autonomy using linear and non-linear controllers (Zulu et al., 2014).

MPC uses a dynamic model of the system to detect faults and disturbances and provide adaptive control (Yu et al., 2015). Residual-based fault detection has been used for fault-tolerant control of a co-axial octocopter (Saied et al., 2015). The built-in co-axial motor redundancy can be exploited to accommodate faults by increasing motor speeds. More recent research has involved adaptive MPC to estimate process model parameters under uncertainty for fast dynamics (Adetola et al., 2009; Pereida and Schoellig, 2018; Chowdhary et al., 2013).

Other researchers have demonstrated the utility of sliding mode control for fault tolerance in rotorcraft. Razmi and Afshinfar use sliding mode quad-rotor attitude control with a neural network to learn the sliding mode parameters is demonstrated (Razmi and Afshinfar, 2019). Hamadi, et al (Hamadi et al., 2020) use a self-tuning sliding mode controller to achieve fault tolerance after fault detection.

Some recent work incorporates adaptation built into popular open-source flight control software for fixed-wing vehicles. Baldi, et al (Baldi et al., 2022) use model-free adaptive control for fixed-wind UAV being flown with Ardupilot in wind conditions. They linearize the system dynamics and use a single-step look-ahead cost to adapt the PID controller gains. Similarly, sliding mode adaptive PID gain control has been incorporated for Ardupilot logic for a fixed-wing UAV (Li et al., 2022a).

7.2.2 Adaptive machine learning-based control

Generally, fault-tolerant control schemes using classic control rely on fault detection and isolation as an initial step to adapting the control function to accommodate fault(s) (Chamseddine et al., 2015; Park et al., 2013). Recently, data-driven and RL approaches have been developed for fault-tolerant control leveraging their inherent adaptive capability while attempting to mitigate their sample inefficiency. Usually, they have been employed in conjunction with classic control schemes.

Adaptive classic control literature parallels meta-reinforcement learning in the field of machine learning (Schweighofer and Doya, 2003; Santoro et al., 2016). The inner loop consists of a *base learner*, which optimizes a single optimal control problem. The outer loop consists of a *meta learner*, which updates the parameters of the inner-loop learner so that it generalizes across varying tasks. Recent work by Richards, et al Richards et al. (2021) uses *offline* meta-learning to pre-train an adaptive controller using data-driven simulations. The controller can actuate a drone to follow trajectories under wind disturbances. Razmi and Afshinfar Razmi and Afshinfar (2019) use a neural network to tune the parameters of a sliding mode controller for a UAV. The neural network helps address the drawbacks of sliding mode control, such as susceptibility to chattering and measurement noise.

RL approaches are inherently adaptive, as they learn to control by interacting with the environment. However, the exploration period to allow adaptation may be unsafe and cause failure states. A solution is to use inverse dynamics with General Policy Iteration to learn value functions and then take greedy actions (Li and Xu, 2020). The sample inefficiency and stochastic exploration of RL are mitigated by constraining the state space to follow only certain trajectories during exploration, which are generated from knowledge of system dynamics.

Similarly, RL controllers have been developed that are robust to actuator and sensor faults (cyber attacks) (Fei et al., 2020). The RL controller runs concurrently with a cascaded PID controller and adds control signals to the PID controller's outputs. The RL controller was trained in simulation on faults and compensated for sub-optimal PID actions when faults occurred. This approach was robust and did not need separate fault detection and isolation.

On a similar note, control logic may be abstracted away from systemic changes where possible, to make control adaptation faster (Pi et al., 2021). A system-agnostic UAV control scheme for quad- and hexacopter platforms using RL has been developed. The controller outputs the needed translational and rotational accelerations, which are then allocated to the rotors via a control allocation method that corresponds to the individual vehicle's geometry. This approach demonstrated the utility of data-driven approaches in abstracting knowledge between tasks for fast adaptation when needed.

Another approach combines neural networks with a physics-based model (Shi et al., 2019). The physics-based model learns the lower-order dynamics, whereas the neural network models higher-order effects like noise and the effects of faults on the nominal dynamics. This provides the basis for more accurate model-based control.

In summary, data-driven approaches have been used to complement classic control techniques. The ability to learn from experience and the expressive power of neural networks to represent complex control laws allows for satisfactory generalization. However, in many cases, machine learning approaches to adaptive control may not be practical in real-world applications due to computational and execution time constraints as well as mismatched input/outputs between the proposed solution and extant flight control software. In the work that follows, we develop an RL control scheme that is robust to wind disturbances and can operate in a supervisory mode in conjunction with research flight software.

7.3 Reinforcement learning (RL)

This work discusses the RL-based control of systems modeled as Markov Decision Processes (MDPs). We define an MDP control task characterized by the process dynamics $P : X \times U \to X$, which maps the current state of a system, x_i to its next state x_{i+1}, given an input u_{i+1}. MDPs adhere to the Markov Property. That is, given a state and an action, the state transition probability is conditionally independent of prior states and actions. Or, each state encodes sufficient information to predict the next state, given an action. Each state transition incurs a reward that is based on the state and the action taken to reach that state $r : X \times U \to \mathbb{R}$. An episode is a sequence of interactions (i.e., inputs) that takes the system from a current state to a terminal goal state. The objective for the control task is to derive a policy $\pi : X \to U$, such that each action picked maximizes the expected discounted future returns $\mathbb{E}[G(x_i)]$, where $G(x_i) = \sum_{j=i} \gamma^{j-i} r(x_i, \pi(x_i))$. The discount factor, $\gamma \in (0,1]$ prioritizes the immediacy of feedback. Expected discounted returns in a given state x under an optimal policy are known as its value, $V(x) = \max_u \mathbb{E}[G(x)]$. In these situations, the control problem can be framed as $\pi_t(x) \leftarrow \arg\max_u V(x)$.

7.4 Adaptive supervisory control of UAVs

This section discusses the specific research problem addressed in this work. First, we discuss the control logic of a UAV implemented in the form of the popular Ardupilot software. Ardupilot uses a cascaded PID architecture for converting waypoint references to control signals for the motors. Secondly, the broader problem of navigation is described in real-world conditions.

7.4.1 UAV control with PID

The standard control logic flow of UAVs is depicted in Figure 7.1. In this standard end-to-end approach, the position reference is converted into propeller speed signals.

The first step is PID control (see Figure 7.2) converts position references into the desired dynamics (forces and torques, $F_z, \tau_x, \tau_y, \tau_z$) to achieve that reference position and yaw. The underlying control for position tracking follows a cascaded PID architecture. The control output u aims to minimize the input error $e = \texttt{reference} - \texttt{measurement}$ with respect to the reference and measured states:

$$u_{PID} = k_p e + k_d \nabla_t e + k_i \int_0^t e \partial t \tag{7.1}$$

This PID setup simultaneously tracks lateral and vertical motion. For lateral motion along the x, y axes, the position error is converted into a velocity reference. The velocity controller outputs the required roll and

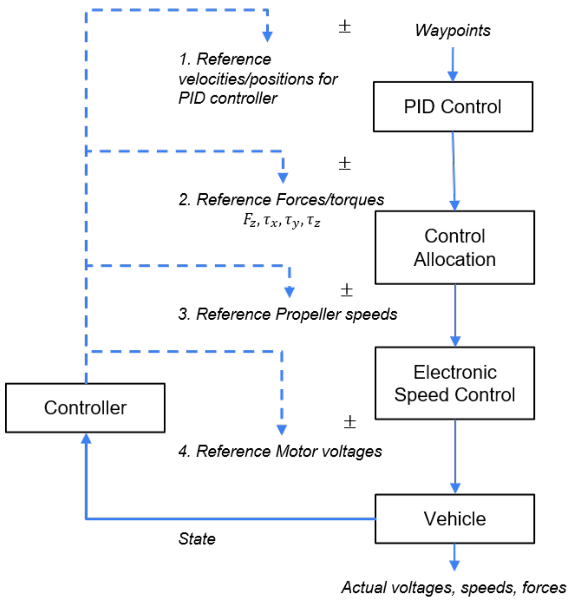

±
Waypoints
1. Reference velocities/positions for PID controller
PID Control
±
2. Reference Forces/torques
$F_z, \tau_x, \tau_y, \tau_z$
Control Allocation
±
3. Reference Propeller speeds
Electronic Speed Control
Controller
±
4. Reference Motor voltages
Vehicle
State
Actual voltages, speeds, forces

Figure 7.1: The control logic flow for a UAV.

pitch angles, which are then converted to the corresponding angular rate. Finally, the rate controller converts the prescribed angular rate into the torques needed from the motors. The cascaded PID controllers operate at different frequencies. For our experiments, the attitude and rate controllers operate at 100Hz to keep up with higher-level position and velocity controllers that operate at a slower rate of 10Hz. In other words, the attitude control is able to catch up with its reference and reduce error before the reference is changed by position control. This is typical of the inner loop of cascaded control architectures.

However, the canonical form of the PID controller in equation 7.1 is susceptible to integral wind-up, noisy inputs, and large actions that are unsafe. The Ardupilot project for autopilot navigation has developed mitigation logic to address these conditions for lateral position control.

Square-root scaling the proportional error term k_p for lateral position and attitude control avoids overshoot in reacting to large initial errors when approaching the reference point. Square-root scaling is a function of the maximum allowed acceleration of e, i.e., $\ddot{e} = \partial^2 e / \partial t^2$ and the time interval of control action Δt. Equation 7.2 shows that if the error exceeds a threshold proportional to $\ddot{e}/k_p^2$, the proportional response is scaled with respect to the square root of e.

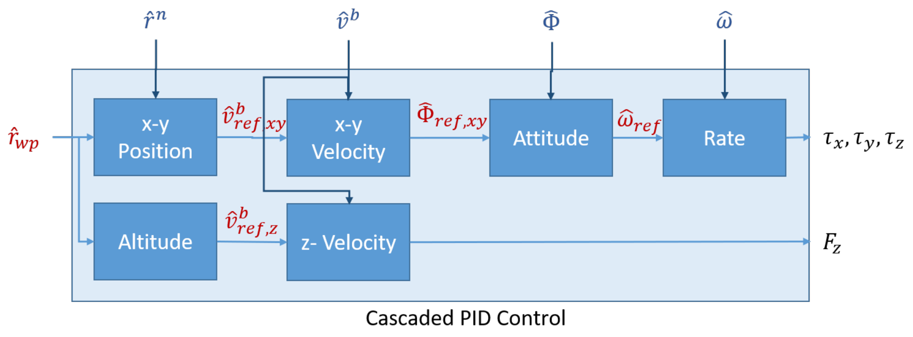


Figure 7.2: The structure of a cascaded PID controller. Position error is eventually converted to desired thrust and torques.

$$u = \texttt{clip}\left(\begin{cases} k_p = 0 & \begin{cases} e > 0 & \sqrt{2\ddot{e}e} \\ e < 0 & -\sqrt{2\ddot{e}(-\ddot{e})} \end{cases} \\ k_p > 0 & \begin{cases} e > \frac{\ddot{e}^2}{k_p^2} & \sqrt{2\ddot{e}(e - \frac{\ddot{e}^2}{2k_p^2})} \\ e < -\frac{\ddot{e}^2}{k_p^2} & -\sqrt{2\ddot{e}(-2 - \frac{\ddot{e}^2}{2k_p^2})} \\ e = 0 & k_p e \end{cases} \end{cases}, -\frac{|e|}{\Delta t}, \frac{|e|}{\Delta t}\right) + k_d \nabla_t e + k_i \int_0^t e \partial t \tag{7.2}$$

When *leashing* is applied to the lateral position controller, it prevents the error input to the PID from becoming too large. As equation 7.3 shows, the position error is clipped to a maximum value given the current kinematics of the system.

$$\texttt{leash} = \left| \frac{|\ddot{e}|}{2k_p^2} + \frac{|\dot{e}|^2}{2|\ddot{e}|} \right| \tag{7.3}$$

Thus a leash can be used to threshold the maximum error fed to the square-root scaling function, which in turn modulates the proportional response of the PID controller.

Once the cascaded PID setup outputs desired dynamics, the control allocation step converts them into desired propeller speeds. The speeds are then converted to voltage signals for input to the motors by the electronic speed controllers (ESCs), and this actuates the vehicle.

Such a control approach presents multiple interaction points for interleaving supervisory control using a reinforcement learning controller. For example, as shown in Figure 7.1, a supervisory controller may modify the trajectory waypoints being fed to the cascaded PID controller. Alternately, the prescribed forces and torques may be modified to optimize for some navigation objectives before they are allocated as speeds. Similarly, the voltage signals to the speed controllers may be changed. In summary, theoretical and practical factors limit the choices of where adaptive control schemes may be applied.

7.4.2 Trajectory control of UAVs

Our overall objective in this work is optimal trajectory control of the UAV in real-world operations, where flights may be disrupted by disturbances, such as wind. In this paper, we modify the control architecture to ensure that the originally intended trajectory is followed as closely as possible in spite of these disturbances.

Reactive controllers, such as PID, can use tailored parameters to adjust to different operating conditions. However, they are designed to follow the original reference trajectory, specified for flights with no environmental disturbances. As a result, a PID position controller in a headwind may fall short of the originally specified reference position unless the proportional and integral terms are amplified. However, such amplifications also cause strong reactions to errors, which can cause overshoot. While a PID controller may be made robust to noise through the derivative term k_d, it ultimately is insufficient for trajectory control which is solved through reference modification. We overcome this problem by designing supervisory controllers that are proactive and can alter the references of PID controllers to accommodate disturbances.

Proactive control methods, such as MPC, can use an accurate model of the environment to replan ahead. However, system identification and the fine-tuning of parameters increase the computational load of the control algorithm. In addition, solving the limited horizon optimization problem for each decision can become intractable for systems with small time constants, because they require very fast reactions.

RL can also be used for proactive control. Developing an RL controller is a two-step process. In the learning step, RL solves an optimization problem to learn the control policy by taking exploratory actions and observing feedback. The control policy, once learned, can generate the best control actions to take for a given state of the system. This, like MPC, addresses the problem of reactivity in PID control. It also ameliorates the recurrent computational costs associated with the MPC optimization problem. The learning process, however, suffers from the challenges of safety because of the stochastic actions used for learning, the large amounts of data that may be needed to learn a good policy, and the computational cost of hyperparameter optimization. Therefore, an RL controller that does not need to learn often under different conditions presents one solution. Such a controller will be robust to disturbances. A robust controller will fulfill the flight objective under different disturbance conditions. In the following section, we present the architecture and the design choices for developing a robust supervisory controller when the UAV system operates under disturbances, such as fixed wind conditions.

7.5 Supervisory control solution formulation

We develop a waypoint-to-waypoint adaptive supervisory strategy for training a robust controller, π_{RL}, and using that controller for trajectory control scenarios. Figure 7.3 shows how the proposed supervisory setup interacts with the vehicle's control systems. There are two supervisory blocks: *Trajectory breakdown* and *RL control*. The former decomposes the desired flight path into intermediate waypoints r_{wp}. It modifies the state observed observed by the RL controller such that the next waypoint r_{wp} is at relative origin. The RL controller modifies the setpoint references of the cascaded PID controller, without modifying its internal logic. It takes the (modified) state of the vehicle as input, and outputs the change in reference for the PID controller, which is implicitly set to relative origin. The new references cascade through the PID system to the control allocation step, which eventually actuates the vehicle.

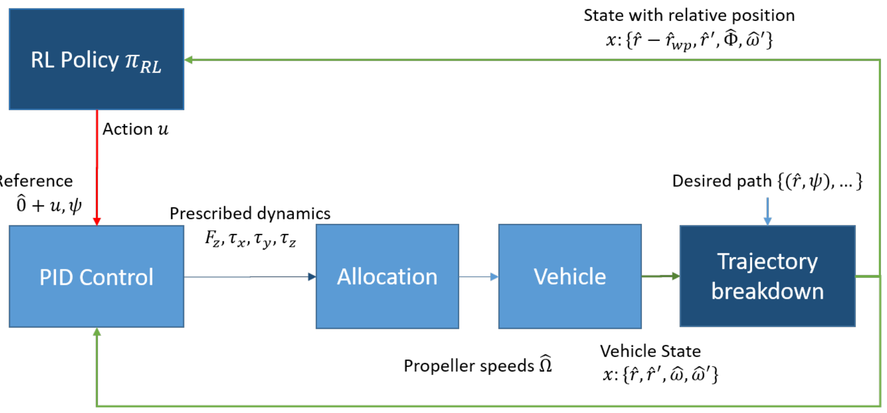

State with relative position
$x: \{\hat{r} - \hat{r}_{wp}, \hat{r}', \hat{\Phi}, \hat{\omega}'\}$
RL Policy π_{RL}
Action u
Reference
$\hat{0} + u, \psi$
Desired path $\{(\hat{r}, \psi), \ldots\}$
Prescribed dynamics
$F_z, \tau_x, \tau_y, \tau_z$
PID Control
Allocation
Vehicle
Trajectory breakdown
Propeller speeds $\hat{\Omega}$
Vehicle State
$x: \{\hat{r}, \hat{r}', \hat{\omega}, \hat{\omega}'\}$

Figure 7.3: The UAV Trajectory control interface via waypoint control. The reference positions being tracked by the PID controller are modified.

This supervision has the practical advantage of being minimally invasive and easy to implement with existing flight control software like Ardupilot. Secondly, the RL controller is based on a neural network, which is a function approximator for the control policy. It is computationally more demanding and may not be able to run at higher frequencies of the embedded PID control logic. Having RL in a supervisory capacity, which modifies references of the low-level control loop at a lower frequency, prevents it from being a performance bottleneck in time-constrained applications, such as this. Finally, by breaking down a longer desired path into intermediate waypoints makes RL more tractable. Instead of a longer path following task with a larger state space, each waypoint represents a granular navigation-to-origin problem. That is, if the RL controller can learn to navigate to origin from an arbitrary position within a bounding box, it can fly a longer desired path by sequentially solving smaller navigation tasks.

Our proposed approach divides the RL controller's development into a *training* and an *operating* phase. The training phase happens inside a virtual 3D *bounding box*. The RL controller learns to fly the vehicle from an initial state to the origin, under a constant disturbance. An initial state imparts a random displacement and velocity to the vehicle. During operation, the desired trajectory is broken down into linear segments marked by intermediate waypoints $\hat{r}_{wp}$ which are no farther apart than the length of the bounding box during training. The RL controller views the vehicle's state with position *relative* to the next waypoint. Then the controller navigates to the relative origin, as in the training phase, while ultimately flying to $\hat{r}_{wp}$.

The UAV's control and dynamics processes, described previously, are modeled as a MDP that is controlled by RL. The actions to the MDP are the changes in position references being tracked by the PID. The state observation is the kinematic state of the vehicle. The reward being optimized for by RL incentivizes short, straight flight segments to reference positions along the pre-specified trajectory (see Figure 7.4). Section 7.5.1, section 7.5.2, and section 7.5.3 explain the choice of these parameters in greater detail. Then Section 7.5.5 and section 7.5.6 explain our proposed algorithms for training and operating a robust controller.

In the following subsections, we define the state and action spaces of the control task, and the reward function being optimized. Finally, the overall experimental setup is discussed. The results of our experimental studies are presented in Section 7.6.

7.5.1 Action space

We model supervisory actions as additive modifications to the waypoint of the navigation problem. The controller's objective is to modify the waypoint from the origin, such that the effects of disturbances, which are also additive (especially in the case of simple wind profiles) are negated. The supervisory controller operates at a lower frequency than the underlying PID control.

Theoretically, the choice of control action u and the frequency of application of control f_{RL} is a function of the computational tractability of the corresponding control optimization problem. Given that the internal PID controller of the UAV has its own transient state, the UAV system dynamics do not satisfy the Markov property (section 7.3). This is because the defined UAV system state is defined only by the kinematic variables of the vehicle's motion (see section 7.5.2). In other words, the system is partially observable. There are two approaches to mitigating partial observability of the state space. In the first approach, we append the variables of the PID controller to the state vector of the vehicle, which is input to the RL agent. In the second approach, we lower the frequency f_{RL} of the interaction of the RL agent with the vehicle. The lower frequency of interactions results in the transient effects of changing the references inside the PID controller to die down.

In this paper, we adopt the second approach. The RL controller takes a supervisory action after $(f_{RL}\delta t)^{-1}$ steps, where δt is the simulation step computed from the time constant of the vehicle dynamics. It allows the RL agent to view the state of the vehicle after the PID controller has actuated the vehicle to follow the reference position. In this way, the RL agent can train on state observations that are correlated with its control outputs.

Practically, the choice of control action is constrained by the problem application. We envision combining the RL-based supervisory controller with Ardupilot-based control operating in the Guided Mode. The Ardupilot-based controller operates on the flight computer. In the Guided Mode, the low-level controller accepts position and velocity references from the ground station and outputs the motor control signals to the electronic speed controllers. Due to the lag associated with communication, and the possibility of connection loss, actions are taken in a supervisory manner wherein a baseline level of control is possible without RL intervention. Having such a supervisory controller makes interoperability with Ardupilot easier, without having to interrupt its internal control logic.

7.5.2 State space

The UAV is modeled with dynamic state variables, which describe its flight (Ahmed et al., 2022). These variables, listed below, describe the kinematic state of the vehicle:

- $\hat{r}^n \in \mathbb{R}^3$ are the navigation coordinates, in the global, inertial navigation reference frame axes n.
- $\hat{v^b} \in \mathbb{R}^3$ is the velocity of the vehicle in the vehicle's local, non-inertial body frame b.
- $\hat{\Phi} = [\phi, \theta, \psi] \in [-\pi, \pi]^3$ is the orientation of the body reference frame b in Euler angles (roll, pitch, yaw) with reference to the inertial reference frame n.
- $\hat{\omega} = [\omega_x, \omega_y, \omega_z] \in \mathbb{R}^3$ is the roll rate of the three body frame axes with reference to the inertial frame.

We assume they also represent the state space for the controller.

For the reasons discussed above, i.e., the PID controller contributing to the state describing the dynamics of the UAV, an action may produce different dynamics. Thus, to satisfy the Markov property, such that the RL controller can evaluate optimal actions, the internal state of the PID should be a part of the state vector input to the controller. However, there are 6 cascaded PID controllers, each with an internal derivative and integral state. This amounts to 12 additional variables, doubling the dimensionality of state space. This makes the search space for an optimal policy exponentially larger.

Furthermore, as explained in our choice of actions, the state of the PID is transient. At constant velocity and infrequent supervisory actions, the rate of change of error will be constant. And clipping the integral error to low magnitudes will prevent the effects of wind-up. Considering this transient nature, and infrequent actions, limiting the state vector to the kinematic variables presents a compromise between accuracy and computational tractability.

7.5.3 Reward

The objective (or, reward) is to reach a waypoint $\hat{r}_{wp} \in \mathbb{R}^3$ in the shortest possible time from the starting position $\hat{r}_0$ without violating velocity constraints. The implication of this is to traverse the shortest distance and at the greatest allowable speed. Therefore, the reward function imposes a constant time penalty for each interaction with the environment. Given an objective waypoint $\hat{r}_{wp}$, the last and current positions $\hat{r}_0, \hat{r}_1$, it rewards motion towards the waypoint and penalizes motion perpendicular to the shortest path. The reward formulation is further elaborated in equation 7.4.

$$r(\hat{r}_i, \hat{r}_{i+1}, \psi_i, \psi_{i+1}) = -\delta t - (|\psi_{i+1}| - |\psi_i|) + |(\hat{r}_{i+1} - \hat{r}_i)| - |(\hat{r}_{i+1} - \hat{r}_i) \times (\hat{r}_{wp} - r_0)| \tag{7.4}$$

This reward formulation is made up of several components. The time penalty δt is a constant penalty for each additional time step before reaching the destination (i.e., the next waypoint). A turn penalty $|\psi_{i+1}| - |\psi_i|$ incentivizes the controller to keep a constant heading by penalizing yaw. For this experiment, position control is only done via roll and pitch changes (ϕ, θ). An "advance" term $|(\hat{r}_{i+1} - \hat{r}_i)|$ rewards motion, towards or away from the target. In our experiments for finding a suitable reward function, we noted that a *signed* advance term yields worse results. This may be because a signed advance term confounds objectives with the time penalty. Advancing *towards* the target necessarily will accumulate a smaller time penalty. However, sometimes, due to an initial velocity away from the destination, the controller will necessarily accrue negative advance as it decelerates. Finally, a "cross" deviation term $|(\hat{r}_{i+1} - \hat{r}_i) \times (\hat{r}_{wp} - r_0)|$ penalizes motion perpendicular to the position vector, the shortest path, to the destination. The ending condition of an episode incurs an additional bonus reward or penalty (equation 7.5).

$$r \leftarrow r + \begin{cases} \texttt{reached} & +\texttt{bounding box} \times 20 \\ \texttt{tipped}(\Phi > 30^\circ) \texttt{ or out-of-bounds} & -\texttt{bounding box} \times 20 \\ \texttt{timed out (t > 20s)} & -|\hat{r}_{i+1} - \hat{r}_{wp}| \times 10 \end{cases} \tag{7.5}$$

7.5.4 Experiment setup

We use a simulation testbed for a Tarot T-18 octo-rotor UAV. For this simulation environment, the maximum roll and pitch angles (ϕ, θ) are restricted to $\pi/12$ radians. Lateral motion is achieved via roll and pitch. Therefore, yaw controls are disabled. The maximum velocity is restricted to $3ms^{-1}$, and the bounding box for the granular navigation problem is $20m$ on all sides. The bounding box dimensions are discussed in greater detail in following subsections. The maximum lateral acceleration the vehicle can achieve is 2.5 ms^{-2}, given the maximum tilt angle and a maximum propeller speed, leading to a net lateral force of $26.65N$, given the UAV mass is $10.66kg$.

Disturbances are modeled as a wind force with a constant heading applied through the duration of a flight. We assume the effects of turbulent wind are minimal, and the duration of the flight is within the time frame of the constant prevailing wind. We conduct experiments with a maximum lateral wind disturbance of $5N$, accounting for 18.7% of the available lateral thrust.

Furthermore, a set of PID-only controller parameters is first obtained for adequate comparison. The parameter space is tabulated in Tables 7.1, 7.2. The relationships between parameters and the performance as measured by robustness are discussed for each set of experiments. Other experimental details are presented in Appendix 7.8.

An episode represents a single, granular, navigation problem inside a 3D bounding box. The vehicle is initialized at a random position and velocity relative to the origin. At each simulation step, the RL controller receives the reward for its state. The episode ends successfully when the vehicle comes within a propeller arm span of the origin. An episode is a failure when it times out, exceeds the bounding box, or tips over. A larger trajectory is simulated by solving a sequence of episodes successfully with waypoints set to relative origin.

The Proximal Policy Optimization (PPO) algorithm is used for RL Schulman et al. (2017). PPO operates over the continuous state and action spaces. It clips the magnitude of iterative updates to avoid drastic changes in the policy. For learning a policy, the state vector is min-max normalized to the range $[-1, 1]$. This is to prevent an asymmetric cost surface for RL optimization. That may cause activation function saturation from state variables that become too large, or gradient updates to policy parameters that overshoot the optimum Sola and Sevilla (1997).

7.5.5 Training phase

The controller is trained by presenting it with "episodes" of navigation problems. For each problem, the vehicle is initialized with a starting position and velocity inside a 3D `bounding box`, along with a wind disturbance force. Since the eventual goal is trajectory control over longer paths, Figure 7.4 illustrates how a longer trajectory may be broken up into a sequence of such episodes. This is done by initializing every subsequent episode with the terminal state of the vehicle from the prior episode and changing its position relative to the next waypoint. We hypothesize that by solving a series of smaller navigation problems, we are able to reduce the problem complexity and also counteract the effects of disturbances over longer trajectories.

During training, the vehicle is exposed to wind disturbances in the two following ways. Later experiments

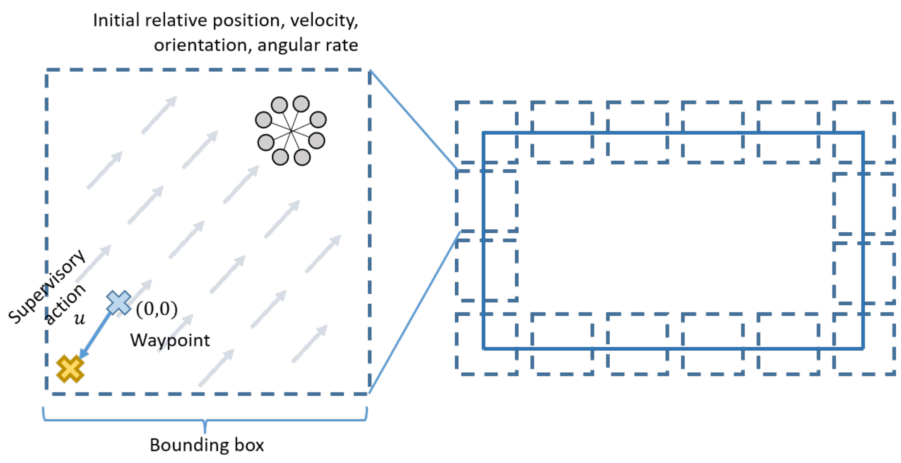

Initial relative position, velocity, orientation, angular rate
Supervisory action u
(0,0)
Waypoint
Bounding box

Figure 7.4: The trajectory modification problem under disturbances such as wind. The goal is to break up a longer trajectory into smaller, granular problems (left), where the supervisory controller changes the relative waypoint to counteract disturbances.

evaluate the payoff between specialization and generalization given the same computational budget.

1. For each episode, the wind disturbance is at a constant heading and magnitude. A controller is optimized for a single wind condition.

2. At the start of each episode, a wind disturbance of constant magnitude from one of four cardinal directions is picked at random. A controller is optimized for random wind directions.

The supervisor modifies the reference position being tracked by the cascaded PID controller, which actuates the vehicle. The amount by which the reference can be modified is limited to the size of the `bounding box`. The `scaling factor` $\in (0,1]$ is a hyperparameter for the controller, which further limits the magnitude of the reference modification. The actual change in reference waypoint is $u \leftarrow$ `scaling factor`$\cdot$`bounding box`$\cdot u$.

The choice of the size of the `bounding box` is a design-time decision for this work. A smaller bounding box, given the flight envelope of the vehicle, namely maximum acceleration and velocity, may not provide sufficient room to navigate against disturbances. For example, when the vehicle is blown off-course by wind. A larger bounding box will make the state space (and policy search space) larger with little marginal benefit. For a large bounding box, a bulk of the flight in a bounding box will be spent at constant velocity from saturated PID control outputs due to a large position error. Therefore, the relationship between supervisory changes to position references and the vehicle's response will have no correlation. Making the bounding box smaller thus increases the efficiency of the RL policy search process.

Algorithm 11 illustrates the steps for training a supervisory controller using reinforcement learning.

Algorithm 11 Training supervisory RL control

```
Require: Wind disturbance force $F_{xy} \in \mathbb{R}^2$
Require: UAV model with state $x$: position $\hat{r}$, velocity $\hat{v^b}$, orientation $\hat{\Phi}$, angular rate $\hat{\omega}$
Require: number of training episodes, bounding box, scaling factor, $f_{RL}$
 1: Initialize supervisory controller $\pi_{RL} : x \to [-1,1]^3$
 2: for episode in episodes do
 3:     Set random UAV position $\hat{r}$ inside bounding box $\in \mathbb{R}^3$
 4:     Set random UAV velocity $\hat{v^b}$
 5:     while $\hat{r} \neq \hat{0}$ do
 6:         if time more than $1/f_{RL}$ since last action then
 7:             Set position reference $u \leftarrow \hat{0} +$ bounding box $\cdot$ scaling factor $\cdot\, \pi_{RL}(x)$
 8:         end if
 9:         Get prescribed dynamics from PID, given reference
10:         Allocate propeller speeds
11:         Apply $F_{xy}$ to UAV model
12:         Update state measurement of UAV $\{\hat{r}, \hat{v^b}, \hat{\Phi}, \hat{\omega}\}$
13:         Observe reward $r$
14:     end while
15:     Collect $x, u, r$ experiences and update $\pi_{RL}$ using PPO algorithm
16: end for
17: Return $\pi_{RL}$
```

7.5.6 Operating phase

During operation, a longer desired path is broken up into intermediate waypoints $\hat{r}_{wp}$, which are a `bounding box` apart. For each step of operation, the state of the vehicle observed by the RL controller is changed, such that the position is relative to the waypoint. Thus, the relative destination is the origin in the controller's frame. This then becomes the same navigation task as during training. The controller sequentially navigates through the waypoints, until it finishes the desired trajectory.Algorithm 12 delineates the proposed approach for the trajectory control problem.

The `bounding box` parameter also has ramifications for safety constraints of operation, in addition to the efficiency of machine learning. By harshly penalizing deviations away from the waypoint that exceed that box, the controller is given a soft-constraint for adhering to a safety corridor in flight.

7.6 Results and Discussion

In this section, the performance of our adaptive supervisory RL control approach is evaluated in terms of robustness to wind disturbances. A hyperparameter search is conducted to find the best-performing parameters for each experiment. The choice of hyperparameters is evaluated by averaging the total reward over 10 episodes. First, we evaluate the baseline case of PID and RL control under nominal conditions. Then, optimal hyperparameters for PID control under disturbances are evaluated. Finally, an RL supervisory controller is learned for different disturbance conditions. We demonstrate how the controller generalizes to novel

Algorithm 12 Supervisory RL control for trajectory following

```
Require: Trajectory of waypoints, bounding box,scaling factor
Require: UAV with state $x$: position $\hat{r}$, velocity $\hat{v^b}$, orientation $\hat{\Phi}$, angular rate $\hat{\omega}$
Require: Supervisory controller $\pi_{RL} : x \rightarrow [-1,1]^3$
 1: Break trajectory into waypoints bounding box apart
 2: for $\hat{r}_{wp}$ in waypoints do
 3:     Set relative position of UAV to $\hat{r}_{wp} - \hat{r}$
 4:     while $\hat{r} \neq \hat{0}$ do
 5:         if time more than $1/f_{RL}$ since last action then
 6:             Set position reference $\leftarrow \hat{0} +$ bounding box $\cdot$ scaling factor $\cdot \pi_{RL}(x)$
 7:         end if
 8:         Get prescribed dynamics from PID, given reference
 9:         Allocate propeller speeds
10:         Update state measurement of UAV $\{\hat{r}, \hat{v^b}, \hat{\Phi}, \hat{\omega}\}$
11:     end while
12: end for
```

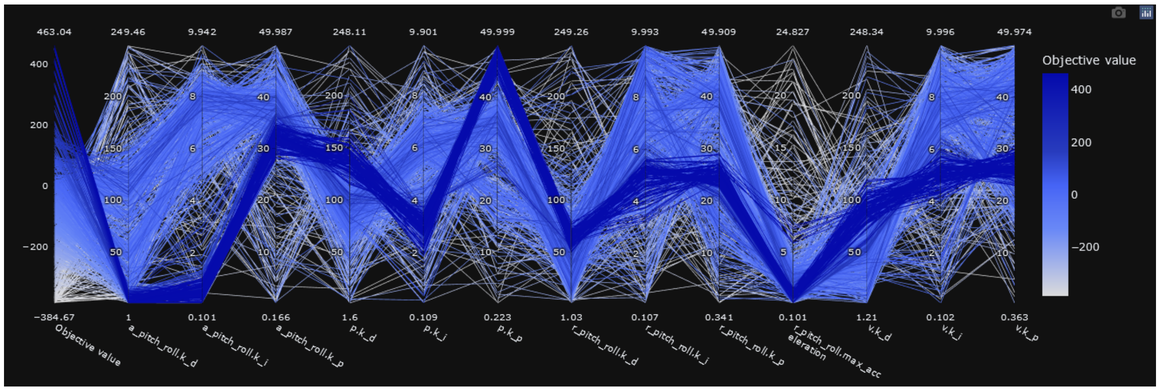
conditions without substantial degradation in performance.

7.6.1 Hyperparameter optimization for nominal control

Figure 7.5a shows the result of 1000 hyperparameter optimization trials. For each trial, the search space is over k_p, k_i, k_d parameters of the cascaded PID controller. Another hyperparameter is the maximum acceleration allowed in the rate controller, max_{acc}. For the position, velocity, and rate controllers, the most important (Hutter et al., 2014) parameter is k_p. For the attitude controller, however, the k_d parameter has a higher importance as the controller damps the proportional response so the vehicle does not tip over the reference attitude angle. For the rate controller, and overall, the most important parameter is the maximum allowed angular acceleration. It governs the maximum prescribed torque output to the control allocation block. A higher maximum acceleration will cause the propellers to suddenly generate a larger torque, causing the vehicle to tip over.

In our experiments for hyperparameter optimization, we noted that re-using optimal PID parameters from the PID-only control case for the RL control case often led to sub-optimal results. By changing the position references marginally, the RL controller modifies the position error, as well as the downstream velocity and angular rate errors to achieve the optimization objective. Therefore, a PID controller with error constants tailored to accommodate the scale of such reference changes is needed. As a result, the hyperparameter search for the RL supervisory case also searched over the space of underlying PID parameters.

Figure 7.5b shows the results of 1000 hyperparameter optimization trials under no wind disturbances for RL control. Like the PID-only case, the most important parameter remains the maximum acceleration limit of the rate controller. k_p is the most important parameter for the position controller. k_d, for the velocity,


(a) Cascaded PID only.

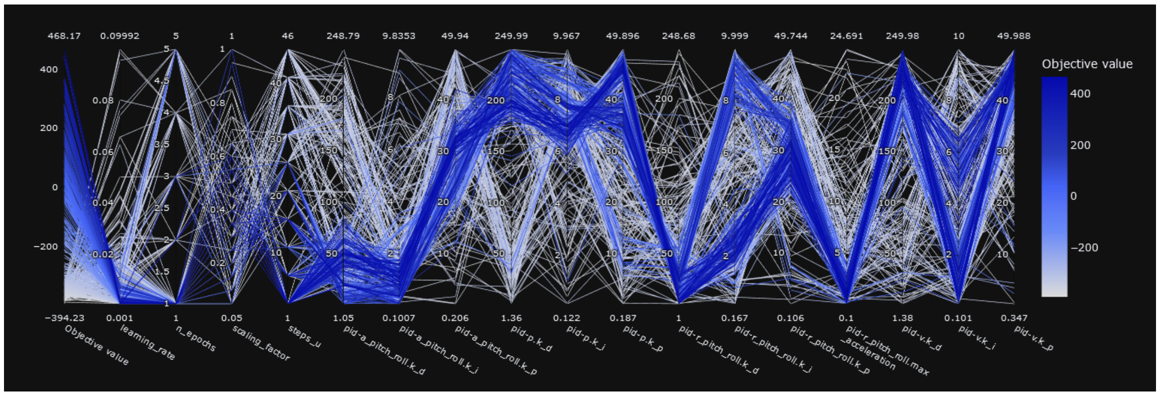

(b) Supervisory RL and cascaded PID.

Figure 7.5: A parallel coordinate plot of controller parameter combinations, shaded by mean total reward over the same 10 episodes in nominal conditions. The darker tract of lines indicates the combinations that performed well on the navigation task.

attitude, and rate controllers. We hypothesize that k_d is more important in the supervised case to damp the changes in the tracking error due to the RL supervisor intermittently changing waypoints.

Figure 7.6 shows the relative parameter importance for the hyperparameter searches across all RL and PID controllers. The most important parameter overall with respect to the scoring metric is the maximum angular acceleration (rate max$_{acc}$), which limits the maximum action output of the Rate controller (τ_x, τ_y, τ_z). A large torque as a result of a sudden change in reference can cause the vehicle to tip over, which incurs a large penalty.

7.6.2 PID control with wind

Next, the derived controllers are tested with wind disturbances. For the duration of the flight, the wind is considered blowing from a constant heading and with a constant magnitude.

A hyperparameter search is run for a PID-only controller for wind disturbances. Figure 7.7a shows the results of 1000 hyperparameter optimization trials for one wind condition. Notably, the total reward

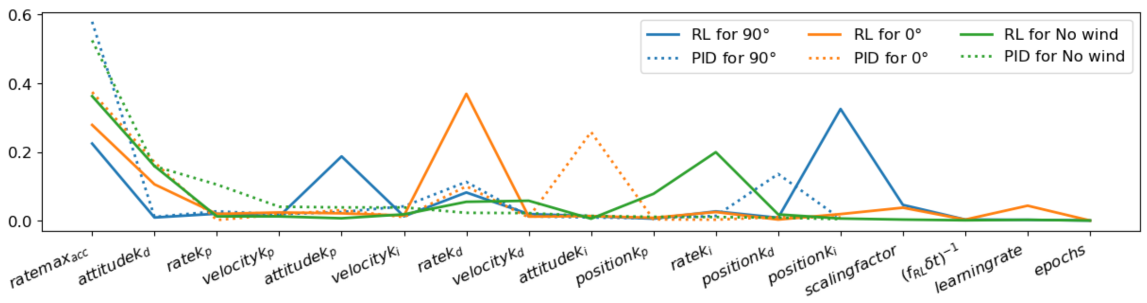


Figure 7.6: hyperparameter importance (Hutter et al., 2014) for RL and PID control with various wind disturbances. The x-axis is ordered in descending order of parameter importance for the PID-only case under no wind.

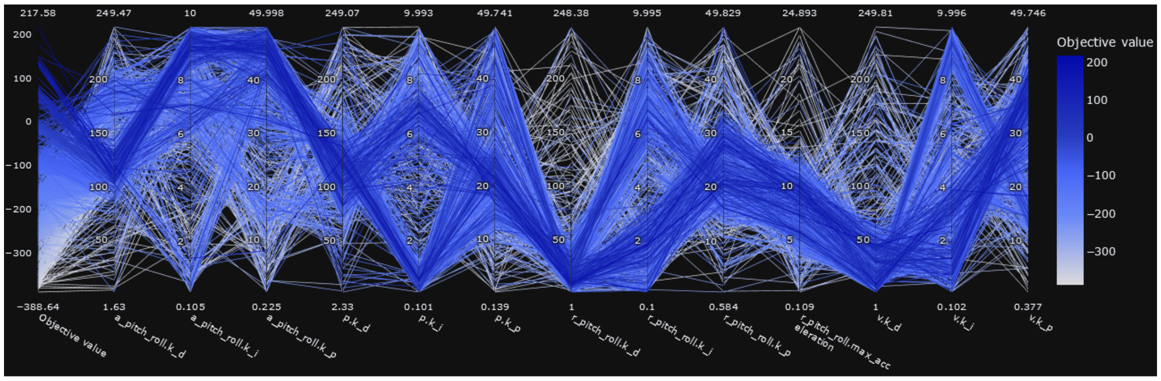
Controller	Params	Nominal	Nominal σ /%	Wind	Wind σ /%	Change /%	Notable
Attitude	k_d	2.182	159.562	129.924	21.353	5853.095	True
	k_i	0.490	34.428	8.026	31.822	1537.930	True
	k_p	33.099	3.528	40.167	30.672	21.353	False
Position	k_d	123.846	6.111	119.274	10.739	-3.692	False
	k_i	3.334	7.591	3.569	93.197	7.039	False
	k_p	46.091	3.602	21.704	24.494	-52.911	True
Rate	k_d	56.596	12.557	15.397	62.921	-72.795	True
	k_i	5.238	8.657	2.649	80.416	-49.431	False
	k_p	27.040	4.462	23.469	19.329	-13.206	False
	max_{acc}	5.376	50.992	8.478	20.002	57.717	True
Velocity	k_d	92.076	6.026	29.712	68.644	-67.731	False
	k_i	5.039	4.172	2.725	43.418	-45.927	True
	k_p	24.393	5.736	32.050	22.363	31.393	True

Table 7.1: Comparison of the top 10 performing parameters for the cascaded PID controller. The "Notable" column is true for percent changes larger than the standard deviations of the parameters. Parameters for yaw control are set to 0. Altitude velocity and altitude rate controllers are kept as P-only controllers with $k_p = 1, 10$ respectively.

over 10 test episodes degrades substantially compared to the nominal case with no wind. The average best performance from disturbances from 0° to 90° is 317 for PID, compared to the nominal case of 463. This is expected since the PID represents reactive control. It can accommodate disturbances, but only to a certain extent without re-tuning all the PID parameters. For the position controller, the most important parameter is k_d. For velocity control, it is k_i. For attitude control, it is k_p. And for rate control, the key parameter is k_d.

7.6.3 Trained adaptive RL supervisory control under wind

In this section, we evaluate the efficacy of the proposed supervisory RL controller with wind disturbances. First, a hyperparameter search consisting of 500 trials is conducted with RL control. Searches are conducted for two prevailing wind disturbances. A parallel plot of hyperparameter combinations is shown in Figure 7.7b. Table 7.2 shows a comparison of the best parameters for RL control for 5N wind coming from 90° to the direction of flight. For the RL controller, Table 7.2 shows that position and velocity k_p decrease with


(a) Cascaded PID only.

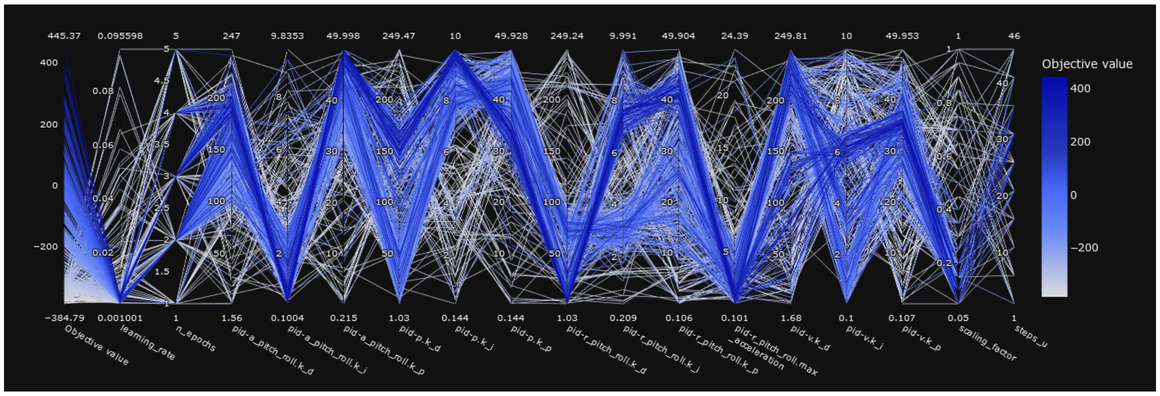

(b) Supervisory RL and cascaded PID.

Figure 7.7: A parallel coordinate plot of controller parameter combinations, shaded by mean total reward over the same 10 episodes in wind conditions. The darker track of lines indicates the combinations that performed well on the navigation task.

wind, but k_i increases. The scaling factor for RL actions decreases by 75%, and the frequency of actions f_{RL} decreases as well. That is, the RL controller is making smaller, less frequent, but ultimately more persistent changes to the waypoint and velocity references. The downstream attitude and rate controllers' higher k_p are quick to act on those changes cascading through the system.

Figure 7.8 shows how the supervisory controller interacts with a navigation problem for two wind conditions. The black points represent the changed waypoints given from the RL supervisor. They are minor changes in position references for the PID controller to follow. Also shown are the trajectories of the nominal PID controller, and the PID controller trained for use with RL but used alone. A nominal PID controller trained on disturbances can complete some trajectories, albeit with overshoots and deviations from the shortest path. When the PID controller trained along with the supervisor is tested alone, it is unable to finish the trajectory.The plots show that a PID controller alone is not sufficient to withstand the simulated disturbance.

Figure 7.9 shows RL and PID controllers, optimized for different disturbances, operating on a small trajectory under a single wind condition. While the RL controller is able to reach all waypoints, the PID

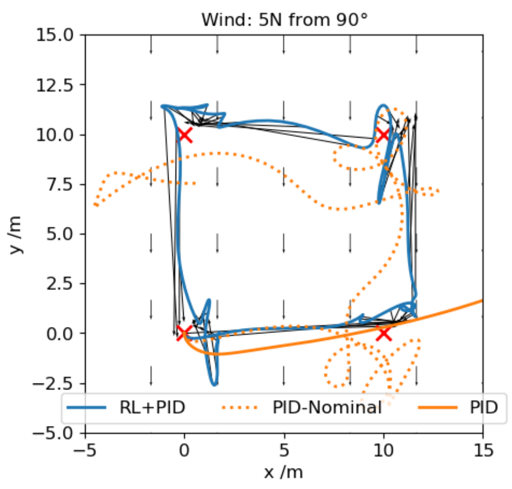
Controller	Params	Nominal	Nominal σ /%	Wind	Wind σ /%	Change /%	Notable
RL	batch size	217.600	9.301	128.000	0.000	-41.176	True
	learning rate	0.001	40.624	0.003	10.301	105.725	True
	epochs	1.200	35.136	3.300	20.453	175.000	True
	steps	15372.800	40.206	825.600	6.005	-94.629	True
	scaling factor	0.485	20.067	0.120	21.517	-75.258	True
	steps u	2.500	189.737	31.000	0.000	1140.000	True
Attitude	k_d	54.174	32.153	200.493	9.926	270.088	True
	k_i	1.023	42.851	0.335	33.189	-67.237	True
	k_p	32.092	13.841	48.807	2.933	52.084	True
Position	k_d	203.020	8.490	156.692	5.118	-22.819	True
	k_i	6.786	6.730	9.742	1.722	43.549	True
	k_p	44.851	7.058	34.399	3.732	-23.303	True
Rate	k_d	6.568	113.713	2.594	66.204	-60.503	False
	k_i	3.209	65.263	7.637	3.711	137.999	True
	k_p	27.769	11.064	41.355	3.028	48.928	True
	max_{acc}	0.973	52.683	0.728	72.387	-25.120	False
Velocity	k_d	227.655	3.126	117.124	65.095	-48.552	False
	k_i	3.404	72.105	6.187	5.269	81.772	True
	k_p	47.343	3.569	36.024	3.124	-23.909	True

Table 7.2: Comparison of the top 10 performing parameters for the RL-supervised cascaded PID controller. The "Notable" column is true for percent changes larger than the standard deviations of the parameters. Parameters for yaw control are set to 0. Altitude velocity and rate controllers are kept as P-only controllers with $k_p = 1, 10$, respectively.

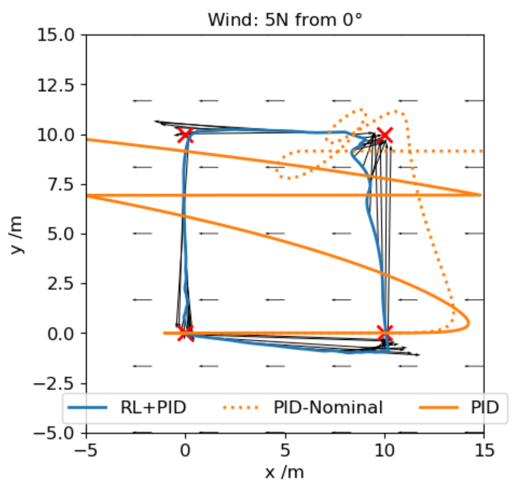
controllers (both nominal and optimized for wind) fail after reaching one or two waypoints.

We generalize this result by evaluating trajectory tracking performance over multiple episodes. For nominal conditions, RL control and PID control produce similar results. However, with wind disturbance, PID-only control degrades, but the RL supervision remains robust. Figure 7.10 shows a density plot of episodic rewards for nominal and wind conditions for both controllers.

We conducted another robustness study by varying the wind force magnitude from four directions (see Figure 7.11 and tables 7.3, 7.4). In this study, controllers optimized for a single wind disturbance during training are compared with controllers optimized for random wind disturbances. For both cases, the computational budget is the same. Performance was measured by total rewards over the square trajectories shown previously. For both training cases, RL controllers could maintain more consistent performance as the magnitude of the disturbance increased. On the other hand, PID-only control produced a greater degradation in response to increasing wind disturbances. For both RL and PID, controllers trained on a single wind condition performed better than their peers when operated in that same condition. The worst performing case was PID control optimized under random wind disturbances. Ultimately, under wind disturbances twice the magnitude encountered during training, RL control under all wind conditions was better than the best PID controller. These results exhibit two insights. RL, while benefiting from specialized training for a desired condition, is also able to generalize to related conditions with no additional training. Conversely, optimizing a reactive PID controller via parameter search with performance over random wind directions as the objective



(a) Wind coming from 90°.



(b) Wind coming from 0°.

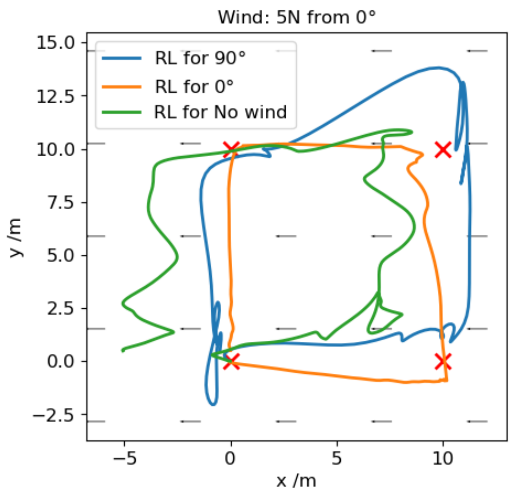
Figure 7.8: Performance of navigation under four granular tracking problems stacked together, representing a square trajectory. Each granular problem is navigation to the waypoint marked by "x".

is unable to come to a performant solution.

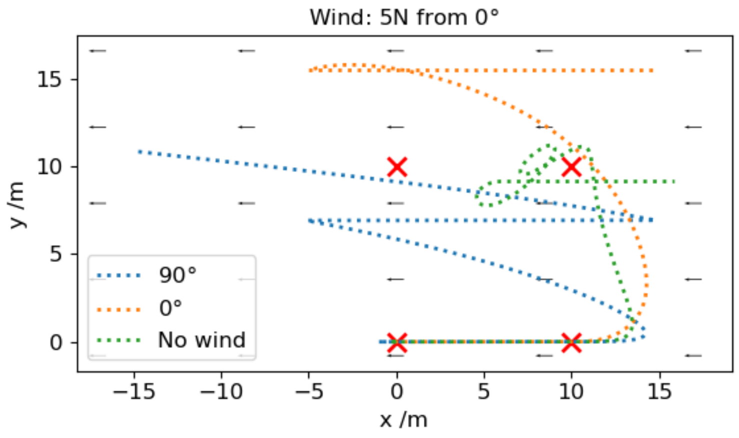
Why does the supervisory RL controller exhibit robustness to disturbances? Both controllers are trained on navigation problems initialized with random velocities and positions. Both have encountered states that may be present in nominal and disturbed environments, such as a net lateral velocity perpendicular to the reference position vector. But, a PID-only controller is unable to continue to plan actions over the trajectory because it needs a reference to follow. From Table 7.1, a PID controller compensates for wind by reducing position k_p, and increasing velocity k_p. Most significantly, the attitude k_d, k_i terms see an order of magnitude increase. That is, the velocity reference due to a position error is smaller. But, the attitude references respond faster to any change in velocity. And consequently, the angular rate references, determining the torque allocation for lateral motion, are more persistent and damped to counteract the effects of wind drag. While such persistent actions may counteract wind, under nominal conditions they would cause overshoot.

The RL controller, on the other hand, has memorized optimal actions from encountered states during training. When a state is encountered corresponding to a specific wind disturbance, such as a perpendicular wind velocity in comparison to the reference position vector, the controller can recall the appropriate actions. Unlike a PID-only controller, the size of the position error to track given to the PID controller, and the cascaded control logic, varies with the state of the vehicle besides position. For example, for headwinds, the position error is increased to give aggressive velocity references. For tailwinds, the error is decreased to ensure a smaller overshoot.

We further reinforce this result using Figure 7.12. Controllers, nominal and optimized for wind, are



(a) Supervisory RL.



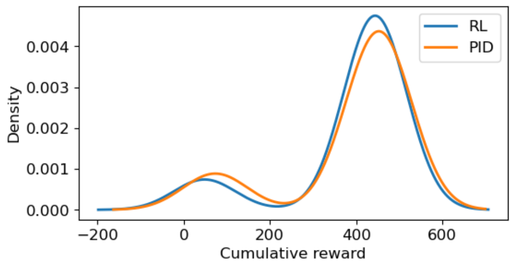
(b) Cascaded PID only.

Figure 7.9: A comparison of the robustness of (a) Supervisory RL and (b) PID-only for a wind condition. Controllers optimized for three different disturbances are tested for a single wind condition.

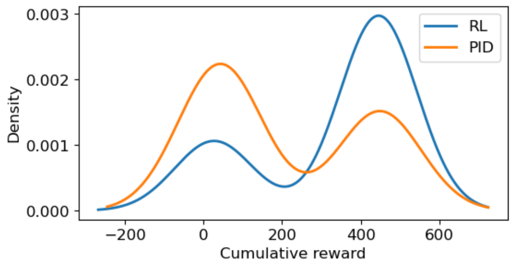
pitted against wind coming from eight different directions and evaluated over 10 episodes each. The PID-only controllers exhibit a larger variance in performance across the disturbances. On the other hand, RL supervision shows more consistent performance under novel disturbances.

Furthermore, the actions of RL controllers under different disturbances are related. The UAV system dynamics exhibit a rotational symmetry on the x, y plane. The state change magnitude in response to forces and torques along the x-axis is similar to that along the y-axis, except in a different direction. The supervisory controllers exhibit a similar relationship.

Figure 7.13a is a histogram of the angles of supervisory action in the x, y-plane, relative to the positive x-axis. Figure 7.13b is the same histogram, but shifted by 90°, representing the change in heading of the wind disturbances for respective controllers. We note that the modes of the distributions align. Further, the cosine

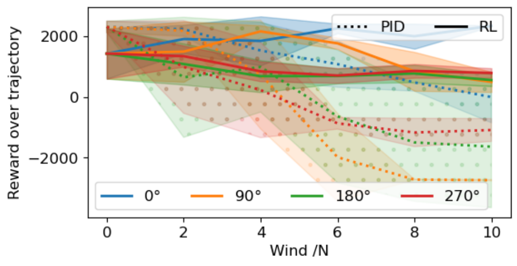


(a) No wind.

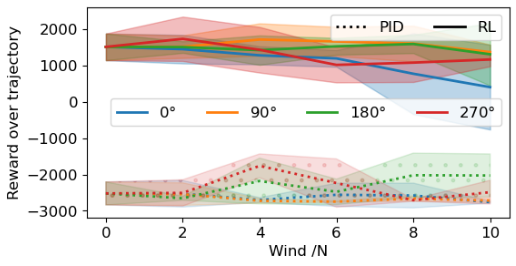


(b) Wind coming from 90°.

Figure 7.10: Performance comparison of running 30 experiments for waypoint following using PID-only, and PID with RL waypoint supervision.



(a) Performance over a square trajectory with increasing wind disturbance. RL and PID controllers trained for $5N$ wind from 0°.



(b) Controllers optimized for $5N$ wind from the 4 cardinal directions.

Figure 7.11: Performance over a square trajectory with increasing wind disturbance. RL and PID controllers trained for different wind disturbances are used over 5 trials.

similarity (Figure 7.13c), between histograms is the highest for a 90° translation among all other angles. The relationship between the actions that the disturbance conditions point to presents an interesting topic for future research. By identifying such relationships, a nominal controller can be adapted to a new control task without the need for tuning from scratch.

7.6.4 Operating robust RL control on longer trajectories

Figure 7.14 simulates longer trajectories by solving granular navigation problems in sequence as described in Algorithm 12. The trajectories are simulations of real-world experiments for the Tarot T-18 octo-rotor conducted by NASA Langley Research Center. The experiments are conducted to show the generalizability of our proposed approach. When there are no disturbances, both PID and RL controllers finish the designated path (see Figure 7.14a). However, under wind disturbance, when an RL controller, not trained on wind conditions is run, it fails (see Figure 7.14b). But when an RL controller trained on one wind disturbance encounters another novel disturbance, it is able to perform well (see Figure 7.14c). Furthermore, the RL controller can operate in nominal and windy conditions that may occur in the same flight. On the other hand,

	RL				PID			
	0°	90°	180°	270°	0°	90°	180°	270°
Wind /N								
0	1.42 ± 0.82	1.42 ± 0.82	1.42 ± 0.82	1.42 ± 0.82	2.29 ± 0.19	2.29 ± 0.19	2.29 ± 0.19	2.29 ± 0.19
2	1.89 ± 0.46	1.48 ± 0.99	1.08 ± 0.68	1.33 ± 0.34	2.22 ± 0.22	2.17 ± 0.20	0.64 ± 1.97	0.98 ± 1.52
4	1.83 ± 0.80	2.15 ± 0.20	0.68 ± 0.52	0.83 ± 0.32	1.52 ± 0.92	0.73 ± 1.79	0.98 ± 1.48	0.22 ± 1.56
6	2.24 ± 0.15	1.75 ± 0.18	0.68 ± 0.24	0.70 ± 0.24	1.07 ± 0.76	-1.98 ± 1.44	-0.64 ± 2.18	-0.87 ± 0.19
8	1.99 ± 0.41	0.83 ± 0.64	0.76 ± 0.29	0.85 ± 0.22	0.47 ± 0.29	-2.72 ± 0.02	-1.50 ± 1.93	-1.17 ± 0.46
10	2.32 ± 0.01	0.49 ± 0.33	0.55 ± 0.20	0.78 ± 0.16	-0.01 ± 0.84	-2.74 ± 0.02	-1.64 ± 2.00	-1.09 ± 0.38

Table 7.3: Tabulated results ($\times 10^3$) for figure 7.11a. Performance degradation with increasing wind disturbance from different headings. Controllers are optimized for $5N$ wind from 0°

	RL				PID			
	0°	90°	180°	270°	0°	90°	180°	270°
Wind /N								
0	1.51 ± 0.37	1.51 ± 0.37	1.51 ± 0.37	1.51 ± 0.37	-2.52 ± 0.32	-2.52 ± 0.32	-2.52 ± 0.32	-2.52 ± 0.32
2	1.45 ± 0.39	1.52 ± 0.31	1.51 ± 0.16	1.74 ± 0.61	-2.53 ± 0.36	-2.54 ± 0.31	-2.66 ± 0.06	-2.51 ± 0.36
4	1.28 ± 0.25	1.72 ± 0.45	1.43 ± 0.41	1.44 ± 0.63	-2.70 ± 0.07	-2.72 ± 0.06	-2.18 ± 0.63	-1.76 ± 0.33
6	1.20 ± 0.23	1.66 ± 0.40	1.53 ± 0.23	1.02 ± 0.48	-2.56 ± 0.32	-2.75 ± 0.02	-2.48 ± 0.35	-2.24 ± 0.66
8	0.78 ± 1.11	1.61 ± 0.49	1.59 ± 0.25	1.08 ± 0.53	-2.58 ± 0.34	-2.66 ± 0.07	-2.02 ± 0.61	-2.71 ± 0.03
10	0.41 ± 1.17	1.37 ± 0.21	1.31 ± 0.86	1.17 ± 0.19	-2.76 ± 0.03	-2.72 ± 0.05	-2.03 ± 0.59	-2.49 ± 0.33

Table 7.4: Tabulated results ($\times 10^3$) for figure 7.11b. Performance degradation with increasing wind disturbance from different headings. Controllers are optimized for $5N$ wind from the 4 cardinal directions.

a PID controller fails early when it encounters a disturbance (see Figure 7.14d). These results reinforce our hypothesis that the robust trajectory control problem can be decomposed into smaller sub-problems that are more tractable to solve for RL. At the same time, the controller can maintain generalizability for longer, more realistic trajectories.

7.7 Conclusions and Future work

In this work, we presented a supervisory control architecture for a class of unmanned aerial vehicles that is robust to wind disturbances. The supervisory RL controller is based on optimizing time to waypoint and deviations from the shortest path. The RL supervisor relies on the redundancy present in longer trajectories, and breaks down a longer desired path into granular navigation problems to reduce the state space for efficient reinforcement learning. The RL controller modifies the position references of the underlying PID controller, allowing it to run without interfering with the internals of low-level control and at a different frequency befitting its computational complexity.

Experiments with an octo-rotor simulation showed that, for wind disturbances, both RL and PID controllers tend to prefer persistent actions to maximally counteract the drag forces. An RL controller, being proactive, can modify the position reference marginally, causing the underlying cascaded PID logic to transparently cancel out disturbance effects.

Another characteristic of RL is robustness to disturbances. The RL controller *memorizes* the value of actions in the form of a neural network that operates over a continuous domain of states. Some states may occur during operation under the controller's nominal environment as well as for novel disturbances. Therefore, the controller can recall (or extrapolate) the most valuable action for a state. Contrast this with another planning algorithm, Model-Predictive Control, which will have to simulate a model recurrently to derive the best action. Under a new disturbance, the model may be inaccurate and thus output sub-optimal actions (Ahmed et al., 2018).

Finally, we explained the robustness of RL control by illustrating how changes in control actions and disturbances are related. By understanding the transformation in the environment (and system dynamics), an adaptive control strategy can be developed that will quickly respond to changes in the control problem by transforming the control policy correspondingly.

Prior work with RL and UAVs have made simplifying assumptions about control. For example, relaxed time constraints for inferring RL actions, or access to all actuators on the vehicle. While these assumptions can be realized with powerful computational hardware and bespoke circuitry, they do not lend themselves to wider, more general adoption. Our proposed supervisory approach is interoperable with popular flight control software like the Ardupilot. By keeping the RL control supervisory, we can separate it from the low-level control application, mitigating risks of complete software failure. The system necessarily operates at lower frequencies than the low-level PID control. Therefore, it can account for complex neural networks to learn RL control and it can operate on a wider range of computers. In addition, it can run on a separate (potentially remote) device, lowering payload requirements (battery, computer) for the vehicle.

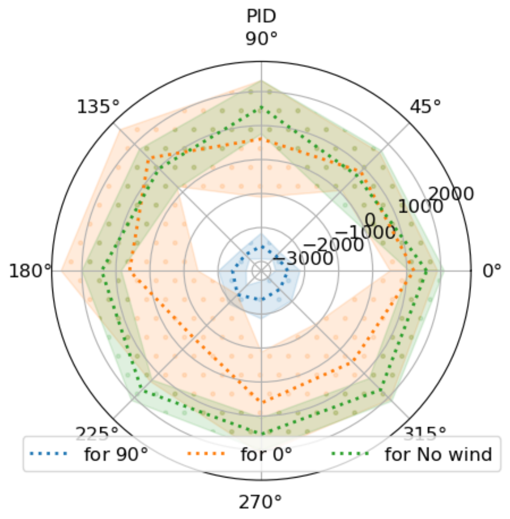
7.8 Implementation details

The code and data for this work are available at https://git.isis.vanderbilt.edu/ahmedi/transfer_similarity.

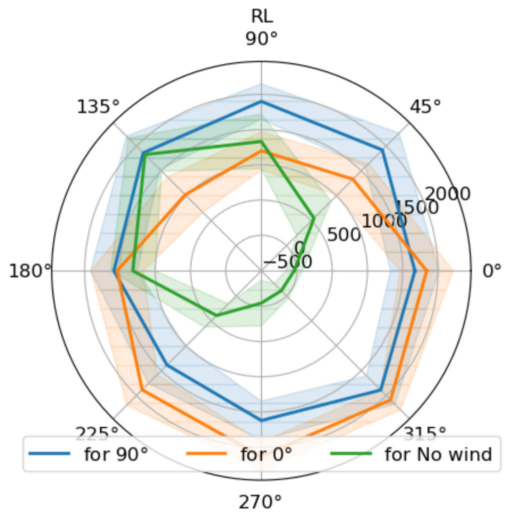
The testbed is a Tarot T-18 octorotor simulation, using the multirotor python library https://github.com/hazrmard/multirotor (Ahmed et al., 2022). The simulation time step is 0.01s. Attitude and rate controllers run at 100Hz, but the position and velocity controllers run at 10Hz. Transient dynamics of the motors are ignored when applying propeller speed signals to the UAV frame.

All reinforcement learning experiments were carried out using the `StableBaselines3` python library (https://stable-baselines3.readthedocs.io/en/master/). Proximal Policy Optimization was used for RL, with default parameters, except when overridden by the hyperparameter search documented in this work. The RL controller used a policy and value network of width 128, and depth 2 with tanh activations.

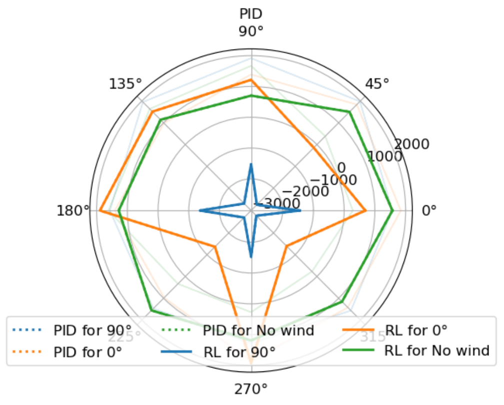
Hyperparameter optimization uses the Optuna library for finding PID and RL parameters (https://optuna.readthedocs.io/en/stable/index.html)(Akiba et al., 2019).



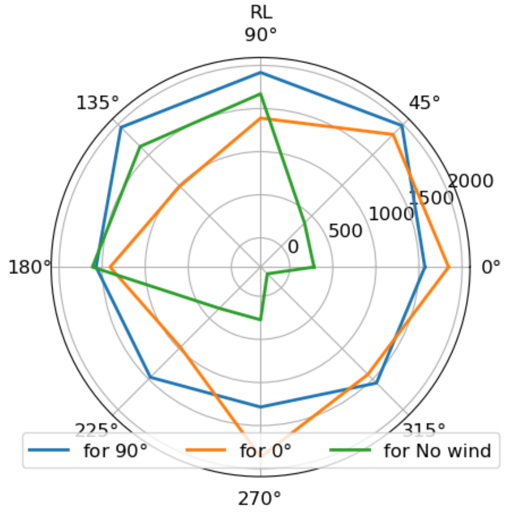
(a) Mean rewards over 10 square trajectories with random initial positions/velocities using PID.



(b) Mean rewards over 10 square trajectories with random initial positions/velocities using RL.

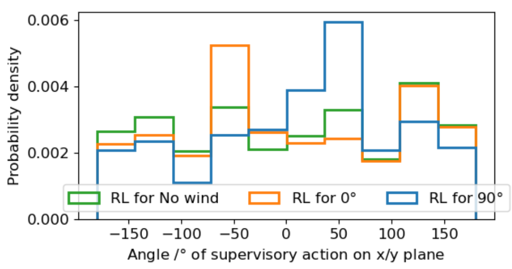


(c) Total rewards over 4 episodes comprising the square trajectory using PID.

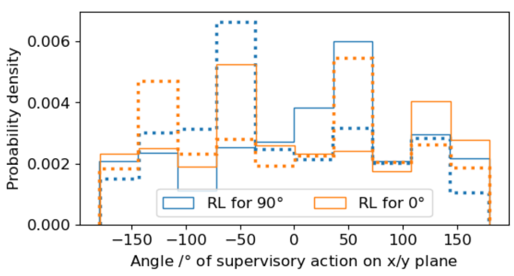


(d) Total rewards over 4 episodes comprising the square trajectory using RL.

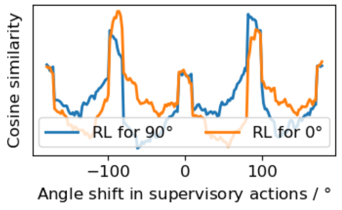
Figure 7.12: A polar plot of rewards for different controllers being tested under $5N$ wind coming from various headings.



(a) Histogram of RL controller actions' angles

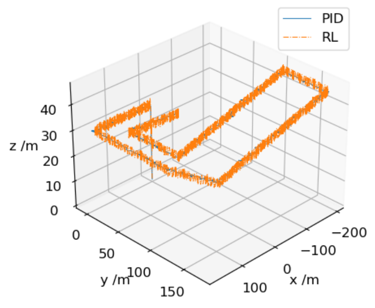


(b) Actions' angles shifted by 90°

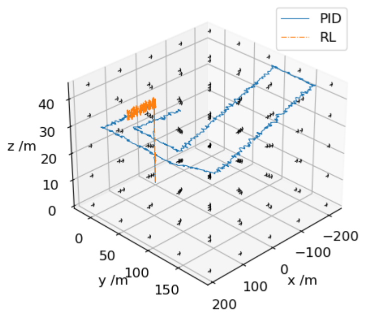


(c) Cosine similarity between histograms of angles of supervisory actions as they are shifted, against the histogram of the other controller's actions' angles.

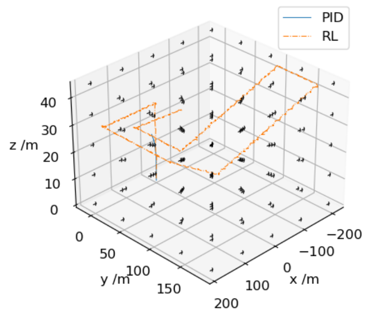
Figure 7.13: A density histogram showing relationship of the angles of the position vectors of the supervisory actions to modify the reference waypoint of the PID controller, calculated over 1000 randomly generated input states.



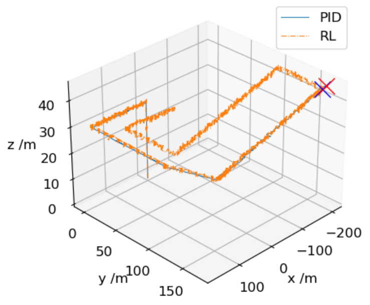
(a) Controllers for no wind, tested on no wind disturbances.



(b) Controllers for no wind, tested on smaller 3N wind coming from 0°.



(c) Controllers for 5N wind from 90°, tested on 0° wind.



(d) RL controllers for 5N wind from 90° and nominal PID controller, tested on 0° wind starting in the middle of the flight (blue cross). The cascaded PID-only controller fails shortly after (red cross).

Figure 7.14: Simulations of UAV flight under different disturbances over a longer flight.

CHAPTER 8

Reinforcement learning control with model-based adaptive control of UAVs under faults

8.1 Introduction

Unmanned Aerial Vehicles (UAVs) are highly dynamical systems, with small time constants and fast-moving parts that are susceptible to internal and external disturbances. Adaptive control is important for autonomous UAVs because it permits the controller to automatically adapt control parameters to changing flight conditions. This is crucial for ensuring safe and accurate navigation of the vehicle, especially when internal faults and disturbances affect vehicle dynamics. In general, adaptive control helps in maintaining vehicle control when parametric, structural, and environmental uncertainties affect system behavior, and the adaptive controller tracks the changes in the process to main system stability and performance. The adaptive controller, by self-tuning its parameters can provide real-time control.

Reactive controllers, such as proportional-integral-derivative (PID) use tailored parameters to continue to operate effectively when disturbances occur. Proactive control methods, such as Model Predictive Control (MPC) use an accurate model of the system and its environment to plan ahead. However, fine-tuning and testing parameters add computational load to control. Similarly, for MPC, solving a limited horizon optimization problem for each decision can become intractable for a system with a small time constant thus requiring a fast reaction.

While RL control methods suffer from the challenges of hyperparameter optimization and computational cost, they also have a number of benefits. The RL controller, once learned, has *memorized* the best control actions to take for a given state, dictated by anticipated feedback and tailored to maximize the accumulated future rewards. This solves the problem of reactivity in PID and ameliorates the computational costs associated with the MPC optimization problem.

8.2 Background

8.2.1 Reinforcement learning

RL frames the control of systems modeled as Markov Decision Processes (MDPs). Here we formalize the control problem from an RL perspective while drawing the parallels with standard control theory. An MDP control problem can be formalized as a task $T \in \mathscr{T}$, characterized by a process dynamic function $P : X \times U \to X$, which describes the probability of the system transitioning from the current state x_i to a next state x_{i+1}, given an input action u_i (or control signal). Each state transition is rewarded based on the state reached and the action that is taken to reach it based on a function $r : X \times U \to \mathbb{R}$ (cost/reward function). An episode is a sequence of interactions between the system and an agent (controller) until a terminal or goal state is reached. The objective for solving a task T is to derive a policy (data-driven control law) $\pi_T : X \rightarrow U$, such that each action selected maximizes the expected discounted future returns $\mathbb{E}[G(x_i)]$, where $G(x_i) = \sum_{j=i} \gamma^{j-i} r(x_i, \pi_T(x_i))$. The discount factor $\gamma \in (0,1]$ weights the importance of future versus immediate rewards. Expected discounted returns in a given state x under an optimal policy are known as its value, $V(x) = \max_u \mathbb{E}[G(x)]$. Then, the control optimization problem can be framed as $\pi_t(x) \leftarrow \arg\max_u V(x)$.

8.2.2 Transfer learning and adaptive control

The transfer problem is summarized as follows. A task T consists of the process dynamics P and the reward function r. Given a target task T_t and a population of source tasks $\mathscr{T}_s \in \mathscr{T}$, find a source task $T_s \in \mathscr{T}_s$ and a transfer mechanism, such that the performance of its policy fine-tuned on T_t provides an optimal solution for T_t. In other words,

$$T_s : \max_{T \in \mathscr{T}_s} G(x \sim T_t \mid \pi_{T \rightarrow T_t})$$

T_s and T_t may differ on process dynamics and reward. Both of these affect evaluated returns G. This objective can only be achieved retroactively when candidate source tasks have been evaluated on the target task. However, exploring with transfer of each source task may violate time and safety constraints in specific applications. Therefore, the challenge is to *preemptively* select a favorable source task and transfer mechanism. For this work, we assume homogeneous transfer: that the state and action spaces of all tasks are identical. Practically, this applies to cases where a single MDP's state transitions are disturbed, due to faults, degradation etc. For the remainder of this work, subscripts $*_s, *_t$ refer to source and target parameters respectively.

The goal, ultimately, is to achieve high returns on the target task. This may be done by either picking a source task liable to transfer well or by tweaking the transfer process such that the source policy converges swiftly to an optimum on the target task. Prior related work has addressed both of these approaches.

An adaptive controller is one "with adjustable parameters and a mechanism for adjusting the parameters" (Åström and Wittenmark, 2013) . Adaptive control can be viewed as a nested control loop: the inner loop actuates the system by taking actions $u \sim \pi(x)$, on the target process P_t, and the outer loop manipulates the operating parameters of the inner loop, such as the policy function. In that sense, adaptive control parallels meta-learning, wherein an outer "meta-learner" learns how the inner task learner learns from data.

8.2.3 Extended Kalman Filter

A fundamental aspect of control systems is state estimation, which involves obtaining an accurate estimate of the system's internal states based on noisy measurements. Filtering techniques, such as the Kalman Filters, have proven to be powerful tools for state estimation in linear systems. However, when dealing with non-linear systems, the Extended Kalman Filter (EKF) offers an effective solution.

Filtering is the process of extracting meaningful information from noisy measurements, removing undesirable disturbances and noise. In control systems, filtering is used to estimate the true state of a system based on observed measurements. State estimation, on the other hand, involves determining the internal states of a dynamic system that are not directly measurable or are subject to uncertainty.

Kalman Filters are based on the assumption that both the system dynamics and measurement equations are linear and that the noise in the system follows a Gaussian distribution. However, in practice, many systems exhibit non-linear dynamics or measurements, which violate these assumptions. Extended Kalman Filters (EKF) (Smith et al., 1962; Ribeiro, 2004) were developed to address this limitation by extending the Kalman Filter to handle non-linear systems. The EKF approximates the system's non-linear dynamics by using a linearization via Taylor series expansion. By iteratively linearizing the system equations around the current estimate, the EKF computes an approximate estimate of the system's true state.

EKF consists of two steps. In the prediction step, the prior belief in the state of the process is predicted $\bar{x}$, along with the covariance of that estimate via the linearized process. In the update step, a noisy measurement of process output is obtained, z. In the case of this work, the state is observable and is identical to process output. The residual $z - \bar{x}$ and its covariance, along with the prior estimates, are used to obtain the posterior estimate of state $\hat{x}$ and its covariance. This posterior estimate is then used by the control system to make decisions. The process is recursive: the last posterior is used to predict the next priors using the process model. The priors are then used to update the next estimate (posterior) using observed measurements.

8.2.4 Model-based adaptive control for linear systems

In this section, we describe prior work on a policy transformation for adaptive control. The policy transformation is derived by looking at how the process dynamics changed, and thus changing the policy such that those changes are counteracted. The transformation assumes that the objective function is not dependent on the action u, but on the state x only.

The optimal policy π - the control law - is derived to reach the optimal set of states *and* using optimal actions, since the reward is a function of both state and action. However, if the reward is only a function of performance (x), the optimal policy would depend only on the states traversed, not the effort (u) it took to traverse them.

$$
\begin{aligned}
& r : X \to \mathbb{R} \implies \nabla_u r = 0 \\
& \pi : \arg\max_{\pi} \mathbb{E}_{x_i \sim X} \sum_{j=i} \gamma^{j-i} r\left(P(x_i, \pi(x_i))\right) \qquad (8.1)
\end{aligned}
$$

That is, if the process experiences a change in process dynamics ($T_s : (P_s, r) \to T_t : (P_t, r)$), the optimal traversal of states remains the same, regardless of how the process responds to actions. The same set of states optimal under T_s are optimal under T_t. This comes with one caveat: P_t is controllable for the same set of states (Jurdjevic and Quinn, 1978; Sussmann and Jurdjevic, 1972). Therefore if π_t can bring about the same state change in P_t as π_s does in P_s, π_t can be guaranteed to be optimal.

To that end, sequential MDPs are framed as deterministic, dynamical time-invariant processes from a control theory lens. A process P has a state $x \in \mathbb{R}^n$, and inputs $u \in \mathbb{R}^m$. The process itself is characterized by the rate of change of its state $\dot{x} = F_{\dot{x}} : X \times U \to X$. For systems with linear dynamics, $F_{\dot{x}} = Ax + Bu$, where $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}$ are constant matrices. A is the response to internal state, and B is the response to external inputs.

Assuming $F_A : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}, F_B : \mathbb{R}^{n \times m} \to \mathbb{R}^{n \times m}$ are transformations of A_s, B_s, and π_s is the control law of the source task. The control law π_s already has optimized inputs to change states to optimal positions. We want a policy π_t such that the change of state under the target process is the same as the source process.

$$
\begin{aligned}
& \dot{x}_s = A_s x + B_s u_s \\
& \dot{x}_t = F_A A_s x + F_B B_s u_t \\
& \text{If } \dot{x}_t = \dot{x}_s \text{, then,} \\
& \implies F_A A_s x + F_B B_s u_t = A_s x + B_s u_s \\
& \implies u_t = (F_B B_s)^{-1}(I - F_A) A_s x + (F_B B_s)^{-1} B_s u_s \qquad (8.2) \\
& \implies u_t = \left((F_B B_s)^{-1}(I - F_A) A_s + (F_B B_s)^{-1} B_s \pi_s\right) x \qquad (8.3)
\end{aligned}
$$

Equation 8.2 represents a transformation of the source policy by $(F_B B_s)^{-1} B_s$, representing a change in the input's effect on state dynamics. And an additive correction by $(F_B B_s)^{-1}(I - F_A) A_s$, representing the changed internal dynamics of the system. Figure 8.1 and algorithm 13 depict how these transformations can be appended in series and parallel, respectively to an existing policy function π_s.

In summary, if reward is a function of only the state, and if optimal states in P_s are reachable in P_t, then

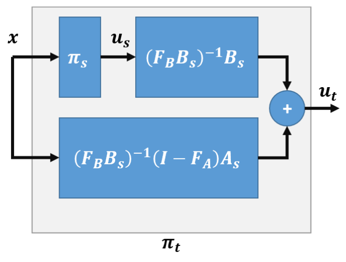


Figure 8.1: A schematic of the policy transformation. The nominal action u_s is transformed as u_t to bring about the same change in state in the target system that was considered optimal by π_s in the source system. The source policy can be any control algorithm for e.g. LQR, MPC, RL etc.

Algorithm 13 Policy transformation via system identification.

```
Require: : $\pi_s, T_s, T_t$
Require: : Data buffer $\mathscr{D}_t$
1: Collect $(x_i, u_{i+1}, x_{i+1})$ into $\mathscr{D}_t$ using $P_t, \pi_s$
2: Use system identification to $F_A, F_B$
3: Use equation 5.7 to get $\pi_t$
4: return $\pi_t$
```

optimal change in state remains invariant. Thus π_t can be derived as a transformation of π_s in terms of P_s and P_t to bring about the same change in state to optimize the target task.

8.3 Model-based adaptive control for non-linear systems

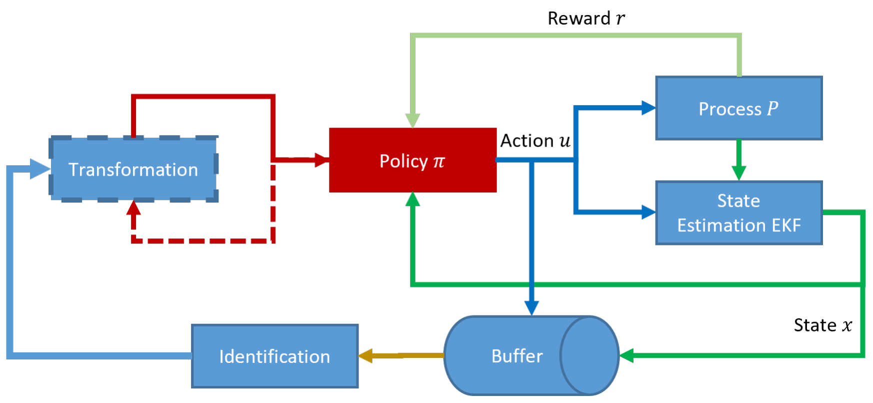


Figure 8.2: A structural schematic of the proposed approach. The policy block outputs actions based on estimated states using an Extended Kalman Filter. Recent states and actions are stored in a buffer for fault detection and identification (FDI). Upon fault detection, the proposed policy transformation is initiated.

Figure 8.2 shows the proposed approach of an adaptive supervisory controller. The main, "inner" loop is the interaction between policy π and the process P. The control policy maps estimated state of the process to an action. For this work, and Extended Kalman Filter (EKF) is used for estimation. The interactions consisting of state-action pairs are stored in a buffer to be used for fault detection and identification (FDI).

The adaptive, "outer" loop contains an FDI block and monitors control performance. Once a fault is detected, the policy transformation stage is initiated which adapts the parameters of the inner loop. Both EKF and the proposed policy transformation approach benefit from relying on a process model, which can be shared between them.

This work is principally concerned with adaptation via policy transformation of continuous, non-linear systems like UAVs. The following sections propose the transformation step in the figure. We assume a timely and accurate fault detection and identification as a pre-condition for this work.

8.3.1 Adaptive control using linear approximations

Policy transformation in section 8.2.4 assumes a linear system represented in the canonical state-space form. Generalizing A, B to maps, which may be non-linear, such that $A : \mathbb{R}^n \to \mathbb{R}^n$, and $B : \mathbb{R}^m \to \mathbb{R}^n$,

$$\dot{x} = A(x) + B(u) \tag{8.4}$$

Similarly, if F_A, F_B are maps, the policy transformation in equation 5.7 holds, given the required maps are invertible. For non-linear maps, non-linear basis functions such as neural networks can be used for representation. Since $F_B \circ B_s$ requires inversion, a monotonic constraint should be put on their learned models (by constraining hidden layer weights to be positive, for example). Or, using probabilistic models which learn a posterior distribution of a variable, given the output of a function Ardizzone et al. (2018) can be used, from which the likely inverse of the function can be sampled.

8.3.2 Adaptive control of controllable, non-linear systems

Extending to the general case of non-linear systems, where dynamics are represented by,

$$\dot{x}_i = \frac{\partial P_i}{\partial t} = \frac{\partial}{\partial t} P(x_i, u_i) \tag{8.5}$$

Then, using a multivariate Taylor series expansion until the first order about a state-action pair (x_0, u_0), and the pair (x, u) we want to solve for,

$$\dot{x}(x,u) = \frac{\partial}{\partial t} P(x_0, u_0) + \frac{\partial}{\partial x}\frac{\partial}{\partial t} P(x_0, u_0)(x - x_0) + \frac{\partial}{\partial u}\frac{\partial}{\partial t} P(x_0, u_0)(u - u_0) \tag{8.6}$$

When a fault occurs, the process dynamics change, where $\partial P_t/\partial t = F_P(\partial P_s/\partial t)$. Then, using the insight from section 8.2.4 that $\dot{x}_s = \dot{x}_t$, where $P_0 = P_s(x_0, \pi_s(x_0))$,

$$\frac{\partial P_0}{\partial t} = F_P\left(\frac{\partial P_0}{\partial t}\right) + \frac{\partial F_P\left(\frac{\partial P_0}{\partial t}\right)}{\partial x}(x - x_0) + \frac{\partial F_P\left(\frac{\partial P_0}{\partial t}\right)}{\partial \pi_t(x)}(\pi_t(x) - \pi_s(x_0)) \tag{8.7}$$

Then, solving for $\pi_t(x)$,

$$\pi_t(x) = \left(\frac{\partial F_P(\frac{\partial P_0}{\partial t})}{\partial u}\right)^{-1}\left(\frac{\partial P_0}{\partial t} - F_P(\frac{\partial P_0}{\partial t}) - \frac{\partial F_P(\frac{\partial P_0}{\partial t})}{\partial x}(x - x_0)\right) + \pi_s(x_0) \tag{8.8}$$

In this case, the state-action pair, $(x_0, \pi_s(x_0))$ is one encountered in the source process P_s. The pair $(x, \pi_t(x))$ is one encountered in the target process P_t which is being solved for. If $x = x_0$,

$$\pi_t(x) = \left(\frac{\partial F_P(\frac{\partial P_0}{\partial t})}{\partial u}\right)^{-1}\left(\frac{\partial P_0}{\partial t} - F_P(\frac{\partial P_0}{\partial t})\right) + \pi_s(x_0) \tag{8.9}$$

Assuming that a differentiable, data-driven model of $\partial P_s/\partial t$ is available as a function of (x, u), the terms on the right-hand side are all known before the new action u is taken. Contrasted with the policy transformation in equation 5.7, this general form of policy transformation is a function of x. For $\partial P_s(x, u)/\partial t = Ax + Bu$, $\partial P_t(x, u)/\partial t = F_A Ax + F_B Bu$, and as $x - x_0 \to 0$, the above relationship reduces to the aforementioned equation.

Equation 8.9 can be understood intuitively. $\left(\frac{\partial F_P(\frac{\partial P_0}{\partial t})}{\partial u}\right)^{-1} = \left(\frac{\partial u}{\partial F_P(\frac{\partial P_0}{\partial t})}\right)$, measures how the action changes with respect to a change in system dynamics. Whereas $\frac{\partial P_0}{\partial t} - F_P(\frac{\partial P_0}{\partial t})$ measures the change in rate of change of state between the source and target processes. Therefore, the product of the two is the approximate change in action merited due to the change in dynamics by F_P. The change is added to the nominal action $\pi_s(x_0)$ to give the guess for the action on the target process.

This policy transformation is based on a first-order expansion of process dynamics, assuming local linearity. While equation 5.7 linearizes a system about *all* states, this transformation linearizes the dynamics about the *current* state. That is, this is a piece-wise linearization of the system dynamics. While the prior equation can be successively applied to different states, estimating A, B matrices sequentially presents, it an additional computational load. In this case, a non-linear function learned (or provided) once can be linearized on demand. This can serve as an initialization for the optimal policy, which itself can be fine-tuned via RL.

For non-linear systems, the remainder of the first-order Taylor approximation can be used as a measure for the quality of the adaptation.

8.4 Adaptive control for UAVs

This section characterizes the specific research problem addressed by this work: trajectory control of a UAV under faults and disturbances. The goal is to maintain flight as close to the intended plan as possible without crashing the vehicle. This builds upon past work in chapter 7, where a supervisory RL controller was developed to navigate a UAV under disturbances. In that case, the flight objective (equation 7.4) was singularly to follow the waypoints in the shortest path possible.

For general UAV flight, the objective may not only be to follow the intended path, but also to minimize energy cost. Under faults, however, we assume the priority shifts. Thus we align the assumptions made for policy adaptation earlier (and in chapter 5), with the objective function used for RL control. That is, it is principally important that the vehicle stays as close to the desired trajectory under a fault as under the nominal condition. Therefore, the policy adaption which modifies control policy and tries to preserve state transitions becomes useful. We use a simulated octo-rotor as a validation test-bed. The following sections describe the control architecture and the proposed choice of policy adaptation mechanism.

8.4.1 The PID-controlled UAV test-bed

The the control logic of the UAV considered is depicted in figure 8.3. The following sequence of steps occur for the vehicle to fly towards a waypoint:

1. The PID controller receives a position reference. The PID is a cascaded set of controllers. The position error is converted to velocity, attitude, and angular velocity references. Finally the angular velocity error is used to compute the torque needed for tilting and achieving lateral motion. Attitude control similarly converts position error to attitude velocity error, and eventually to vertical thrust. The thrust and torques denote the prescribed dynamics $D = [F_z, \tau_x, \tau_y, \tau_z]$.

2. The allocation step assigns propeller velocities needed for achieving D. The thrust for each propeller p, $F_{z,p}$ is related to its speed ω_p by $F_{z,p} = k_{thrust} * \Omega_p^2$. The thrust is used for vertical motion and for generating torques about the lateral axes.

3. Electronic Speed Controllers take the speed signals and regulate the speed of individual motors.

4. Finally, the propellers generate net forces and torques about the body of the vehicle. The resulting state is reported back to the controller.

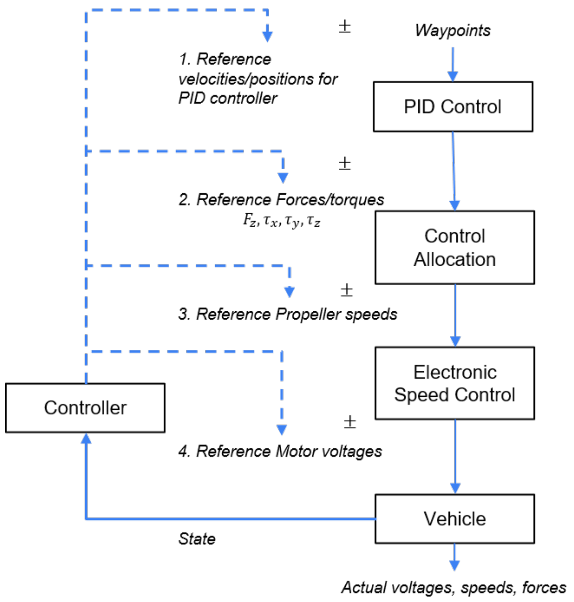

±
Waypoints
1. Reference velocities/positions for PID controller
PID Control
±
2. Reference Forces/torques
$F_z, \tau_x, \tau_y, \tau_z$
Control Allocation
±
3. Reference Propeller speeds
Electronic Speed Control
Controller
±
4. Reference Motor voltages
Vehicle
State
Actual voltages, speeds, forces

Figure 8.3: The control logic flow for an unmanned aerial vehicle (UAV).

Such a control approach presents multiple interaction points for interleaving supervisory control from a reinforcement learning controller. The choice of where to apply control to mitigate faults and disturbances is limited by theoretical and practical factors. Namely, the technical cost of modifying, and certifying software of the embedded PID controller. And the ability of different control signals to negate fault effects. Ultimately, the objective is optimal trajectory control of the UAV under faults. The challenge then is how to modify control architecture such that the originally intended trajectory is followed. For this work, we assume a parameter-based fault which is diagnosed and identified before our adaptation takes effect.

The UAV is modeled with the following dynamic state variables x_{uav} which govern flight (Ahmed et al., 2022):

- $\hat{r}^n \in \mathbb{R}^3$ are the navigation coordinates, in the global, inertial navigation reference frame axes n.
- $\hat{v^b} \in \mathbb{R}^3$ is the velocity of the vehicle in the vehicle's local, non-inertial body frame b.
- $\hat{\Phi} = [\phi, \theta, \psi] \in [-\pi, \pi]^3$ is the orientation of the body reference frame b in Euler angles (roll, pitch, yaw) with reference to the inertial reference frame n.
- $\hat{\omega} = [\omega_x, \omega_y, \omega_z] \in \mathbb{R}^3$ is the roll rate of the three body frame axes with reference to the inertial frame.

We consider two kinds of faults in the vehicle.

1. Motor failures occur individually in single motors, effectively modulating the thrust coefficient $k_{th} \leftarrow$

$k_{th} \cdot (1 - l_{mot})$, where for the speed for i^{th} propeller, Ω_i, $\texttt{thrust}_i = k_{th}\Omega_i^2$. We assume complete motor failures, reducing propeller speed to 0.

2. Motor saturation causes the speed of a motor to get stuck at one value, regardless of the control signal. This produces a constant torque throughout the flight of the vehicle.

8.4.2 Evaluating adaptive control approaches

Figure 8.4 shows various control layouts for this problem. A trajectory modification controller (figure 8.4a) modifies the reference position (input) of the vehicle's PID position controller. Whereas the dynamics modification controller (figure 8.4b) changes the output of the attitude rate and altitude velocity controllers. Finally, the reallocation controller (figure 8.4c) modifies control allocation to the vehicle propellers by converting prescribed dynamics to speed signals.

To gauge the ability of these layouts to adapt to faults, we examine the control pipeline closely (figure 8.5). The PID controller, $\mathbb{R}^3 \to \mathbb{R}^4$, takes reference position $\hat{r}_{wp} \in \mathbb{R}^3$ as input and outputs the required dynamics in the vehicle's non-inertial frame $D = [F_z, \tau_x, \tau_y, \tau_z]$ needed to match the waypoint. The control allocation block relates speed signals $\hat{\Omega} \in \mathbb{R}^8$ for each propeller to D. The allocation step can be modeled as a map $A : \mathbb{R}^8 \to \mathbb{R}^4$. Specifically,

$$\begin{bmatrix} F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \underset{4\times1}{D} = \underset{4\times8}{A} \cdot \underset{8\times1}{\hat{\Omega}^2}$$

$$\hat{\Omega} = \sqrt{A^{-1} \cdot D} \tag{8.10}$$

For a propeller p with a thrust coefficient $k_{th,p}$, a drag coefficient $k_{dr,p}$, at a radius r_p from the center of mass and angle θ_p from the reference forward direction in the vehicle's frame of reference, and where propellers are alternatively spinning clockwise and counter-clockwise,

$$A = \begin{bmatrix} \cdots & k_{th,i} & \cdots \\ \cdots & k_{th,i} \cdot r_p \sin(\theta_i) & \cdots \\ \cdots & k_{th,i} \cdot r_p \cos(\theta_i) & \cdots \\ \cdots & k_{dr,i} \cdot \texttt{sign}(i \mod 2) & \cdots \end{bmatrix} \tag{8.11}$$

When a fault occurs in the system, a propeller is unable to generate thrust and torque from the speed signal allocated to it. Note that for more than 3 co-planar propellers, $\sqrt{A^{-1}} : \mathbb{R}^4 \to \mathbb{R}^8$ is a consistent under-determined system. That is, there are multiple solutions for $\hat{\Omega}$ to get D. For example, if the PID controller for an octo-rotor doubles the thrust F_z to follow a reference, every other propeller can increase speed by a factor of 2. Or, all propellers can speed up by $\sqrt{2}$. Therefore, if a fault occurs, the allocation matrix may be modified to get a viable solution.

However, changing waypoint references may not mitigate fault effects. For instance, under a fault, a supervisory waypoint controller (figure 8.4a) may suggest a higher altitude reference such that the larger proportional response of the PID controller for F_z will compensate for the smaller thrust capability of a propeller. All propellers will speed up, except for the one with a fault. Imbalanced forces will cause a net torque, therefore causing lateral and yaw drift. Extending the above logic, changing prescribed dynamics D (figure 8.4b) will also be unable to compensate for faults.

In the following section, we present the adaptive controller architecture that is able to mitigate motor faults, by using insights from this discussion, and optimal policy transformation discussed in section 8.3.2.

8.5 Adaptive controller for UAV faults

From prior discussion, we have established that adaptation for motor faults necessarily must happen after the control allocation stage. The adaptation step applies to the allocation mechanism. It changes propeller speeds $\hat{\Omega}$ such that the ensuing change in state x_{uav} is preserved, as if no fault had occurred. Figure 8.6 shows the proposed control logic layout.

For the adaptive allocation mechanism, the action space is the vector $\hat{\Omega}$ of speed signals. The input to the allocation step is D, the dynamics prescribed by the trajectory controller, given the reference waypoint $\hat{r}_{wp}$. We consider x_{uav} as the state measurement of the system. The process `PID` is deterministic with respect to x_{uav}. If the change in x_{uav} is preserved, the inputs and outputs to the `PID` process will be invariant as well.

This means that the adaptation step can be decoupled from the direct output of a RL controller. For example, a supervisory RL controller may modify PID references, all the while the adaptation step is modifying the allocation outputs downstream. So long as the adaptation is able to maintain invariance in the state transitions of this deterministic process, it is effective.

The adaptation mechanism needs a model of the system dynamics. The output of the allocation step, $\hat{\Omega}$ can be converted to approximate resulting dynamics on the vehicle via the allocation matrix, $D_{uav} = A \cdot \Omega^2$. The `UAV` process is governed by the set of following equations. They take as input the state $x_{uav} = [\hat{r^n}, \hat{v^b}, \hat{\Phi}, \hat{\omega}]$ and the dynamics acting on the vehicle after actuating motors, $D_{uav} = [F_z, \tau_x, \tau_y, \tau_z]$. Equation 8.12 shows this relationship, where c_*, s_* represent sine and cosine of orientations.

$$\frac{dP_{uav}}{dt} = \begin{bmatrix} \dot{\hat{r}}^n_1 \\ \dot{\hat{r}}^n_2 \\ \dot{\hat{r}}^n_3 \\ \\ \dot{\hat{v}}^b_1 \\ \dot{\hat{v}}^b_2 \\ \dot{\hat{v}}^b_3 \\ \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \\ \dot{\hat{\omega}} \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi \hat{v^b}_1 + (-c_\phi s_\psi + s_\phi s_\theta c_\psi)\hat{v^b}_2 + (s_\phi s_\psi + c_\phi s_\theta c_\psi)\hat{v^b}_3 \\ c_\theta s_\psi \hat{v^b}_1 + (c_\phi c_\psi + s_\phi s_\theta s_\psi)\hat{v^b}_2 + (-s_\phi c_\psi + c_\phi s_\theta s_\psi)\hat{v^b}_3 \\ (-s_\theta \hat{v^b}_1 + s_\phi c_\theta \hat{v^b}_2 + c_\phi c_\theta \hat{v^b}_3) \\ \\ \frac{m}{F_x} + g s_\theta + \omega_3 \hat{v^b}_2 - \omega_2 \hat{v^b}_3 \\ \frac{m}{F_y} - g s_\phi c_\theta - \omega_3 \hat{v^b}_1 + \omega_1 \hat{v^b}_3 \\ \frac{m}{F_z} - g c_\phi c_\theta + \omega_2 \hat{v^b}_1 - \omega_1 \hat{v^b}_2 \\ \\ \omega_1 + (\omega_2 s_\phi + \omega_3 c_\phi) s_\theta / c_\theta \\ \omega_2 c_\phi - \omega_3 s_\phi \\ (\omega_2 s_\phi + \omega_3 c_\phi)/c_\theta \\ \\ I^{-1} \cdot (\hat{\tau} - \hat{\omega} \times I \cdot \hat{\omega}) \end{bmatrix} \tag{8.12}$$

Therefore the overall process, from the output of the allocation step, $\hat{\Omega}$ to the measurement of state, x_{uav}, becomes a function of $\hat{\Omega}, x_{uav}$ as $P_{uav}(A \cdot \hat{\Omega}^2, x_{uav})$, using equations 8.10, 8.12. Since an approximate model of process dynamics is known, it can be differentiated with respect to actions as required by equation 8.9 as $\partial P_{uav}/\partial\hat{\Omega}$. In this case, an analytical derivative is possible. However, we delegate that calculation to auto differentiation libraries like pyTorch to maintain generality with data-driven black-box models of processes.

Algorithm 14 Adaptive controller for trajectory following

```
Require: Trajectory of waypoints
Require: Nominal, faulty model of UAV dynamics $\frac{\partial P_{uav}}{\partial t}$, $F_P(\frac{\partial P_{uav}}{\partial t})$
Require: Nominal supervisory (RL or PID) controller $\pi_s : x \to \hat{\Omega}$
1: for $\hat{r}_{wp}$ in waypoints do
2:     Set reference position of supervisor to $\hat{r}_{wp}$
3:     Get nominally allocated speeds $\hat{\Omega}_s = u_s \leftarrow \pi_s(x)$
4:     Get re-allocation: $\hat{\Omega}_t = u_t \leftarrow \left(\frac{\partial F_P(\frac{\partial P_{uav}}{\partial t})}{\partial\hat{\Omega}}\right)^{-1} \left(\frac{\partial P_{uav}}{\partial t} - F_P(\frac{\partial P_{uav}}{\partial t})\right) + u_s$
5:     Apply action $\hat{\Omega}_t$
6:     Update state measurement of UAV $x_{uav} \leftarrow \{\hat{r}, \hat{v^b}, \hat{\Phi}, \hat{\omega}\}$
7: end for
```

8.6 Experiments

In this section, simulations are done with the two kinds of motor faults: complete failure and saturation. The efficacy of the adaptation is demonstrated under faults, and with supervisory control and external disturbances. Overall, we show a robust, adaptive flight control mechanism that is able to complete flight missions under adverse conditions. Supervisory RL and PID controllers are trained according the the methods described in our prior work.

As a simple example, we consider a take-off scenario under a complete motor failure in motor 1 (figure 8.7). The thrust and drag coefficients are set to 0. Under a nominal control policy, the vehicle is unable to take off and loses altitude. All propeller speed signals are the same to ensure no net torques and a net thrust. Under a fault, however, motor 1 is unable to counteract the opposing torques and meet the thrust requirement. But, when the policy is adapted with knowledge of fault dynamics, the vehicle is able to take off. Some motors are assigned lower speeds to minimize net yaw torque. Whereas other motors spin faster to make up for lost thrust. Ultimately, the vehicle follows a closer trajectory to the nominal case.

Note that the adapted trajectory is not identical to the nominal trajectory. This is the result of the assumption made during policy transformation. The transformation is based on a linear approximation of the process dynamics about the current state. Since the vehicle's dynamics are non-linear, this approximation is not fully accurate. We hypothesize that this "transfer" gap can be overcome by using a learnable supervisory controller.

In fact, the relative invariance in state trajectory bodes well for the fault-tolerant capability of a supervisory RL controller. RL optimizes a Markov Decision Process characterized by a state transition function. Under a fault, the transition function, therefore the distribution of states, is modified. The learned policy may no longer be optimal. By keeping the states seen by the supervisory controller similar to the nominal case, the adaptation block is able to ensure that the overall control loop is able to perform with minimal change to performance.

8.6.1 Motor failure

Next, we demonstrate the proposed adaptation on a longer trajectory under complete motor 1 failure (figure 8.8b). The adaptive controller is aware of the disturbance and fault effects to the dynamics. Our approach is compared against re-calculating the allocation matrix, given the diagnosed thrust and drag parameters ($k_{th,1} = 0, k_{dr,1} = 0$), as shown in equation 8.11. Since the values are known, and since the vehicle has redundant propellers, it is able to complete the trajectory (figure 8.8a) by fulfilling the allocated dynamics. Given a model of the process dynamics, our approach is also able to make minor adjustments to the propeller speeds, leading to a more smoother, faster trajectory.

8.6.2 Motor saturation

Under saturated motors, the thrust and torque outputs are constant, irrespective of the speed signal, breaking the quadratic relationship assumed in equation 8.10. Therefore, k_{thrust}, k_{drag} in the allocation matrix do not have a constant value for all motor speeds, to be able to predict the constant thrust accurately. Instead an approximation needs to be calculated. We compare our adaptation approach against an allocation matrix calculated with the "best-fit" coefficients, given the saturation value $\Omega_{sat,i}$ of the i-th motor. From the saturation value, the saturation thrust $F_{z,sat}$ and yaw torque $\tau_{z,sat}$ can be calculated. The coefficients are calculated by generating 100 pairs of values on the interval $\omega = [\Omega_{sat,i} - 50, \Omega_{sat,i} + 50]$, and solving the equation $F_{z,sat} = k_{th,sat} \cdot \Omega^2$ and $\tau_{z,sat} = k_{dr,sat} \cdot \Omega^2$. Figure 8.9 shows how, for saturation values closer to zero, the approximated coefficients are accurate. As the saturation value increases, so does the slope of the thrust/speed curve, and so does the approximation error. For larger saturation frequencies, a re-calculated allocation matrix is unable to complete the desired trajectory. A value of $\Omega_{sat} = 0$ is a special case corresponding to a motor failure, where the approximated thrust and drag coefficients will have zero error.

Figure 8.10a shows performance under a re-calculated allocation matrix. The coefficients are estimates as described. Under a small saturation fault of $\Omega_{sat,1} = 25 rad\ s^{-1}$, the vehicle is unable to complete the trajectory. However, our approach is able to dynamically change the allocated speeds at every step. Figure 8.10b shows a successful completion of a flight under a high saturation fault of $\Omega_{sat,1} = 450 rad\ s^{-1}$. We note how the adaptive allocation mechanism differs between failure (figure 8.8b) and saturation faults here. To compensate for the residual torques and thrust of a high motor speeds, the remaining motors are correspondingly spun down. Ultimately the trajectory seen in both cases is very similar. This reinforces our claim that the adaptation step is able to maintain a relative invariance in state trajectories, which reduces the transfer gap for an upstream supervisory controller.

8.6.3 Robust adaptation with disturbances and faults

Faults are not the only systemic changes experienced by a control system. We combine our prior work with this, by introducing external wind disturbances into the mix. First, performance under wind and faults using re-calculated allocation and our approach is demonstrated, but without any supervisory RL controller. Experiments are carried out under $5N$ wind blowing from East. Figure 8.11a shows how our approach is able to mitigate fault effects, such that the PID-controlled vehicle is able to finish the trajectory. However, for larger fault magnitudes, the adaptation step fails as shown in figure 8.11b. There is a larger variance in the magnitude of allocated speeds as the PID controller tries to track position with a constant wind disturbance displacing the vehicle.

Figure 8.12a shows a supervisory RL controller operating under a constant wind force by modifying waypoints to counteract wind disturbances. Figure 8.12b shows an adaptive controller operating under wind conditions and a saturation fault. While the vehicle is recurrently blown off-course, the supervisory RL controller is able to set the position error to counteract the disturbance. Concurrently, the adaptive controller is able to successfully allocate thrust and torques prescribed by the PID controller, and complete the flight.

We conduct a robustness study over different wind conditions and motor faults. The supervisory RL controller is trained on a 5*N* wind force blowing from the East. Complete motor failures are introduced in the system. During the experiments, the vehicle experiences a constant 5*N* wind coming from the North, and flies in a reverse figure eight curve, starting from the origin towards South-West. Figure 8.13 shows the result of these experiments. Tee heatmap is the total reward of running each flight. We examine some characteristics about the results. First, failure of motors 2,3,4,5 at high wind disturbances degrades performance. This is because those motors are South-facing. They need to spin up fast to induce a net North lateral force to counteract wind. There are also flight degradations for low wind speeds. This is due to the transfer learning gap. Recall that the supervisory controller was trained *only* on 5*N* Westwards wind. It learned to modify waypoints aggressively to fly into a headwind or along a tailwind. Therefore it must compensate for the absence of any external forces during operation. Future work may involve training trajectory control under a host of different disturbance conditions. It will come with a higher computational budget to ensure that the learning process converges to a sufficiently optimal policy.

Finally, a comparison is run using the policy transformation with instrumentation noise and state estimation via extended kalman filters (EKF), and perfectly observable process state.

Figure 8.14 compares the effect of using noise measurements, replicating instrumentation, for a flight. Using an EKF, an estimate of the vehicle's state is obtained for the PID and supervisory controllers to act upon. There is marginal difference between trajectories when the true state is known, versus when it is estimated. State estimation can add additional computational burden. Therefore, the update step is run at a lower frequency. In between update intervals, noisy measurements are used to predict state of the vehicle for control. The payoff between the utility of frequent updates and of the nature of faults is an interesting topic for future research.

8.7 Discussion and future work

In this paper a model-based adaptive, robust fault-tolerant control algorithm was developed. We established conditions in the control loop necessary for fault adaptation, and demonstrated our proposal on an octo-rotor simulation. The adaptive control algorithm changes actions such that the change in state due to a fault is nullified. It does this by linearizing the system dynamics model about each encountered state, and making additive modifications to the nominal action to reverse dynamics due to faults. This essentially preserves the state transition dynamics of a Markov Decision Process, such that a supervisory RL controller does not have to adapt to a new MDP even though a fault has occurred. Our results showed that the adaptive controller can work in conjunction with supervisory controllers under internal faults and external disturbances.

The nature of adaptive control algorithms is similar to meta-learning algorithms. Both have an outer loop which acts as an introspective component, modifying the inner algorithm such that it performs well on the control/learning task. In our work, we assume that the source policy is optimal. Given the state, it will take the optimal action to transition to the evaluated optimal state. We show that this is true for objective functions which are solely a function of state. Therefore, our adaptation step modifies actions such that the state transitions under the target task are similar to the transitions deemed optimal in the source task. The adaptation of policy parameters depends on reasoning about how the dynamics of the process changed. This is different from meta-learning algorithms, like MAML (Finn et al., 2017a). There, the parameters of the policy are changed such that they would be *expected* to perform well on a sample of candidate target tasks. This means that a sample of target tasks is available, and that some learning is conducted on the population of those tasks before the source policy can be adequately modified.

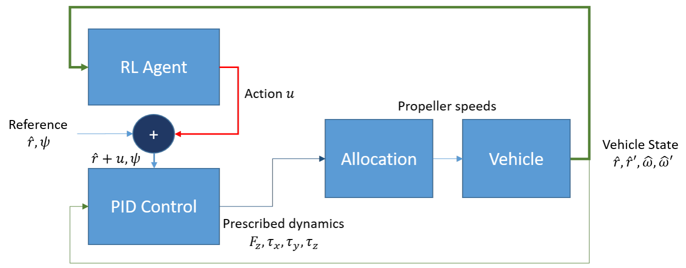
While the adaptation algorithm relies on a model of the nominal and faulty system, it is not restricted to an analytical formulation. In fact, the proposed algorithm makes room for any differentiable model (such as neural networks) of the system dynamics.

There are several limitations of this work. The adaptation step makes a first-order approximation of dynamics about each state. While this reduces the learning gap for the nominal supervisory controller towards the faulty system, it does not eliminate it entirely. Therefore, the remainder of the Taylor expansion of system dynamics (equation 8.9) may be used as a measure of the quality of adaptation.

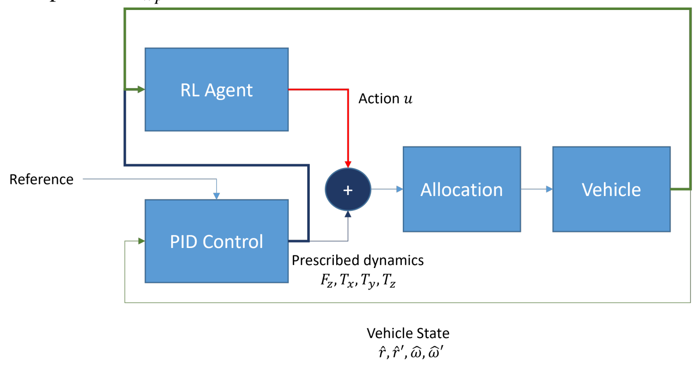
Secondly, we assume an accurate fault detection and identification step preempting the adaptation mechanism. This is a substantial pre-condition of our work. While in our prior work, a robust controller was able to function under novel disturbances without any knowledge that the MDP's transition function had changed, in this case the adaptation step will

When using data-driven models, there may be non-trivial prediction errors. Similarly, the validation errors on those models may signify the quality of our adaptation approach, and therefore inform any more control decisions to mitigate fault effects, such as ending a flight early.

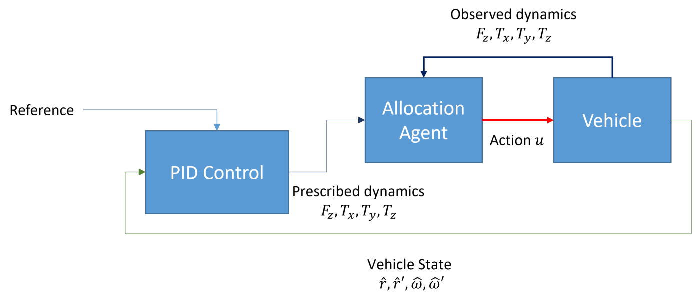
Future work pertains to examining the limitations of this approach in more detail. If, for example, the adaptation linearizes the system about an operating point, perhaps a piece-wise linear model of the system would yield similar results. This will ameliorate computational and design cost of providing a higher-order analytical model. We provided some conditions for linearization to be optimal in section 8.3.1). Furthermore, data-driven but physics-inspired models like neural ordinary differential equations may be used as data-driven models (Chen et al., 2018). The physical limitations imposed on state changes between steps will add to the safety guarantees of a data-driven policy learned from stochastic gradient descent.



(a) Waypoint modification. A supervisory controller adds to the reference position $\hat{r}_{wp}$ for the PID controller to track.



(b) Prescribed dynamics modification. A supervisory controller modifies the output of the PID controller before the prescribed dynamics D are allocated as speeds $\hat{\Omega}$ for the propellers.



(c) Dynamics allocation modification. The prescribed dynamics are modified (re-allocated) as speed signals for propellers.

Figure 8.4: Possible supervisory control layouts for the UAV problem.

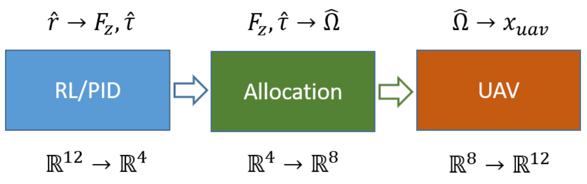

$\hat{r} \rightarrow F_z, \hat{\tau}$
$F_z, \hat{\tau} \rightarrow \hat{\Omega}$
$\hat{\Omega} \rightarrow x_{uav}$
RL/PID
Allocation
UAV
$\mathbb{R}^{12} \rightarrow \mathbb{R}^4$
$\mathbb{R}^4 \rightarrow \mathbb{R}^8$
$\mathbb{R}^8 \rightarrow \mathbb{R}^{12}$

Figure 8.5: The flow of control variables through the control, allocation, and actuation processes.

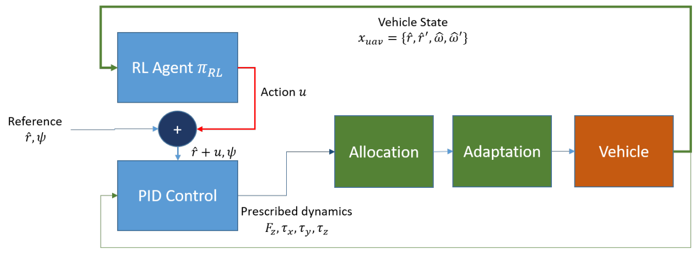


Figure 8.6: The proposed approach combines supervisory and adaptive control.

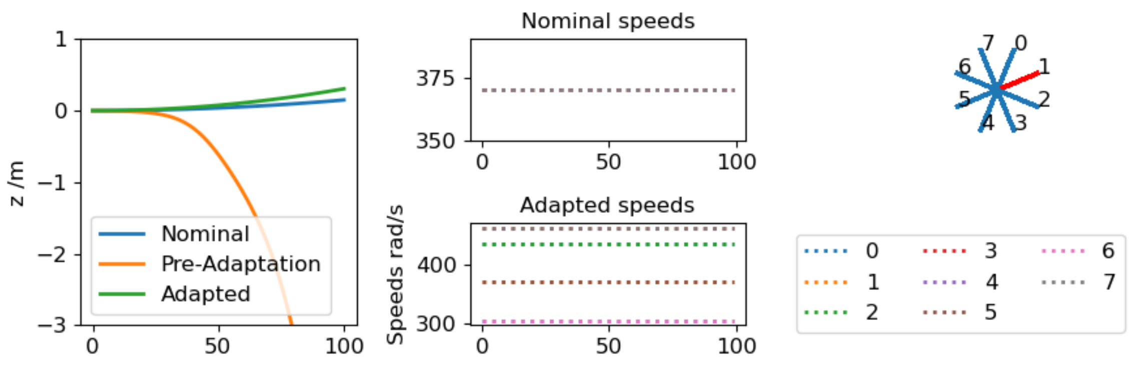
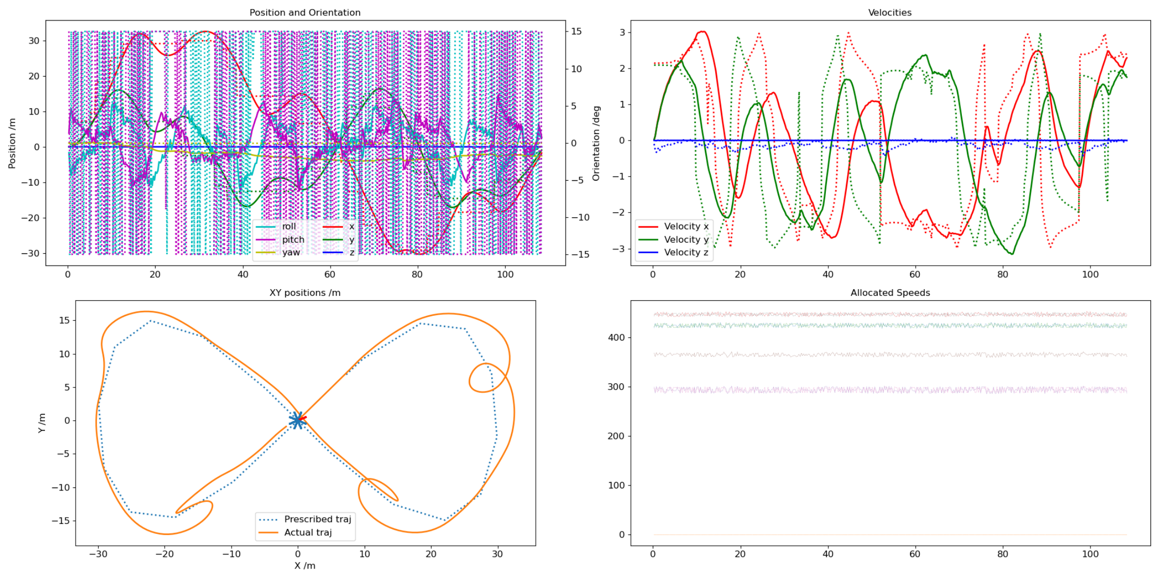
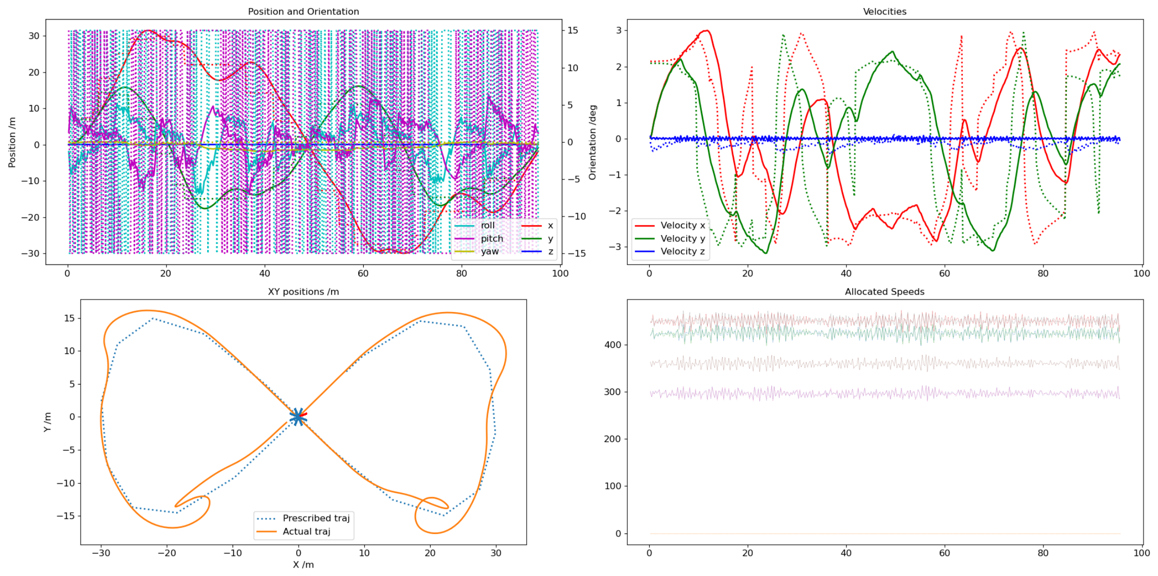


Figure 8.7: A take-off scenario showing fault adaptation.



(a) Performance under motor failure with re-calculated allocation matrix.



(b) Performance using our adaptation approach.

Figure 8.8: Comparison of flight performance with and without adaptation under motor failure, $\Omega_{sat,1} = 0$. Note that our approach finishes the trajectory faster.

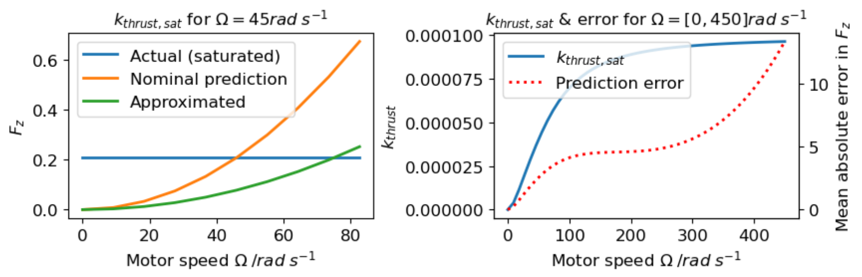
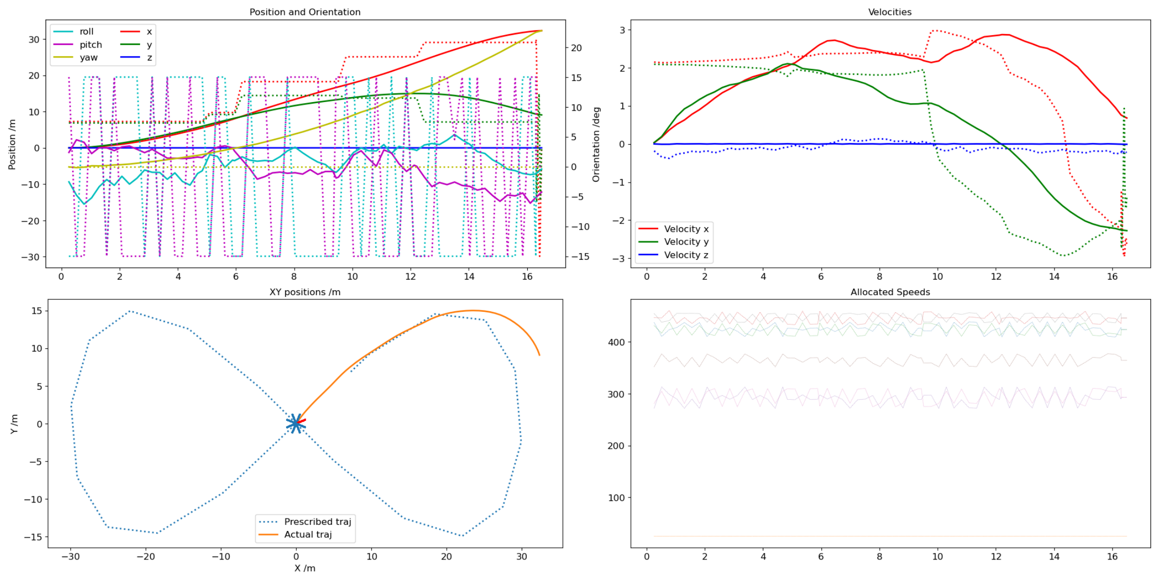
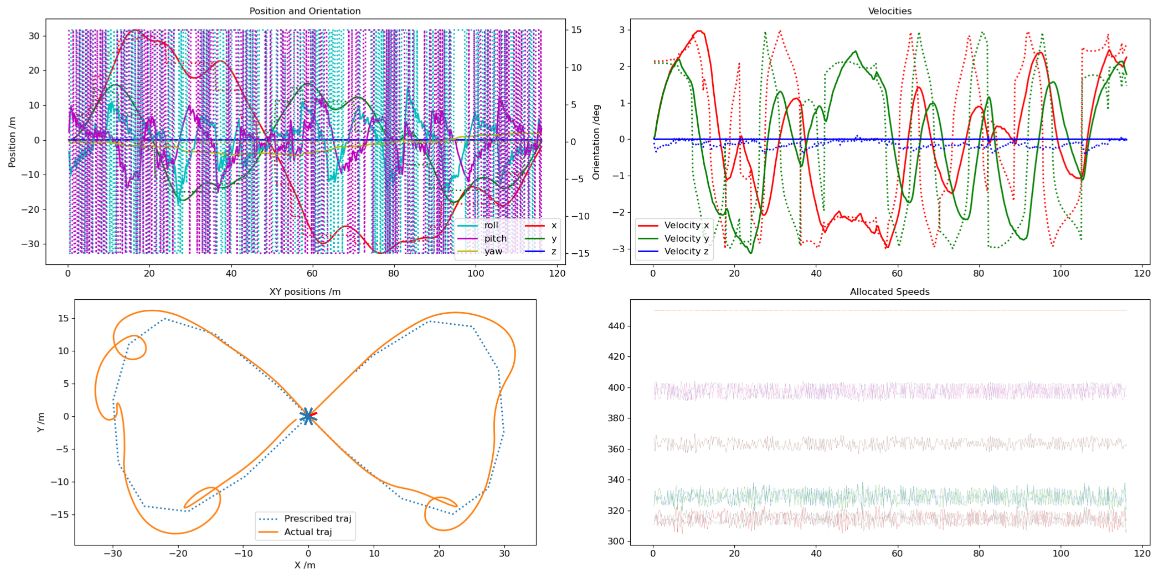


Figure 8.9: Plotting the dynamic thrust coefficient under saturation faults. The approximation error increases with higher saturation speed, leading to worse performance from re-calculating the allocation matrix.

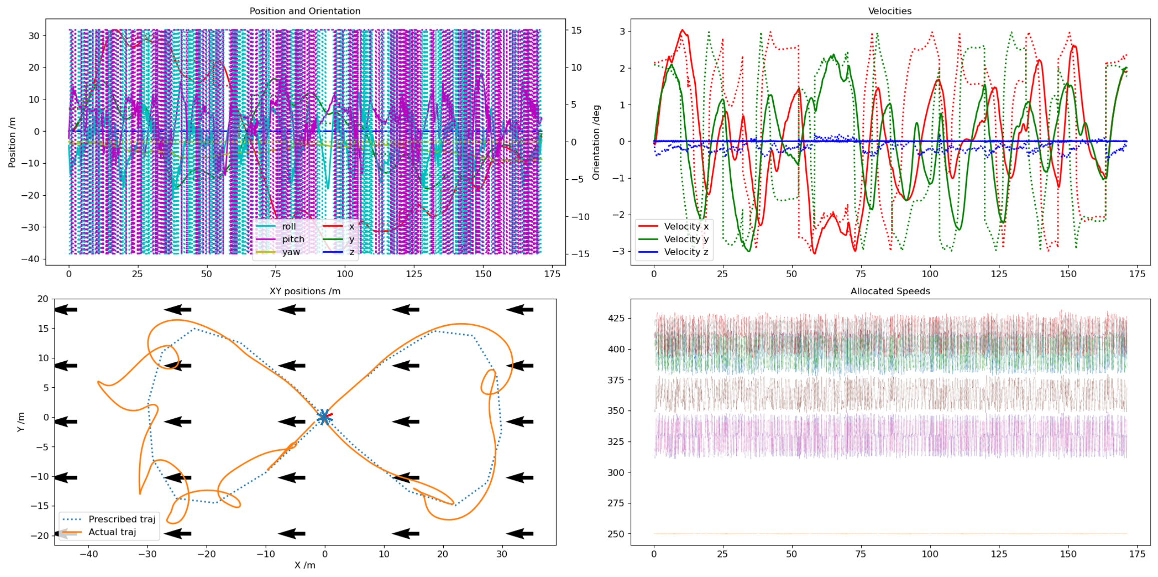


(a) Re-calculated allocation matrix for $\Omega_{sat,1} = 25$.

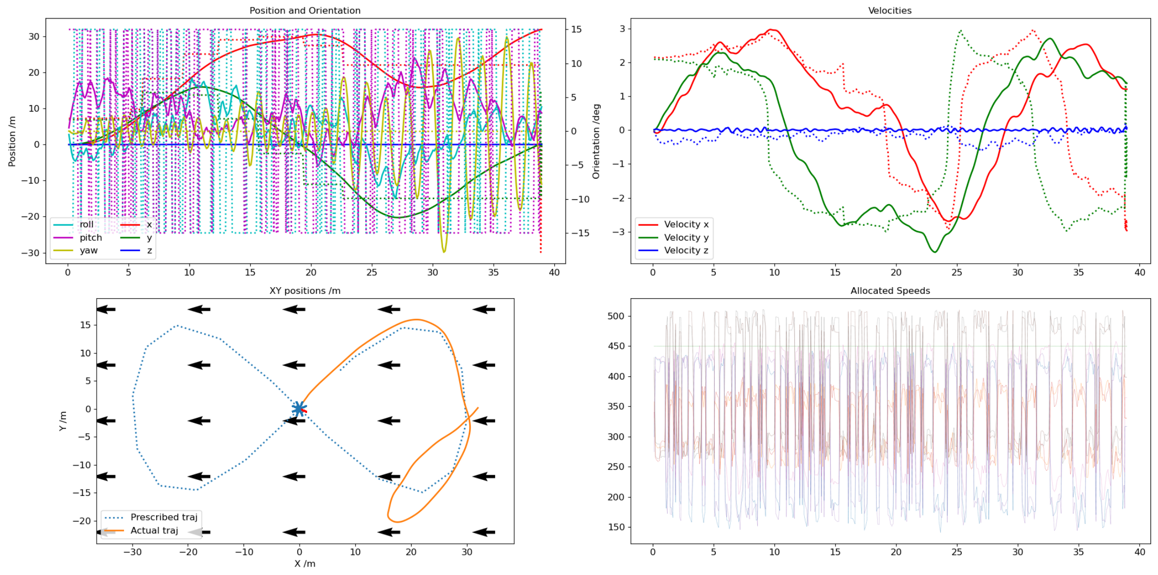


(b) Our adaptation approach for $\Omega_{sat,1} = 450$.

Figure 8.10: Comparison of flight performance with and without adaptation under saturation fault.

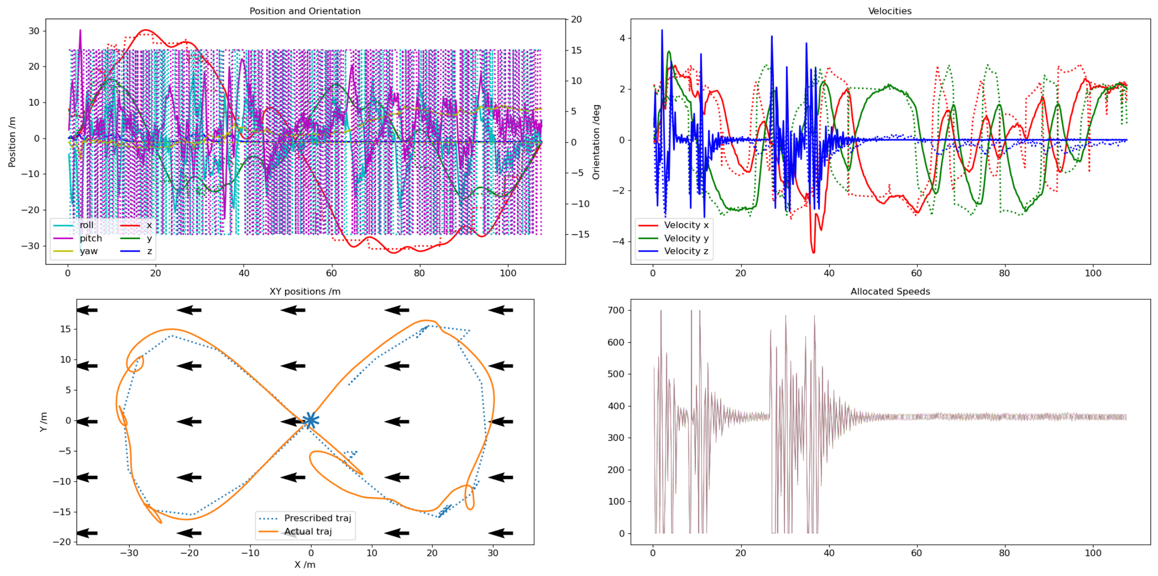


(a) Our adaptation approach for $\Omega_{sat,1} = 250$ and under a 5N wind coming from East.

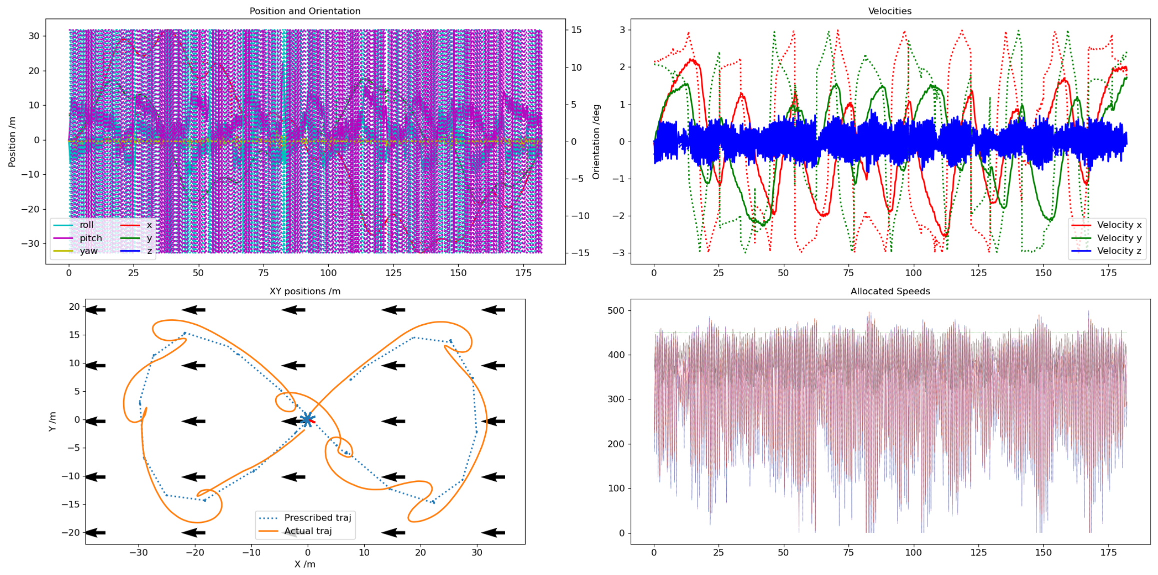


(b) Our adaptation approach for $\Omega_{sat,2} = 450$ and under a 5N wind coming from East.

Figure 8.11: No supervisory control. Comparison of flight performance with and without adaptation under saturation fault and with external wind disturbances.



(a) RL supervisory trajectory control under wind disturbance and no fault. A purely supervisory trajectory controller cannot adapt to a motor fault and will fail.



(b) Our adaptation approach for $\Omega_{sat,2} = 450$ and under a 5N wind coming from East.

Figure 8.12: With supervisory control. Comparison of flight performance with and without adaptation under saturation fault and with external wind disturbances.

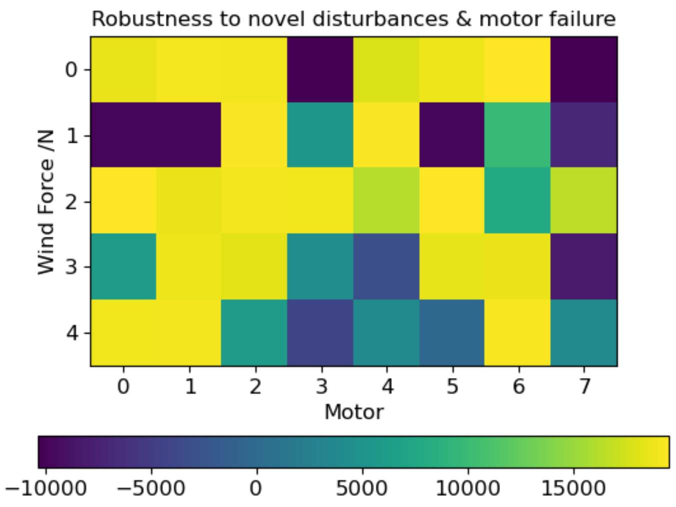


Figure 8.13: Cumulative episodic rewards with different motor failures and wind forces.

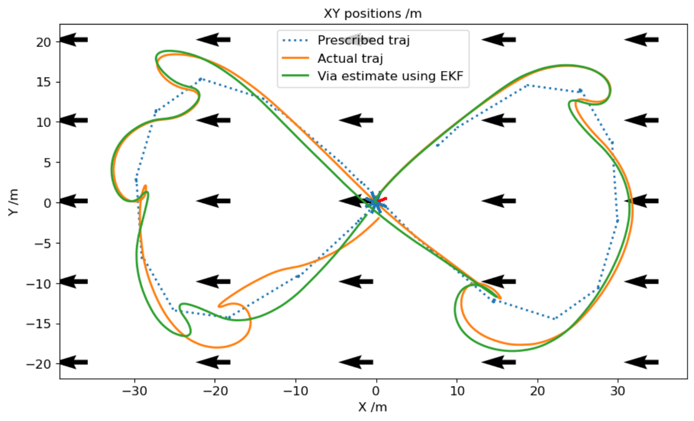


Figure 8.14: Comparison of flown trajectory using true measurement of state and an EKF. The update step of the EKF is run every 10 simulation steps, at 10Hz. The prediction step is run at 100Hz. The conditions are $\Omega_{sat,1} = 250$ and a 5N wind coming from east. Instrument noise is distributed normally with variances of $\sigma^2 = 5m^2, 0.5m^2s^{-2}, 0.01rad^2, 0.01rad^2s^{-2}$ for position, velocity, orientation, and angular rate.

CHAPTER 9

Conclusion

Adaptive, fault-tolerant control of hybrid systems is becoming an inalienable part of an increasingly autonomous world. In this work, reinforcement learning was explored as an adaptive control solution. RL is inherently adaptive because it learns from experience to optimize for an objective, instead of learning from previously labelled data under supervision. However, RL comes with a host of challenges.

Foremost among them is sample efficiency. RL approaches use stochastic machine learning algorithms (like stochastic gradient descent) to learn policies interacting with Markov Decision Processes. For the sake of learning stability, updates using SGD are small and many. Furthermore, the algorithm learns by continuously interacting with the MDP using its concurrently updating policy. Therefore, RL relies on large sets of interactions sampled over time. This means that an adaptive algorithm needs to be efficient in the number of samples to be of practical use to a real-world system where time is a constraint.

When systemic changes occur, such as an internal fault or external environmental disturbances, RL learns to corresponding policy. But, there may not be enough time to converge to a sufficiently optimal policy before system failure. In this work, fault-tolerant control of hybrid systems is analyzed with that challenge in mind. Two approaches are taken from the outset:

1. Adaptation to faults is considered in the machine learning algorithm domain. That is, the machine learning process itself is modified, agnostic to the underlying system, to learn fast. This is done by:
    (a) Learning data-driven models to synthesize data (section 4.2).
    (b) Transfer-learning from similar policy functions reusing a similar policy. The Jensen-Shannon metric is proposed for quantifying similar policies to transfer from to reduce transfer gap (section 4.3).
    (c) Using meta-learning to distill some policies into a new initialization. Instead of sampling different processes anew for the inner update in Model-agnostic Meta-Learning, here, previously learned policy parameters are exploited (section 4.4).
2. Adaptation to faults is considered in the dynamics domain. That is, agnostic to the machine learning algorithm, the policy is transformed by reasoning about how the system dynamics changed. The policy, once transformed, is reused as an initialization for the target system to be further tuned via RL.
    (a) In this work, a policy transformation is devised which works with a linearized system. It is

optimal for a linear system optimizing for an objective which is a function of state only. The policy transformation can be appended to control laws derived from RL, MPC, and LQR among others (section 5.1).

(b) For non-linear systems, or linear systems with an inaccurate model, a quality measure is proposed to evaluate the transformation. In some cases, such as highly non-linear domains, the prior transformation may have detrimental effects on adaptation (section 5.3).

(c) Finally, a generalized policy transformation is proposed for non-linear systems. It does not assume a linear model of system dynamics from the outset. Instead, it adapts actions for each state by piece-wise linearizing about that operating point, given the non-linear model (section 8.3).

In addition, practical constraints of developing RL control for complex systems are addressed. Specifically, case studies are done for developing transferable control for HVAC systems (section 4.5), and developing robust and adaptive trajectory control for UAVs. Principally, focus is directed towards practical fault-tolerant control of UAV systems. A high-fidelity test-bed is developed for RL research (chapter 6). Then, a supervisory RL controller robust to novel wind disturbances, and interoperable with extant flight control software is developed (chapter 7). Finally, robust control is combined with adaptive fault-tolerant control under UAV motor faults using our proposed approach (chapter 8).

The thesis of the latter half of this work is exploiting domain-specific knowledge when using machine learning for control. Adaptation in ML tends to be agnostic to the underlying system. Instead, data is opaquely extracted from the system, and the ML process focuses on efficiently learning a mapping between features (states) and labels (actions). The proposed adaptation in RL via policy transformation ideally relies on domain knowledge. While data-driven models of source and target systems are compatible with this approach, they suffer from the same drawback of inefficiency. However, if an accurate diagnosis of disturbances or faults is made, the adaptation is able to provide an instant change in the policy which reduces the transfer gap for the extant policy on the new task.

Appendix A

Publications

The following publications have been made over the course of this work:

1. Ahmed, Ibrahim, Marcos Quinones-Grueiro, and Gautam Biswas. ”A Reinforcement Learning Approach for Robust Supervisory Control of UAVs Under Disturbances.” Applied Soft Computing Journal. (in review)

2. Ahmed, Ibrahim, Marcos Quinones-Grueiro, and Gautam Biswas. ”Model-based adaptation for sample efficient transfer in reinforcement learning control of parameter-varying systems.” IEEE CoDiT. 2023

3. Ahmed, Ibrahim, Marcos Quinones-Grueiro, and Gautam Biswas. ”Adaptive fault-tolerant control of octo-rotor UAV under motor faults in adverse wind conditions.” AIAA SCITECH 2023 Forum. 2023

4. Ahmed, Ibrahim, Marcos Quinones-Grueiro, and Gautam Biswas. ”A high-fidelity simulation testbed for fault-tolerant octo-rotor control using reinforcement learning.” 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC). IEEE, 2022.

5. Ahmed, Ibrahim, Marcos Quinones Grueiro, and Gautam Biswas. ”Analysis of the deployment strategies of reinforcement learning controllers for complex dynamic systems.” Annual Conference of the PHM Society. Vol. 13. No. 1. 2021.

6. Ahmed, Ibrahim, Marcos Quinones-Grueiro, and Gautam Biswas. ”Transfer reinforcement learning for fault-tolerant control by re-using optimal policies.” 2021 5th International Conference on Control and Fault-Tolerant Systems (SysTol). IEEE, 2021.

7. Ahmed, I., M. Quiñones-Grueiro, and G. Biswas. “Complementary Meta-Reinforcement Learning for Fault-Adaptive Control”. Annual Conference of the PHM Society, vol. 12, no. 1, Nov. 2020, p. 8, doi:10.36001/phmconf.2020.v12i1.1289.

8. Ahmed, Ibrahim, Marcos Quiñones-Grueiro, and Gautam Biswas. ”Fault-Tolerant Control of Degrading Systems with On-Policy Reinforcement Learning.” IFAC-PapersOnLine 53.2 (2020): 13733-13738.

9. Naug, Aviek, Ibrahim Ahmed, and Gautam Biswas. ”Online energy management in commercial buildings using deep reinforcement learning.” 2019 IEEE International Conference on Smart Computing (SMARTCOMP). IEEE, 2019.

10. Ahmed, Ibrahim, Hamed Khorasgani, and Gautam Biswas. ”Comparison of model predictive and reinforcement learning methods for fault tolerant control.” IFAC-PapersOnLine 51.24 (2018): 233-240.

References

(2022a). Energy use in commercial buildings. https://www.eia.gov/energyexplained/use-of-energy/.

(2022b). Use of energy explained. https://www.eia.gov/energyexplained/use-of-energy/.

Abdelwahed, S., Wu, J., Biswas, G., Ramirez, J., and Manders, E. J. (2005). Online fault adaptive control for efficient resource management in advanced life support systems. *Habitation*, 10(2):105–115.

Abel, D., Jinnai, Y., Guo, S. Y., Konidaris, G., and Littman, M. (2018). Policy and value transfer in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 20–29. PMLR.

Adetola, V., DeHaan, D., and Guay, M. (2009). Adaptive model predictive control for constrained nonlinear systems. *Systems & Control Letters*, 58(5):320–326.

Afram, A. and Janabi-Sharifi, F. (2014). Theory and applications of hvac control systems–a review of model predictive control (mpc). *Building and Environment*, 72:343–355.

Ahmed, I. (2017). Qlearning. Github repository.

Ahmed, I., Khorasgani, H., and Biswas, G. (2018). Comparison of model predictive and reinforcement learning methods for fault tolerant control. *IFAC-PapersOnLine*, 51(24):233–240.

Ahmed, I., Quiñones-Grueiro, M., and Biswas, G. (2020). Fault-tolerant control of degrading systems with on-policy reinforcement learning. *IFAC-PapersOnLine*, 53(2):13733–13738.

Ahmed, I., Quiñones-Grueiro, M., and Biswas, G. (2021). Transfer reinforcement learning for fault-tolerant control by re-using optimal policies. In *2021 5th International Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 25–30. IEEE.

Ahmed, I., Quinones-Grueiro, M., and Biswas, G. (2022). A high-fidelity simulation test-bed for fault-tolerant octo-rotor control using reinforcement learning. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE.

Ahmed, I., Quinones-Grueiro, M., and Biswas, G. (2023). Adaptive fault-tolerant control of octo-rotor uav under motor faults in adverse wind conditions. In *AIAA SCITECH 2023 Forum*, page 2535.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Al-Emran, M. (2015). Hierarchical reinforcement learning: a survey. *International journal of computing and digital systems*, 4(02).

Alam, M. F., Shtein, M., Barton, K., and Hoelzle, D. J. (2022). Sample efficient transfer in reinforcement learning for high variable cost environments with an inaccurate source reward model. In *2022 American Control Conference (ACC)*, pages 2892–2898.

Alwi, H. and Edwards, C. (2013). Fault tolerant control of an octorotor using lpv based sliding mode control allocation. In *2013 American Control Conference*, pages 6505–6510. IEEE.

Ammar, H. B., Eaton, E., Taylor, M. E., Mocanu, D. C., Driessens, K., Weiss, G., and Tuyls, K. (2014). An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Anderson, K. L., Blankenship, G. L., and Lebow, L. G. (1988). A rule-based adaptive pid controller. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 564–569. IEEE.

Antoniou, A., Edwards, H., and Storkey, A. (2018a). How to train your maml. In *International Conference on Learning Representations*.

Antoniou, A., Edwards, H., and Storkey, A. (2018b). How to train your maml. *arXiv preprint arXiv:1810.09502*.

Ardizzone, L., Kruse, J., Rother, C., and Köthe, U. (2018). Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Åström, K. J. and Wittenmark, B. (2013). *Adaptive control*. Courier Corporation.

Baldi, S., Sun, D., Xia, X., Zhou, G., and Liu, D. (2022). Ardupilot-based adaptive autopilot: architecture and software-in-the-loop experiments. *IEEE Transactions on Aerospace and Electronic Systems*, 58(5):4473–4485.

Barrett, S., Taylor, M. E., and Stone, P. (2010). Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, volume 1.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20.

Bengea, S. C., Li, P., Sarkar, S., Vichik, S., Adetola, V., Kang, K., Lovett, T., Leonardi, F., and Kelman, A. D. (2015). Fault-tolerant optimal control of a building hvac system. *Science and Technology for the Built Environment*, 21(6):734–751.

Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE access*, 6:64270–64277.

Blanke, M., Izadi-Zamanabadi, R., Bogh, S. A., and Lunau, C. P. (1997). Fault-tolerant control systems - a holistic view. *Control Eng. Practice*, 4(5):693–702.

Bocsi, B., Csató, L., and Peters, J. (2013). Alignment-based transfer learning for robot models. In *The 2013 international joint conference on neural networks (IJCNN)*, pages 1–7. IEEE.

Bregon, A., Biswas, G., and Pulido, B. (2011). A decomposition method for nonlinear parameter estimation in transcend. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(3):751–763.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015a). Reinforcement learning from demonstration through shaping. In *Twenty-fourth international joint conference on artificial intelligence*.

Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015b). Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188.

Cabi, S., Colmenarejo, S. G., Hoffman, M. W., Denil, M., Wang, Z., and Freitas, N. (2017). The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. In *Conference on Robot Learning*, pages 207–216. PMLR.

Cao, B., Pan, S. J., Zhang, Y., Yeung, D.-Y., and Yang, Q. (2010). Adaptive transfer learning. In *proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 407–412.

Carroll, J. L. and Seppi, K. (2005). Task similarity measures for transfer in reinforcement learning task libraries. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 803–808. IEEE.

Chakraborty, I., Chen, Y., Li, J., and Vrabie, D. (2020). Theoretical development of controller transfer applied to dynamical systems. In *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*, pages 445–453.

Chamseddine, A., Theilliol, D., Zhang, Y., Join, C., and Rabbath, C.-A. (2015). Active fault-tolerant control system design with trajectory re-planning against actuator faults and saturation: application to a quadrotor unmanned aerial vehicle. *International Journal of Adaptive Control and Signal Processing*, 29(1):1–23.

Charles Tytler (2017). Modeling Vehicle Dynamics - Quadcopter Equations of Motion.

Chen, B., Liu, Z., Zhu, J., Xu, M., Ding, W., Li, L., and Zhao, D. (2021). Context-aware safe reinforcement learning for non-stationary environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10689–10695. IEEE.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. *Advances in neural information processing systems*, 31.

Chen, Y., Tong, Z., Zheng, Y., Samuelson, H., and Norford, L. (2020). Transfer learning with deep neural networks for model predictive control of hvac and natural ventilation in smart buildings. *Journal of Cleaner Production*, 254:119866.

Cheng, R., Orosz, G., Murray, R. M., and Burdick, J. W. (2019). End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. (2018). A lyapunov-based approach to safe reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8103–8112.

Chowdhary, G., Mühlegg, M., How, J. P., and Holzapfel, F. (2013). Concurrent learning adaptive model predictive control. In *Advances in Aerospace Guidance, Navigation and Control*, pages 29–47. Springer.

Clavera, I., Nagabandi, A., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. (2019). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*.

Cohn, P., Green, A., Langstaff, M., and Roller, M. (2017). Commercial drones are here: The future of unmanned aerial systems. *McKinsey & Company*.

Cortinovis, G. F., Paiva, J. L., Song, T. W., and Pinto, J. M. (2009a). A systemic approach for optimal cooling tower operation. *Energy Conversion and Management*, 50(9):2200–2209.

Cortinovis, G. F., Ribeiro, M. T., Paiva, J. L., Song, T. W., and Pinto, J. M. (2009b). Integrated analysis of cooling water systems: Modeling and experimental validation. *Applied thermal engineering*, 29(14-15):3124–3131.

Cowling, I. D., Whidborne, J. F., and Cooke, A. K. (2006). Optimal trajectory planning and lqr control for a quadrotor uav. In *International Conference on Control*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Czarnecki, W. M., Pascanu, R., Osindero, S., Jayakumar, S., Swirszcz, G., and Jaderberg, M. (2019). Distilling policy distillation. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1331–1340. PMLR.

Da Silva, F. L. and Costa, A. H. R. (2019). A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703.

Dagan, I., Lee, L., and Pereira, F. (1997). Similarity-based methods for word sense disambiguation. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 56–63.

Dhulipalla, A., Han, N., Hu, H., and Hu, H. (2022). A comparative study to characterize the effects of adverse weathers on the flight performance of an unmanned-aerial-system. In *AIAA AVIATION 2022 Forum*, page 3962.

Doya, K., Samejima, K., Katagiri, K.-i., and Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369.

Du, X., Farrahi, K., and Niranjan, M. (2019). Transfer learning across human activities using a cascade neural network architecture. In *Proceedings of the 23rd international symposium on wearable computers*, pages 35–44.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Dwivedi, K. and Roig, G. (2019). Representation similarity analysis for efficient task taxonomy & transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12387–12396.

Ebeid, E., Skriver, M., and Jin, J. (2017). A survey on open-source flight control platforms of unmanned aerial vehicle. In *2017 euromicro conference on digital system design (dsd)*, pages 396–402. IEEE.

Fachantidis, A., Partalas, I., Taylor, M. E., and Vlahavas, I. (2011). Transfer learning via multiple inter-task mappings. In *European Workshop on Reinforcement Learning*, pages 225–236. Springer.

Fakoor, R., Chaudhari, P., Soatto, S., and Smola, A. J. (2020). Meta-q-learning. In *International Conference on Learning Representations*.

Fei, F., Tu, Z., Xu, D., and Deng, X. (2020). Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7358–7364. IEEE.

Fernandes, K. and Cardoso, J. S. (2019). Hypothesis transfer learning based on structural model similarity. *Neural Computing and Applications*, 31(8):3417–3430.

Fernández, F., García, J., and Veloso, M. (2010). Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871.

Fernández, F. and Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727.

Fernandez-Gauna, B., Lopez-Guede, J. M., and Graña, M. (2013). Transfer learning with partially constrained models: application to reinforcement learning of linked multicomponent robot system control. *Robotics and Autonomous Systems*, 61(7):694–703.

Finn, C., Abbeel, P., and Levine, S. (2017a). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

Finn, C., Abbeel, P., and Levine, S. (2017b). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Fourlas, G. K. and Karras, G. C. (2021). A survey on fault diagnosis and fault-tolerant control methods for unmanned aerial vehicles. *Machines*, 9(9):197.

Frisk, E., Krysander, M., and Jung, D. (2017). A toolbox for analysis and design of model based diagnosis systems for large scale models. *IFAC-PapersOnLine*, 50(1):3287–3293.

Fulton, N. and Platzer, A. (2018). Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030.

Garcia, C. E., Prett, D. M., and Morari, M. (1989). Model predictive control: Theory and practice - a survey. *Automatica*, 25(3):335–348.

Garcia, J. and Fernández, F. (2012). Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45:515–564.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

García, J. and Fernández, F. (2019). Probabilistic policy reuse for safe reinforcement learning. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 13(3):1–24.

Geibel, P. (2001). Reinforcement learning with bounded risk. In *In Proceedings of the Eighteenth International Conference on Machine Learning*. Citeseer.

Goldberg, J., Greenberg, I., and Lawrence, T. F. (1993). Adaptive fault tolerance. *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*.

González-Jorge, H., Martínez-Sánchez, J., Bueno, M., et al. (2017). Unmanned aerial systems for civil applications: A review. *Drones*, 1(1):2.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Goraj, Z. and Zakrzewski, P. (2005). Aircraft fuel systems and their influence on stability margin. *Prace Instytutu Lotnictwa*, pages 29–40.

Grishin, I., Vishnevsky, V., Dinh, T. D., Vybornova, A., and Kirichek, R. (2020). Methods for correcting positions of tethered uavs in adverse weather conditions. In *2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 308–312. IEEE.

Gros, S., Zanon, M., and Bemporad, A. (2020). Safe reinforcement learning via projection on a safe set: How to achieve optimality? *IFAC-PapersOnLine*, 53(2):8076–8081.

Grubinger, T., Chasparis, G. C., and Natschläger, T. (2017). Generalized online transfer learning for climate control in residential buildings. *Energy and Buildings*, 139:63–71.

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. *Advances in Neural Information Processing Systems*, 31:5302–5311.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Hamadi, H., Lussier, B., Fantoni, I., Francis, C., and Shraim, H. (2020). Comparative study of self tuning, adaptive and multiplexing ftc strategies for successive failures in an octorotor uav. *Robotics and Autonomous Systems*, 133:103602.

Helwa, M. K. and Schoellig, A. P. (2017). Multi-robot transfer learning: A dynamical system perspective. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4702–4708. IEEE.

Hentati, A. I., Krichen, L., Fourati, M., and Fourati, L. C. (2018). Simulation tools, environments and frameworks for uav systems performance analysis. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1495–1500. IEEE.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2021). Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Jonathan Ho, O., and Abbeel, P. (2018). Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China. PMLR.

IV, G. P. J., Pearlstine, L. G., and Percival, H. F. (2006). An assessment of small unmanned aerial vehicles for wildlife research. *Wildlife society bulletin*, 34(3):750–758.

Jagannath, J., Jagannath, A., Furman, S., and Gwin, T. (2021). Deep learning and reinforcement learning for autonomous unmanned aerial systems: Roadmap for theory to deployment. In *Deep Learning for Unmanned Systems*, pages 25–82. Springer.

Jiang, J. and Zhang, Y. (2006). Accepting performance degradation in fault-tolerant control system design. *IEEE transactions on control systems technology*, 14(2):284–292.

Jin, G.-Y., Cai, W.-J., Lu, L., Lee, E. L., and Chiang, A. (2007). A simplified modeling of mechanical cooling tower for control and optimization of hvac systems. *Energy conversion and management*, 48(2):355–365.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Junges, S., Jansen, N., Dehnert, C., Topcu, U., and Katoen, J.-P. (2016). Safety-constrained reinforcement learning for mdps. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 130–146. Springer.

Jurdjevic, V. and Quinn, J. P. (1978). Controllability and stability. *Journal of differential equations*, 28(3):381–389.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Koch, W., Mancuso, R., West, R., and Bestavros, A. (2019). Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Kusiak, A. and Xu, G. (2012). Modeling and optimization of hvac systems using a dynamic neural network. *Energy*, 42(1):241–250.

Lampton, A., Valasek, J., and Kumar, M. (2010). Multi-resolution state-space discretization for q-learning with pseudo-randomized discretization. *The 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona*.

Lavretsky, E. and Wise, K. A. (2013). Optimal control and the linear quadratic regulator. In *Robust and Adaptive Control*, pages 27–50. Springer.

Lewis, F. L., Vrabie, D., and Vamvoudakis, K. G. (2012). Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6):76–105.

Li, L., Luo, H., Ding, S. X., Yang, Y., and Peng, K. (2019). Performance-based fault detection and fault-tolerant control for automatic control systems. *Automatica*, 99:308–316.

Li, P., Liu, D., Xia, X., and Baldi, S. (2022a). Embedding adaptive features in the ardupilot control architecture for unmanned aerial vehicles. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 3773–3780. IEEE.

Li, Q. and Xu, Y. (2020). Unmanned aerial vehicle angular velocity control via reinforcement learning in dimension reduced search spaces. In *2020 American Control Conference (ACC)*, pages 4926–4931. IEEE.

Li, S. and Bastani, O. (2020). Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7166–7172. IEEE.

Li, W., Huang, R., Li, J., Liao, Y., Chen, Z., He, G., Yan, R., and Gryllias, K. (2022b). A perspective survey on deep transfer learning for fault diagnosis in industrial scenarios: Theories, applications and challenges. *Mechanical Systems and Signal Processing*, 167:108487.

Liu, L., Wang, Z., and Zhang, H. (2016). Adaptive fault-tolerant tracking control for mimo discrete-time systems via reinforcement learning algorithm with less learning parameters. *IEEE Transactions on Automation Science and Engineering*, 14(1):299–313.

Liu, W. and Li, P. (2019). Disturbance observer-based fault-tolerant adaptive control for nonlinearly parameterized systems. *IEEE Transactions on Industrial Electronics*, 66:8681–8691.

Liu, Y., Li, Z., Liu, H., and Kan, Z. (2020). Skill transfer learning for autonomous robots and human–robot cooperation: A survey. *Robotics and Autonomous Systems*, 128:103515.

Long, M., Cao, Y., Wang, J., and Jordan, M. (2015). Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR.

Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., and Zhang, G. (2015). Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, 80:14–23. 25th anniversary of Knowledge-Based Systems.

Lu, Y., Chen, L., Saidi, A., Dellandrea, E., and Wang, Y. (2017). Discriminative transfer learning using similarities and dissimilarities. *IEEE transactions on neural networks and learning systems*, 29(7):3097–3110.

Luo, G., Yang, Y., Yuan, Y., Chen, Z., and Ainiwaer, A. (2019). Hierarchical transfer learning architecture for low-resource neural machine translation. *IEEE Access*, 7:154157–154166.

Lütjens, B., Everett, M., and How, J. P. (2019). Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE.

Marks, A., Whidborne, J. F., and Yamamoto, I. (2012). Control allocation for fault tolerant control of a vtol octorotor. In *Proceedings of 2012 UKACC International Conference on Control*, pages 357–362. IEEE.

Mas-Colell, A., Whinston, M. D., Green, J. R., et al. (1995). *Microeconomic theory*, volume 1. Oxford university press New York.

Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, pages 664–671.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

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

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. (2018). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*.

Naug, A., Ahmed, I., and Biswas, G. (2019). Online energy management in commercial buildings using deep reinforcement learning. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 249–257. IEEE.

Nelson, J. R., Grubesic, T. H., Wallace, D., and Chamberlain, A. W. (2019). The view from above: A survey of the public's perception of unmanned aerial vehicles and privacy. *Journal of urban technology*, 26(1):83–105.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Noura, H., Sauter, D., Hamelin, F., and Theilliol, D. (2000). Fault-tolerant control in dynamic systems: Application to a winding machine. *IEEE control systems magazine*, 20(1):33–49.

Paleyes, A., Urma, R.-G., and Lawrence, N. D. (2020). Challenges in deploying machine learning: a survey of case studies. *ACM Computing Surveys (CSUR)*.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Park, S., Bae, J., Kim, Y., and Kim, S. (2013). Fault tolerant flight control system for the tilt-rotor uav. *Journal of the Franklin Institute*, 350(9):2535–2559.

Patton, R. J. (1997). Fault-tolerant control systems: The 1997 situation. *IFAC Proceedings*, 30(no. 18).

Paul, S., Kurin, V., and Whiteson, S. (2019). Fast efficient hyperparameter tuning for policy gradient methods. *Advances in Neural Information Processing Systems*, 32.

Pereida, K. and Schoellig, A. P. (2018). Adaptive model predictive control for high-accuracy trajectory tracking in changing conditions. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7831–7837. IEEE.

Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Pi, C.-H., Dai, Y.-W., Hu, K.-C., and Cheng, S. (2021). General purpose low-level reinforcement learning control for multi-axis rotor aerial vehicles. *Sensors*, 21(13):4560.

Pounds, P. E., Bersak, D. R., and Dollar, A. M. (2012). Stability of small-scale uav helicopters and quadrotors with added payload mass under pid control. *Autonomous Robots*, 33(1):129–142.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836. PMLR.

Qin, S. J. (2012). Survey on data-driven industrial process monitoring and diagnosis. *Annual reviews in control*, 36(2):220–234.

Ravichandar, H., Polydoros, A. S., Chernova, S., and Billard, A. (2020). Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330.

Razmi, H. and Afshinfar, S. (2019). Neural network-based adaptive sliding mode control design for position and attitude control of a quadrotor uav. *Aerospace Science and technology*, 91:12–27.

Reyes-Valeria, E., Enriquez-Caldera, R., Camacho-Lara, S., and Guichard, J. (2013). Lqr control for a quadrotor using unit quaternions: Modeling and simulation. In *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, pages 172–178. IEEE.

Ribeiro, M. I. (2004). Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741.

Richards, S., Azizan, N., Slotine, J.-J., and Pavone, M. (2021). Adaptive-control-oriented meta-learning for nonlinear systems. In *Robotics science and systems*.

Rosman, B., Hawasly, M., and Ramamoorthy, S. (2016). Bayesian policy reuse. *Machine Learning*, 104(1):99–127.

Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016). Policy distillation. In *ICLR (Poster)*.

Sadeghzadeh, I., Mehta, A., Chamseddine, A., and Zhang, Y. (2012). Active fault tolerant control of a quadrotor uav based on gainscheduled pid control. In *2012 25th IEEE Canadian conference on electrical and computer engineering (CCECE)*, pages 1–4. IEEE.

Sæmundsson, S., Hofmann, K., and Deisenroth, M. P. (2018). Meta reinforcement learning with latent variable gaussian processes. *arXiv preprint arXiv:1803.07551*.

Saied, M., Lussier, B., Fantoni, I., Francis, C., Shraim, H., and Sanahuja, G. (2015). Fault diagnosis and fault-tolerant control strategy for rotor failure in an octorotor. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 5266–5271. IEEE.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR.

Sayyaadi, H. and Nejatolahi, M. (2011). Multi-objective optimization of a cooling tower assisted vapor compression refrigeration system. *international journal of refrigeration*, 34(1):243–256.

Schaal, S. et al. (1997). Learning from demonstration. *Advances in neural information processing systems*, pages 1040–1046.

Schacht-Rodríguez, R., Ponsart, J.-C., García-Beltrán, C.-D., Astorga-Zaragoza, C.-M., Theilliol, D., and Zhang, Y. (2018). Path planning generation algorithm for a class of uav multirotor based on state of health of lithium polymer battery. *Journal of Intelligent & Robotic Systems*, 91(1):115–131.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schweighofer, N. and Doya, K. (2003). Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9.

Shao, K., Zhu, Y., and Zhao, D. (2019). Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 3(1):73–84.

Shi, G., Shi, X., O'Connell, M., Yu, R., Azizzadenesheli, K., Anandkumar, A., Yue, Y., and Chung, S.-J. (2019). Neural lander: Stable drone landing control using learned dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790. IEEE.

Shui, C., Abbasi, M., Robitaille, L.-É., Wang, B., and Gagné, C. (2019). A principled approach for learning task similarity in multitask learning. *arXiv preprint arXiv:1903.09109*.

Sigaud, O. and Stulp, F. (2019). Policy search in continuous action domains: an overview. *Neural Networks*, 113:28–40.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR.

Smith, G. L., Schmidt, S. F., and McGee, L. A. (1962). *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*, volume 135. National Aeronautics and Space Administration.

Sola, J. and Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science*, 44(3):1464–1468.

Stevens, B. L., Lewis, F. L., and Johnson, E. N. (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons.

Suicmez, E. C. and Kutay, A. T. (2014). Optimal path tracking control of a quadrotor uav. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 115–125. IEEE.

Sussmann, H. J. and Jurdjevic, V. (1972). Controllability of nonlinear systems. *Journal of Differential Equations*, 12.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S. and Barto, A. R. (2017). *Reinforcement learning: An introduction*. Cambridge: MIT Press, 2nd edition. Draft.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Taylor, M. E. and Stone, P. (2007). Representation transfer for reinforcement learning. In *AAAI Fall Symposium: Computational Approaches to Representation Change during Learning and Development*, pages 78–85.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7).

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine learning*, 8(3):257–277.

Thananjeyan, B., Balakrishna, A., Nair, S., Luo, M., Srinivasan, K., Hwang, M., Gonzalez, J. E., Ibarz, J., Finn, C., and Goldberg, K. (2021). Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3):4915–4922.

Tjokro, S. and Shah, S. L. (1985). Adaptive pid control. In *1985 American Control Conference*, pages 1528–1534. IEEE.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.

Vidyadharan, A., Philpott, R., Kwasa, B. J., and Bloebaum, C. L. (2017). Analysis of autonomous unmanned aerial systems based on operational scenarios using value modelling. *Drones*, 1(1):5.

Vu, H. D., Chai, K. S., Keating, B., Tursynbek, N., Xu, B., Yang, K., Yang, X., and Zhang, Z. (2017). Data driven chiller plant energy optimization with domain knowledge. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1309–1317. ACM.

Wachi, A. and Sui, Y. (2020). Safe reinforcement learning in constrained markov decision processes. In *International Conference on Machine Learning*, pages 9797–9806. PMLR.

Wang, B., Mendez, J., Cai, M., and Eaton, E. (2019a). Transfer learning via minimizing the performance gap between domains. *Advances in Neural Information Processing Systems*, 32.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Wang, S. and Ma, Z. (2008). Supervisory and optimal control of building hvac systems: A review. *HVAC&R Research*, 14(1):3–32.

Wang, T., Gao, H., and Qiu, J. (2015). A combined adaptive neural network and nonlinear model predictive control for multi-rate networked industrial process control. *IEEE Transactions on Neural Networks and Learning Systems*, 27:416–425.

Wang, Y., Huang, C., and Zhu, Q. (2020a). Energy-efficient control adaptation with safety guarantees for learning-enabled cyber-physical systems. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE.

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020b). Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34.

Wang, Z., Dai, Z., Póczos, B., and Carbonell, J. (2019b). Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11293–11302.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Wei, X., Xu, G., and Kusiak, A. (2014). Modeling and optimization of a chiller plant. *Energy*, 73:898–907.

Wen, M. and Topcu, U. (2020). Constrained cross-entropy method for safe reinforcement learning. *IEEE Transactions on Automatic Control*.

Xian, B., Zhao, B., Zhang, Y., and Zhang, X. (2017). A low-cost hardware-in-the-loop-simulation testbed of quadrotor uav and implementation of nonlinear control schemes. *Robotica*, 35(3):588–612.

Xu, T., Liang, Y., and Lan, G. (2021). Crpo: A new approach for safe reinforcement learning with convergence guarantee. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11480–11491. PMLR.

Xu, Z., van Hasselt, H., and Silver, D. (2018). Meta-gradient reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 2402–2413.

Yen, G. G. and Ho, L.-W. (2003). Online multiple-model-based fault diagnosis and accommodation. *IEEE transactions on industrial electronics*, 50(2):296–312.

Yogatama, D. and Mann, G. (2014). Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial intelligence and statistics*, pages 1077–1085. PMLR.

Yu, B., Zhang, Y., and Qu, Y. (2015). Mpc-based ftc with fdd against actuator faults of uavs. In *2015 15th International Conference on Control, Automation and Systems (ICCAS)*, pages 225–230. IEEE.

Yu, M., Yang, Z., Kolar, M., and Wang, Z. (2019). Convergent policy optimization for safe reinforcement learning. *Advances in Neural Information Processing Systems*, 32:3127–3139.

Zeghlache, S., Mekki, H., Bouguerra, A., and Djerioui, A. (2018). Actuator fault tolerant control using adaptive rbfnn fuzzy sliding mode controller for coaxial octorotor uav. *ISA transactions*, 80:267–278.

Zhang, K., Yang, Z., and Başar, T. (2021). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384.

Zhang, W., Fang, Y., and Ma, Z. (2017). The effect of task similarity on deep transfer learning. In *International Conference on Neural Information Processing*, pages 256–265. Springer.

Zhang, Y. and Jiang, J. (2003). Fault tolerant control system design with explicit consideration of performance degradation. *IEEE Transactions on Aerospace and Electronic Systems*, 39(3):838–848.

Zhang, Y. and Jiang, J. (2008). Bibliographical review on reconfigurable fault-tolerant control systems. *Annual reviews in control*, 32(2):229–252.

Zhang, Y. and Yang, Q. (2021). A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*.

Zhang, Y. and Yeung, D.-Y. (2010). A convex formulation for learning task relationships in multi-task learning. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 733–742.

Zhao, B., Liu, D., and Li, Y. (2017). Observer based adaptive dynamic programming for fault tolerant control of a class of nonlinear systems. *Information Sciences*, 384:21–33.

Zhou, P., Zou, Y., Yuan, X.-T., Feng, J., Xiong, C., and Hoi, S. (2021). Task similarity aware meta learning: Theory-inspired improvement on maml. In *Uncertainty in Artificial Intelligence*, pages 23–33. PMLR.

Zhu, Z., Lin, K., and Zhou, J. (2020). Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.

Zulu, A., John, S., et al. (2014). A review of control algorithms for autonomous quadrotors. *Open Journal of Applied Sciences*, 4(14):547.