Computationally Efficient Gaussian Process Changepoint Detection and Regression

by

Robert Conlin Grande

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Masters of Science in Aerospace Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author . . . . . .
Department of Aeronautics and Astronautics
May 22, 2014

Signature redacted

Certified by . . . . . .
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by . . . . . .
Paulo C. Lozano
Chair, Graduate Program Committee

Computationally Efficient Gaussian Process Changepoint Detection and Regression

by

Robert Conlin Grande

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2014, in partial fulfillment of the
requirements for the degree of
Masters of Science in Aerospace Engineering

Abstract

Most existing GP regression algorithms assume a single generative model, leading to poor performance when data are nonstationary, i.e. generated from multiple switching processes. Existing methods for GP regression over non-stationary data include clustering and changepoint detection algorithms. However, these methods require significant computation, do not come with provable guarantees on correctness and speed, and most algorithms only work in batch settings. This thesis presents an efficient online GP framework, GP-NBC, that leverages the generalized likelihood ratio test to detect changepoints and learn multiple Gaussian Process models from streaming data. Furthermore, GP-NBC can quickly recognize and reuse previously seen models. The algorithm is shown to be theoretically sample efficient in terms of limiting mistaken predictions. Our empirical results on two real-world datasets and one synthetic dataset show GP-NBC outperforms state of the art methods for nonstationary regression in terms of regression error and computational efficiency.

The second part of the thesis introduces a Reinforcement Learning (RL) algorithm, UCRL-GP-CPD, for multi-task Reinforcement Learning when the reward function is nonstationary. First, a novel algorithm UCRL-GP is introduced for stationary reward functions. Then, UCRL-GP is combined with GP-NBC to create UCRL-GP-CPD, which is an algorithm for nonstationary reward functions. Unlike previous work in the literature, UCRL-GP-CPD does not make distributional assumptions about task generation, does not assume changepoint times are known, and does not assume that all tasks have been experienced a priori in a training phase. It is proven that UCRL-GP-CPD is sample efficient in the stationary case, will detect changepoints in the environment with high probability, and is theoretically guaranteed to prevent negative transfer. UCRL-GP-CPD is demonstrated empirically on a variety of simulated and real domains.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank a number of people throughout my career at MIT. Firstly, I would like to thank my parents and brother for always being there and supporting me through this degree. It has been difficult at times, but I could always take comfort in talking with my family and coming home whenever I needed a break. I would also like to thank my close friends at Johns Hopkins and the JHU AllNighters as well. It could be said that my once a semester trip to Baltimore to see the JHU AllNighters and catch up with friends were some of the best times I've had.

I would also like to thank Aerojet-Rocketdyne for generously having supported me in my time at MIT through the Aerojet Fellowship. I seriously appreciate the opportunity that Aerojet-Rocketdyne has given me, and it has been a pleasure working with James Drake and others at Aerojet-Rocketdyne.

I would also like to thank the academics that I have met at MIT and who have reshaped the way in which I view research and life itself. I have learned a great deal about how to ask the important questions in research and delineate what constitutes great research that will have a lastly impact versus research that is incremental and not likely to have a greater impact. I would like to thank my advisor Jonathan How for teaching me how to properly analyze research and really get to the crux of the problem. I learned how to quickly analyze a field, realize the existing holes in the literature, identify the key challenges, and propose solutions to the meaningful problems. Jon has also taught me a great deal about presenting research, which I am very grateful for. It is not enough to simply do good research. It is equally, if not more, important to be able to communicate to others what it is you have done. One must properly convey why the research is important and why the listener should care. Without properly motivating your research and presenting it such that the listener is keenly aware of the flaws in the existing literature and how your solution fixes these issues, your work, no matter the quality, will fall on deaf ears.

I would like to thank the former post-docs in my lab Girish Chowdhary and Thomas Walsh, who helped me immensely with the creative and technical processes of research. As a master's student, it is often difficult to come up with new research ideas or to know

what has been done in the field already. I found that Girish and Tom were always incredibly knowledgeable about the field and were extremely good at motivating new ideas, problems, and possible solutions. There were enough ideas generated in our conversations to fill several theses. I would also like to thank a large set of members of the lab past and present who I have collaborated with, received advice from, or with whom I drank a few beers at the Muddy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis examines two subproblems within the larger field of algorithms operating in a nonstationary stochastic world. In the first problem, nonstationary prediction and model learning, the algorithm passively receives new samples from the environment, and must return accurate predictions about some latent function. In the second problem, reinforcement learning with nonstationary reward models, an active agent may take actions to explore the environment and exploit maximal reward. Since the agent may select actions dynamically, the problem is not to just make accurate predictions, but also to plan under uncertainty.

1.1 Background

Many prediction, decision making, and control algorithms adapt to uncertainty by learning online a model of the environment from noisy observations. Gaussian Processes (GPs) have emerged as a widely applied framework for inference in many machine learning and decision making applications, including regression, [102], classification, [85], adaptive control, [26], and reinforcement learning, [38]. However, most existing (online) GP inference algorithms assume that the underlying generative model is stationary, that is, the data generating process is time-invariant. Yet, many online learning tasks, from predicting stock market prices to controlling aircraft, may involve *changepoints*: points in time in when abrupt changes occur (for example changes in market demand or actuator failure) and affect the generating stochastic process itself. When the data contains changepoints, i.e., is

non-stationary[1], an online learning and prediction algorithm, such as a GP, needs to be augmented to quickly detect changepoints, learn new models online, and reuse old models if they become applicable again. Furthermore, it is useful for such an algorithm to have theoretical guarantees on sample complexity and accuracy, so its performance can be reliably characterized for applications requiring learning rates and safety guarantees.

One way of dealing with non-stationarity is to add time, or another counting mechanism, to the GP kernel [26, 97]. However, this technique forgets useful information from earlier phases of learning if no changepoints occur, and cannot make use of previously learned models that may reappear and become applicable again. The latter is important in many domains in which models may be revisited, such as pedestrian tracking [6] and robotic ping pong [126]. Online GP changepoint detection algorithms have been proposed previously [47, 106]. However, these algorithms come with no theoretical guarantees on accuracy and cannot reuse models after changepoints. Additionally, as shown in the experiments section, Section 3.3, these algorithms require significant computation and are therefore ill-suited for applications requiring real-time prediction.

The first portion of this thesis presents a novel algorithm for online regression over data with changepoints, the Gaussian Process Non-Bayesian Clustering (GP-NBC) algorithm. GP-NBC decouples the problems of changepoint detection, regression, and model reuse, resulting in efficient online learning in the presence of changepoints. GP-NBC uses a non-Bayesian test based on a Generalized Likelihood Ratio (GLR) test [7] to detect changepoints and to re-identify previously seen functions, and performs efficient inference using a sparse online GP inference algorithm [34].

GP-NBC is computationally efficient and shown to be orders of magnitude faster than existing methods, which enables the real-time computation needed in decision making and control domains. In addition, GP-NBC is able to reuse previously learned models, which is in contrast to methods that use time or counters in the kernel function. Furthermore, this thesis derives polynomial sample complexity bounds on the number of inaccurate predictions by GP-NBC, a property not shared by competing algorithms. GP-NBC is validated on two real-world and two synthetic datasets in Section 3.3, where it is shown to outper-

[1]Here "non-stationary" refers to data with abrupt changes and not to spatially nonstationary GP kernels

form state of the art hierarchical GP and changepoint detection algorithms by orders of magnitude in computational efficiency and prediction accuracy.

The second portion of this thesis presentes an algorithm for reinforcement learning over nonstationary reward functions. Within this framework, the thesis first introduces a novel algorithm for stationary reward functions in continuous state spaces, the Upper Confidence Bound Reinforcement Learning using Gaussian Processes (UCRL-GP) algorithm. Then, this algorithm is modified to detect changepoints using GP-NBC, resulting in the full algorithm, Upper Confidence Bound Reinforcement Learning using Gaussian Processes with Changepoint Detection (UCRL-GP-CPD). UCRL-GP-CPD uses UCRL-GP to explore the state space and learn an optimal policy in between changepoints, and uses GP-NBC to detect changepoints as they occur.

Reinforcement Learning (RL) [119] is a widely studied framework in which an agent interacts with an environment and receives rewards for performing actions in various states. Through exploring the environment, the agent learns to optimize its actions to maximize the expected return of discounted rewards in the future. As is common in the literature, the RL environment is modeled as a Markov Decision Process [100], in which each action taken by the agent results in a stochastic reward as well as a stochastic transition which is dependent on the current state and action. Reinforcement Learning and Markov Decision Processes are discussed in further detail in Section 2.4.

In single-task RL, the agent must learn to perform some task or policy for a single, stationary reward function. However, in order for robots or other autonomous agents to learn so called *life long learning*, this requires an agent to learn multiple distinct tasks, detect when changes have occurred in the environment, and transfer past model knowledge when it becomes applicable again. Therefore, this thesis also considers reward functions that can switch discretely in time at changepoints and may reoccur. In this case, the agent must also identify when the environment has changed as well as determine if the current environment is similar to previously learned tasks. If the agent re-encounters a previous environment, the agent should ideally re-identify the task and transfer past experience to speed up learning and improve performance.

Most multi-task reinforcement solutions that have been proposed previously require

that all tasks be encountered in a training phase [14, 123, 129] or require a distributional assumption about how the MDPs are generated [129]. These algorithms also assume that changes may only occur in between episodes, requiring that the changepoint time is known to the agent. These assumptions are overly restrictive in many domains such as pedestrian avoidance [6] and strategic games [126]. UCRL-GP-CPD makes no such assumptions about the distribution of tasks and does not require the changepoint to be known by the agent. Rather, it can detect changes in the environment by making inferences from changes in observations.

Lastly, many algorithms [123, 129] come without theoretical guarantees on sample complexity. That is, given any MDP, the algorithm is guaranteed to find an ϵ-optimal policy with high probability, $1 - \delta$, in a polynomial number of suboptimal, or exploratory steps. Many of these algorithms also do not come with theoretical guarantees on preventing negative transfer, that is, transferring an incorrect training task to the current test task.

It is proven that UCRL-GP is PAC-MDP (Probably Approximately Correct in MDPs), one of the first RL algorithms for continuous domains which has sample complexity guarantees. It is also proven that UCRL-GP-CPD is theoretically guaranteed to detect changes in the environment and prevent negative transfer.

UCRL-GP is demonstrated on several empirical domains and outperforms the state of the art algorithms in terms of learning the optimal policy in a the fewest number of samples. Empirically, UCRL-GP-CPD is also demonstrated on several simulated and real domains. It is shown that UCRL-GP-CPD effectively learns the optimal policy, detects changes in the environment, and successfully identifies and transfers old models when they become applicable.

1.2 Problem Statement

In this section, the problem statements for online prediction for nonstationary data and reinforcement learning with nonstationary environments are defined.

1.2.1 Online Non-Stationary Prediction

On each step t of an *online non-stationary prediction problem*, a learning agent receives an input $x_t \in U \subset \mathbb{R}^d$ that may be chosen adversarially (i.e., not chosen i.i.d. from a distribution). The agent makes a prediction $\hat{\mu}(x_t)$ of the expected value of the distribution $\mathbb{E}_{p_i(\cdot|x)}[y \mid x_t] = f_i(x_t)$ with generating distribution $p_i(y \mid x) \in \mathcal{P}$ that changes between changepoints. The agent then observes an output drawn from the generative distribution $y_t \sim p_i(y \mid x_t)$. The goal is to limit the number of *mistakes* in agent predictions for some tolerance ϵ_E. More formally, the goal is to bound the number of timesteps where $|f_i(x_t) - \hat{\mu}(x_t)| > \epsilon_E$.

Nonstationarity is introduced when mean of the underlying process changes from $f_i(x)$ to some $f_j(x)$ at unknown *changepoints*. Specifically, if timestep t is a changepoint, then for some region U': $\forall x \in U'$, $|f_i(x) - f_j(x)| > \epsilon_E$. The period in between changepoints is called a *phase* and a learning agent's goal in this scenario is to limit the number of mistakes in each phase. This requires the agent to detect each changepoint and potentially re-use previously learned functions. To further solidify the problem, it is assumed there is a lower bound on the number of samples between changepoints. Section 3.2 provides sufficient conditions regarding this lower bound and the realizability of each f_t (i.e., how well a GP captures f_i) that lead to a bounded number of mistakes during each phase.

In the stationary analysis Section 3.2.1, no assumptions are made about the structure of the distribution, other than it is Lipschitz. In the nonstationary analysis section, additional assumptions are made about the class of functions $\mathcal{F}$ that may be detected using the nonstationary algorithm. These are detailed in the preliminaries Section 2.2.

1.2.2 Reinforcement Learning

A RL environment is modeled as a Markov Decision Process [99] (MDP). A MDP M is a tuple $\langle S, A, R, T, \gamma \rangle$ where S is the potentially infinite set of states, A is a finite set of actions available to the agent, and $0 \leq \gamma < 1$ is a discount factor for the sums of rewards. Let $\mathcal{P}_X$ denote the set of probability distributions over the space X. Then, the reward distribution $R(s, a) : S \times A \mapsto \mathcal{P}_{\mathcal{R}}$ is a distribution over rewards conditioned on the current

state and action, with bounded output, $R(s,a) \in [R_{\min}, R_{\max}]$. The transition function is a distribution from the current state and action to a new state s', $T(s,a) : S \times A \mapsto \mathcal{P}_S$. This thesis only considers the case in which the transition distribution is deterministic and Lipschitz with respect to the states. Furthermore, the expectation of the reward function $\bar{r}(s,a) = \mathbb{E}[R(s,a)]$ is assumed Lipschitz with respect to the states.

At the beginning of each episode, the agent starts in some initial state s_0. At each time step, the agent observes its current state and must choose an action $a \in A$. It receives a reward $r(s,a) \sim R(s,a)$ and transitions to a new state $s' \sim T(s,a)$. The goal of the agent is to obtain the largest possible reward possible over the episode. A *policy* is a mapping from states to actions that a agent uses to determine which action to take in each state. For any policy π, the value function, $V_M^\pi(s) = \mathbb{E}_\pi[\sum_{j=1}^{\infty} \gamma^{j-1} r_j \mid s]$ is the expected discounted infinite-horizon return of rewards following some policy π on MDP M. The Q-function is defined as $Q_M^\pi(s,a) = \mathbb{E}_\pi[\sum_{j=1}^{\infty} \gamma^{j-1} r_j \mid s,a]$ and returns the expected value function after taking an action a in state s, and then following policy π. Throughout the analysis, the notation $Q_t(s,a)$ will be used to denote the agents estimate of the Q-function at timestep t. Every MDP has an optimal value function $Q^*(s,a) = \bar{r}(s,a) + \gamma \int_{s'} T(s' \mid s,a) V^*(s') ds'$ where $V^*(s) = \max_a Q^*(s,a)$ and corresponding optimal policy $\pi^* : S \mapsto A$. Note the bounded reward means $V^* \in [V_{\min}, V_{\max}]$. In the analysis, it is assumes that $T(s,a)$ is a deterministic function, although in practice the algorithm performs well even in stochastic domains.

In RL, an agent is given S, A, and γ and then acts in M with the goal of learning π^*. For value-based methods (as opposed to policy search [63]), there are roughly two classes of RL algorithms: model-based and model-free. Model-based algorithms, such as KWIK-Rmax [77], build models of $T(s,a)$ and $R(s,a)$ and then use a planner to find $Q^*(s,a)$. Many model-based approaches have *sample efficiency* guarantees, that is bounds on the amount of *exploration* they perform.

This thesis also considers reward functions that may switch at changepoints. On each step t of an non-stationary reinforcement learning problem, a learning agent receives an observation from the reward function $r_t \sim R_i(s,a)$ with the reward function $R_i(s,a) \in \mathcal{R}$ and associated expected value $\bar{r}_i(s,a)$. $\mathcal{R}$ is a class of reward functions that satisfies

certain assumptions of seperability and well-behavedness (See Section 2.2). At unknown changepoints, the mean of the underlying process changes from $\bar{r}_i(s,a)$ to some $\bar{r}_j(s,a)$. Specifically, if timestep t is a changepoint, then for some region, U', which is sufficiently large, there exists a *model gap* Γ such that: $\forall(s,a) \in S \times A$, $|\bar{r}_i(s,a) - \bar{r}_j(s,a)| > \Gamma$. The size of this region is established in Section 2.4.1.

The goal of the agent is still to maximize the expected infinite sum of discounted rewards *given* the current reward function. Although the reward function may change in the future, it is assumed that sufficiently large number of actions may be taken between changepoints, so the goal of maximizing the expected return of returns over the finite horizon can be approximated using an infinite sum. Furthermore, since no distributional structure is assumed about the sequence of MDPs, the problem of maximizing the finite sum of rewards over each MDP is ill-defined.

In the problem formulation, the goal of the agent is to maximize the expected return of rewards, which requires maintaining an accurate estimate of the current reward function. Thus, while correctly identifying changepoints and transferring models improves the accuracy of the reward function, the goal is not to correctly identify the MDP. GP-NBC and UCRL-GP-CPD provide guarantees that if a changepoint occurs in the environment, the algorithm will either detect a changepoint, or will be able to successfully learn the changed portion of the state space. In this way, if a change occurs, but predictions are still accurate, it is not important for the sake of maximizing rewards.

1.3 Literature Review

This section reviews relevant literature in the fields of changepoint detection and reinforcement learning.

1.3.1 Changepoint Detection

In the GP literature, algorithms for non-stationary data with changepoints can be broadly classified into two categories: changepoint detection (CPD) algorithms and hierarchical Bayesian clustering algorithms. First, general CPD algorithms are reviewed followed by

a focused review of CPD algorithms which use GPs. Then, hierarchical GP-clustering algorithms are reviewed.

Online changepoint detection has been studied extensively in the context of time series analysis [7]. There, a set of random variables is sampled, often at uniform time intervals, and an algorithm infers points in time where the generative parameters of the time series change. Several results for fast online CPD have been proposed for functions that slowly drift [124], but this thesis considers time series with abrupt changepoints. For abrupt changes, many CPD algorithms do not use or learn models. CPD algorithms with models have been proposed *given* a set of possible models [84], but this thesis considers the case where new models may need to be learned online.

Approaches to CPD and time series monitoring using GPs include GP-changepoint detection algorithm (GP-CPD) [106] based on the Bayesian online changepoint detection (BOCPD) algorithm [1] and using special kernels which include changepoint times [47]. GP-CPD creates a GP model for every possible run length, and performs Bayesian inference to get a distribution over changepoint times. Garnett et al. [47] include the changepoint time in the kernel function and uses optimization to determine the changepoint time. In each case, the focus is not on model learning but rather on instantaneous regression. Additionally, these approaches require significant computation at each instant and the complexity grows as the number of data points increase. The experiments in Section 3.3 demonstrate that wrapping such CPD approaches around GP regression creates a computational burden that is outperformed by GP-NBC and that is not suitable for real-time applications. Additionally, these approaches come with no theoretical guarantees on regression accuracy.

On the other hand, algorithms that learn hierarchical GP-mixture models largely deal with changepoints using BNP approaches, such as the Infinite Mixture of Experts (referred to in this thesis as DP-GP) [105] and the algorithm proposed in [113], which is referred to as MCMC-CRP in this thesis. DP-GP uses a Dirichlet prior and Gibbs sampling over individual data points to obtain the posterior distribution over model assignments and the number of models. MCMC-CRP uses a similar sampling procedure, except the distribution is over changepoints. In this case, the Gibbs sampler creates, removes, and shifts changepoints instead of individual data points. These methods require batch data and the

experiments in Section 3.3 show that these Bayesian inference methods fail to achieve the required computational efficiency for many real-time prediction applications. Additionally, the Dirichlet prior may not converge to the correct number of models [83] and comes with no guarantees on regression accuracy or rates of convergence to the true posterior. While variational inference [8, 13] can reduce computation, these approaches require a variational approximation of the distribution, which do not exist for GPs, to the author's knowledge.

In contrast to these methods, GP-NBC has been designed to work with online decision making algorithms; the theoretical results provide bounds on the number of inaccurate predictions GP-NBC may make per phase and the empirical results show that GP-NBC is orders of magnitude faster than existing methods. In contrast to the sample efficiency results for GPs in the optimization context [41], the bounds given here are designed specifically for online prediction with non-stationary data.

1.3.2 Reinforcement Learning

In this section, relevant work in Reinforcement Learning (RL) is reviewed. Related algorithms in RL for stationary reward functions are reviewed, followed by a review of algorithms and frameworks in RL for multi-task and non-stationary MDPs.

Single Task Reinforcement Learning

In this thesis, a novel algorithm for continuous state spaces and discrete actions is introduced, UCRL-GP, which uses GPs to model reward and transition functions, and it is proven that UCRL-GP is sample efficient, i.e. PAC-MDP.

GPs have been studied in both model-free and model-based RL. In model-free RL, GP-SARSA [43] has been used to model a value function and heuristically extended for better empirical exploration (iGP-SARSA) [32]. However, I prove in [52] that these algorithms may require an exponential (in $\frac{1}{1-\gamma}$) number of samples to reach optimal behavior since they only use a single GP to model the value function. This work is reproduced in Appendix C. More recently, model-free DGPQ [52] was proven to be PAC-MDP. This algorithm generalizes discrete Delayed Q-Learning [115] to continuous state spaces using a GP to

model the Q-function and determine regions of high accuracy.

In model-based RL, the PILCO algorithm trained GPs to represent T, and then derived policies using policy search [37]. However, PILCO does not include a provably efficient (PAC-MDP) exploration strategy. GP-Rmax [62] does include an exploration strategy, specifically replacing areas of low confidence in the (T and R) GPs with high valued states, but no theoretical results are given in that paper. In [52], I prove that this algorithm is actually PAC-MDP. It is shown in Section 4.2.1 that UCRL-GP actually has the same worst case complexity as GP-Rmax, however, in practice, UCRL-GP outperforms GP-Rmax since it uses a more nuaced notion of uncertainty in its planning phase. Instead of replacing regions of high uncertainty with maximal reward and values, UCRL-GP uses an upper probabilistic bound on the estimate of the reward.

A similar analogy between GP-Rmax and UCRL-GP exists in discrete RL as well. In particular, discrete R-max [12] only uses information about a state action pair after sufficient samples have been observed. Thus, until that number of samples is collected, the algorithm ignores all previous information at that state action pair. On the other hand, the UCRL [5] and MBIE [116] algorithms maintain optimistic upper bounds on the reward function and error bounds on the empirical transition probability function, using the information collected in the state action pair already. Using this information, the UCRL and MBIE algorithms are able to achieve optimal regret [5]. Similarly, GP-Rmax only uses information about a state action pair after the confidence of the GP is sufficiently high, i.e. the variance is sufficiently low, whereas UCRL-GP maintains an optimistic upper bound on the reward function, thus allowing for a less conservative approach to planning. UCRLC [92] is also an algorithm for continuous RL that maintains an optimistic upper bound on the reward, but this algorithm relies on state space discretization, and requires a specific planner that may not be tractable in practice.

In order to extend the UCRL algorithms to continuous spaces without discretization, a notion of function approximator (FA) confidence is required even when the underlying distribution is unknown. This is a non-trivial problem, and this idea has been studied previously in the Knows What it Knows (KWIK) framework [78]. In particular, if a FA is KWIK-learnable, then the FA can determine whether the estimate is within some defined

ϵ with high probability. While many FAs have been shown to be KWIK learnable, these are restricted to discrete or parameterized FAs. In order to represent a general function whose structure is unknown, a nonparametric FA is required which is KWIK-learnable. Thus, the contribution of this work is proving that GPs, a nonparametric FA, provide such a notion of predictive confidence. The full proof that GPs are KWIK-learnable are available in Appendix C.

Lastly, the C-PACE algorithm [95] has already been shown to be PAC-MDP in the continuous setting, though it does not use a GP representation. C-PACE stores data points that do not have close-enough neighbors to be considered "known". When it adds a new data point, the Q-values of each point are calculated by a value-iteration like operation.

Multi-task Reinforcement Learning

The problem of learning multiple tasks has been studied under a number of different names and fields, including multi-task reinforcement learning, inverse reinforcement learning, transfer learning, and non-stationary MDPs. As opposed to learning each MDP from scratch at the beginning of every episode, multi-task algorithms aim to leverage past knowledge and *transfer* past model knowledge to increase learning speeds in the current task. While it was widely believed that transfer learning could improve the learning speed of algorithms in RL, it was not shown theoretically until recently [14]. Transferring past model knowledge can greatly improve the speed of learning, however, a significant challenge in multi-task reinforcement learning is to determine when it is appropriate to transfer past knowledge. In cases where a task is incorrectly labeled, an algorithm may transfer knowledge from a different task to the current task, resulting in extremely poor performance. This phenomena is known as *negative transfer*. While some algorithms provide guarantees against negative transfer [3, 14, 74], others offer no such guarantees [122, 129].

In multi-task reinforcement learning, one of the more common frameworks is the hierarchical Bayesian RL framework [129]. In this framework, tasks are drawn i.i.d. from a distribution over MDPs at the beginning of each episode, and parameters are drawn from a distribution conditioned on the drawn task of the MDP drawn. In the case that the true number of tasks is unknown, a Dirichlet prior can be used [129], however, it has been

shown that the Dirichlet prior is not a consistent estimator of the number of tasks [83].

While Bayesian methods have had a variety of success in RL [98], Bayesian methods do not offer any theoretical guarantees on performance or against negative transfer. Indeed, if the prior distribution is significantly different than the true, or posterior distribution, of tasks, the algorithm will perform poorly. Additionally, this framework requires the following two restrictive assumptions: tasks are drawn i.i.d. and the changes only occur at known times. For many real-life tasks, the tasks are not drawn i.i.d. and changes may occur at arbitrary times unknown to the agent. For example, in pedestrian avoidance, the goal position of a pedestrian is highly correlated with other pedestrians and with previous goals selected by the pedestrian [6]. Additionally, the pedestrian may change his/her intent at any time within the trajectory [45]. In robotic ping pong [69], where the person aims the ball is likely to be highly correlated with past strategies. Lastly, these approaches require a costly sampling procedure using Gibbs sampling to obtain the posterior. In many real life domains which require real-time decision making, this sampling procedure is prohibitive. BOSS [3] is an algorithm that is based on the hierarchical Bayesian RL framework, but only performs Gibbs sampling at specific points in time in which a new state becomes known. BOSS comes with some guarantees on accuracy, however, BOSS still requires a distributional assumption and cannot accomodate changepoints.

In [14], an algorithm for discrete multi-task RL is presented which has theoretical guarantees on correctly identifying the number of tasks and correctly transferring model knowledge. However, this algorithm requires an extensive training phase in which all tasks are encountered and sufficient exploration is allowed. This is a restrictive assumption in domains in which all the tasks may be unavailable a priori for training or in which the number of tasks is unknown. For example, in dynamic soaring, every type of wind field may not be known. In pedestrian avoidance, every pedestrian trajectory and goal pattern may not be available or known a priori.

In these works, it is assumed that changes occur only in between episodes. Therefore, the changepoint times are known, so no changepoint detection algorithm is required. Algorithms do exist which perform changepoint detection using hidden Markov models [70] and by formulating the problem as a partially observed MDP (POMDP) [6], but these algo-

rithms assume that the models are known a priori. In many applications, however, the time of the changepoint is unknown to the agent or all the models may not be available a priori.

Unlike these approaches, UCRL-GP-CPD does not make any distributional assumptions, and is capable of detecting changes at arbitrary points in time. Unlike Bayesian methods, it has PAC-MDP guarantees on learning the current reward function efficiently, as well as guarantees on preventing negative transfer. The analog is that in contrast to Bayesian methods, UCRL-GP-CPD provides a framework for multi-task learning in a nonBayesian, or frequentist, framework. Bayesian methods work well when a prior over possible tasks is available and tasks are drawn i.i.d. However, in cases in which a prior is not available, or in which tasks are not drawn i.i.d., a frequentist perspective is better suited. In order to facilitate a frequentist approach to multi-task RL, extra machinery is required to detect changes that may occur at any time. In particular, if a change occurs in a location of the environment that the agent has not explored recently, the agent will be unable to detect this change. In order to maintain an up to date representation of the environment, the agent must periodically explore the environment. This idea was first introduced in [2] as a *fog of war*. In that paper, a naïve exploration strategy was proposed which utilized a variance based exploration policy. In this thesis, a new periodic exploration policy is proposed that is guaranteed to discover a change in the environment if one has occurred.

Lastly, in inverse reinforcement learning (IRL), the agent attempts to learn a task through demonstration by an expert. In the case that the task contains multiple subtasks, many algorithms [70, 82] exist to automatically decompose a task into the relevant subtasks, or segments. While there are many technical parallels between decomposing a task and changepoint detection for multi-task RL, the focus of IRL is fundamentally different than that of multi-task RL. In particular, IRL is focused primarily on creating a reward function which adequately describes the observed behavior of an expert. In this case, the agent does not perform active exploration, but rather focuses on passive observation. In multi-task RL, the agent has direct access to the true reward function and attempts to learn an optimal policy through exploration with the domain.

1.4 Thesis Contributions

The main contributions of this thesis are as follows. In Chapter 3, a novel algorithm, GP-NBC, is introduced for changepoint detection and model reuse using Gaussian processes. It is proven that, unlike previous algorithms for regression over data with changepoints using GPs, GP-NBC has provable guarantees on the number of errorenous predictions. In addition, GP-NBC is able to reuse previously learned models, which is in contrast to methods that use time or counters in the kernel function. It is demonstrated empirically that GP-NBC is orders of magnitude faster than state of the art competing algorithms without losing accuracy. These results are currently under review at the Uncertainty in Artificial Intelligence Conference (UAI 2014) [51].

In Section 4.1, a novel algorithm, UCRL-GP, is presented for single-task RL. It is proven that UCRL is PAC-MDP, one of the first algorithms for continuous RL ever proven to be PAC-MDP. Empirically, it is shown that UCRL outperforms state of the art RL algorithms in terms of learning the optimal policy in the smallest number of samples. In Section 4.1, an algorithm for multi-task RL is presented, UCRL-GP-CPD. This algorithm uses UCRL-GP to explore the domain in between changepoints, and uses GP-NBC to detect changes in the environment. A new periodic exploration algorithm is presented to maintain an up to date representation of the reward function. It is proven that when coupled with the periodic exploration algorithm, GP-NBC will detect changes in the reward function with high probability. It is also proven that using a modified version of GP-NBC model transfer algorithm, UCRL-GP-CPD will not transfer incorrect models, i.e. there is no negative transfer. UCRL-GP-CPD is empirically demonstrated on a variety of simulated and real domains to detect changes in the environment as well as transfer past models correctly.

Lastly, the appendices of this thesis include work for two other projects which I worked on during my Masters. These include my work on Gaussian Processes in adaptive control [49, 50] in Appendix B, as well as the first sample complexity results ever for a model-free continuous RL algorithm, DGPQ [52], which is presented in Appendix C.

Chapter 2

Preliminaries

This section discusses preliminaries for Gaussian processes, nonstationary prediction, reinforcement learning, large number asymptotic theory, and sample complexity definitions.

2.1 Gaussian Processes

Each underlying mean function $f_i(x)$ is modeled as a Gaussian Process (GP). A GP is defined as a collection of random variables, any finite subset of which has a joint Gaussian distribution [102] with mean $\hat{\mu}(x)$ and covariance $k(x, x')$. The experiments section uses the Radial Basis Function, $k(x, x') = A\exp(-\frac{\|x-x'\|^2}{2\sigma})$. This thesis assumes that $f(x) \in \mathbb{R}$. Multidimensional extensions are straightforward in that each dimension is represented by a separate GP. See [102] for a complete analysis of the properties of GPs.

As is common in the GP literature, it is assumed that the GP has a zero mean. In general, this assumption is not limiting since the posterior estimate of the latent function is not restricted to zero. The elements of the GP kernel matrix $K(X, X)$ are defined as $K_{i,j} = k(x_i, x_j)$, and $k(X, x_{i+1}) \in \mathbb{R}^i$ denotes the kernel vector corresponding to the $i + 1^{th}$ measurement. The joint distribution is given by

$$\begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(Z,Z) + \omega_n^2 I & k(Z, z_{i+1}) \\ k^T(Z, z_{i+1}) & k(z_{i+1}, z_{i+1}) \end{bmatrix}\right). \tag{2.1}$$

The conditional probability can then be calculated as a normal variable [102] with mean

$$\hat{\mu}(x_{i+1}) = \alpha^T k(X, x_{i+1}), \tag{2.2}$$

where $\alpha = [K(X,X) + \omega_n^2 I]^{-1} y$ are the kernel weights, and covariance

$$\Sigma(x_{i+1}) = k(x_{i+1}, x_{i+1}) - k^T(X, x_{i+1})[K(X,X) + \omega_n^2 I]^{-1} k(X, x_{i+1}).$$

Due to Mercer's theorem and the addition of the positive definite matrix $\omega_n^2 I$, the matrix inversion in (B.12) and (B.13) is well defined.

2.1.1 Online Budgeted Inference

In general, traditional GP inference does not scale well with the number of data points because it requires that a kernel center be added for every data point seen [104]. Hence, in order to enable efficient GP inference online on resource constrained platforms, modifications must be made that can ensure online tractability. The matrix inversion required for prediction scales with the number of data points n as $O(n^3)$. Csato [34] provides recursive, rank 1 updates for the weights α and covariance (B.13). However, even with rank-1 updates, prediction computation grows as $O(n)$, where n is the number of kernel points. Instead of creating a kernel at every data point, a restricted set of points referred to as the *active basis vector set* is used. Furthermore, the size of this set is limited by a user selectable "budget". When the number of points stored exceeds the budget, new points can only be added by removing an existing point in a way that further enriches the active basis set. In order to determine the novelty of a new point, a linear independence test is performed [34] as,

$$\gamma = k^* - k(BV, x_{i+1})^T \alpha. \tag{2.3}$$

If γ exceeds some threshold ϵ_{tol}, then the point is added to the data set. Otherwise, the weights α and covariance Σ are updated using a rank-1 update, but do not increase in dimension. This sparse representation requires $O(|\mathcal{BV}|)$ calculations for prediction and

$O(|\mathcal{BV}|^2)$ for variance. If the budget is exceeded, a basis vector element must be removed prior to adding another element. For this purpose, the KL divergence based scheme presented by Csato [34] is used. This scheme efficiently approximates the KL divergence between the current GP and the $(t+1)$ alternative GPs missing one data point each, then removes the data point with the lowest score. The online sparse GP algorithm allocates new basis vector locations dynamically, thus preventing ad-hoc a priori feature selection as is common with other methods such as neural networks [107]. Hyperparameters can be optimized online as well [50, 106].

2.2 Nonstationary Prediction and Learning

In the stationary analysis Section 4.2.1, no assumptions are made about the structure of the distribution, other than it is Lipschitz. In the nonstationary analysis Section 4.2.2, additional assumptions are made about the class of functions $\mathcal{F}$ that may be detected using the nonstationary algorithm: *realizability*, *separability*, and *Gaussianity*; it is assumed each mean function $f_i \in \mathcal{F}$ can be exactly modeled by a GP given infinite data and the expected approximation loss of using a GP model with finite data S is approximately the same over all functions, functions are well separated regionally, and the distributions are Gaussian, although possibly heteroskedastic.

Assumption 2.1 (Realizability) The approximation error between a GP model learned from a subset of the data and the true generative distribution does not change between phases by more than ϵ_{DS},

$$\sup_{i,j} |D(p_i \| GP_S) - D(p_j \| GP_S)| \leq \epsilon_{DS} \tag{2.4}$$

Assumption 2.2 (Separability) All functions $f \in \mathcal{F}$ differ by some ϵ_F over at least one compact input region $\tilde{U}$:

$$\forall i, j : \exists x \in \tilde{U} \subset U \text{ s.t. } |f_i(x) - f_j(x)| \geq \epsilon_F > \epsilon_E \tag{2.5}$$

Assumption 2.3 (Gaussianity) $\forall i$, $p_i(y_i \mid x) \sim \mathcal{N}(f_i(x), \omega_i^2(x))$ with nonzero variance $\inf_x \omega_i^2(x) \geq \omega_{min}^2 > 0$ and Lipschitz constant K associated with $f_i(x)$.

Assumption 2.3 ensures well-behavedness of the KL-divergence, since the KL-divergence between a deterministic function and a probabilistic distribution may be ill-defined.

Lastly, auxiliary definitions are used in the sample complexity analysis of GP-NBC and UCRL-GP-CPD: the covering number (adapted from [95]) and the equivalent distance map. The covering number quantifies the notion of how large a continuous space is (according to a distance metric) to a discrete number. The equivalent distance is a mapping between the Euclidean distance between points and the distance in the reproducing kernel Hilbert space used by GPs.

Definition 2.1 The **Covering Number** $\mathcal{N}_U(r)$ of a compact domain $U \subseteq \mathbb{R}^n$ is the cardinality of the minimal set $C = \{c_i, \ldots, c_{N_c}\}$ s.t. $\forall x \in U$, $\exists c \in C$ s.t. $d(x, c) \leq r$, where $d(\cdot, \cdot)$ is some distance metric. The minimal set is referred to as the **Covering Set**.

Definition 2.2 The **equivalent distance** $r(\epsilon_{tol})$ is the maximal distance s.t. $\forall x, c \in U$, if $d(x, c) \leq r(\epsilon_{tol})$, then the linear independence test $\gamma(x, c) = k(x, x) - \frac{k(x,c)^2}{k(c,c)} \leq \epsilon_{tol}$.

2.3 Information and Decision Theory

This section discusses decision theory for probabilistic classification as well as results from large numbers asymptotic theory. These theorems underpin many of the theoretical results of GP-NBC.

2.3.1 Decision Theory

In this thesis, a hypothesis will refer to the proposal that a set of points is generated from a given model. For two models, H_1 and H_0 with known priors, the decision rule that maximizes the probability of correct model identification reduces to a likelihood ratio test

(LRT):

$$\frac{P(y \mid H_1)}{P(y \mid H_0)} \underset{\hat{H}=H_0}{\overset{\hat{H}=H_1}{\gtrless}} \exp(\eta) \tag{2.6}$$

where $\eta = \log((1-p_1)/p_1)$, and $p_1 = p(H_1)$. If the left hand side is greater than η, then $\hat{H} = H_1$ is chosen, otherwise $\hat{H} = H_0$.

When a prior is not available, the problem is formulated as a *nonBayesian* hypothesis test based on the probability of detecting an event H_1 and the probability of a false alarm. The decision rule that maximizes the probability of detection subject to some maximum probability of a false hypothesis choice is still a LRT by the Neyman-Pearson lemma [90].

A variation of the Generalized Likelihood Ratio (GLR) [7], an instance of non-Bayesian hypothesis testing, is utilized to perform changepoint detection. The GLR detects changes by comparing a windowed subset S of data to a null hypothesis. At each step, a new distribution $p(y \mid H_1(S))$ of the same family as the null hypothesis is created with the maximum likelihood statistics from S. The likelihood of the windowed set is taken with respect to both distributions, and if the LRT exceeds some ratio, a changepoint is detected. The joint log likelihood of a subset of points given a GP model is

$$\log P(y \mid x, \Theta) = -\frac{1}{2}(y-\mu(x))^T(\Sigma_{xx}+\omega_n^2 I)^{-1}(y-\hat{\mu}(x)) - \log|\Sigma_{xx}|^{1/2} + C \tag{2.7}$$

The log-likelihood contains two terms which account for the deviation of points from the mean, $\frac{1}{2}(y-\hat{\mu}(x))^T\Sigma_{xx}(y-\hat{\mu}(x))$, as well as the relative certainty in the prediction of the mean at those points $\log|\Sigma_{xx}|^{1/2}$.

2.3.2 Large Deviations and Large Number Asymptotic Theory

Given two models H_0 and H_1, Large Deviations theory states that if m observations are generated i.i.d. from H_1, the likelihood of those samples with respect to H_0 decays with $\exp(-mD(H_1||H_0))$. In the case that the true models are known a priori, the LRT values will tend towards the KL-Divergence $D(H_1 \mid H_0)$ exponentially fast. This implies that even a small set of points will give good performance for identifying the correct hypothesis. In

online learning, H_1 is not known a priori and must be approximated using a subset of the current data, so the LRT will tend towards

$$\mathbb{E}[L_{H_1|\,H_0}(y)] = D(H_1 \,|\, H_0) - D(H_1 \,|\, \hat{H}_1) \tag{2.8}$$

which is the KL-divergence between the current model and the true distribution minus the approximation error of using a model $\hat{H}_1$ instead of H_1 (see Appendix A).

The theory of large deviations depends on i.i.d. observations of y_i, but the inputs may not be i.i.d. Therefore, the expectation operator is conditioned on x such that if H_1 is implicitly a function of x, (2.8) holds for each x.

Although the strong law of large numbers states that the expected value will tend to (3.1), and that the distribution will Gaussian in the limit as the number of observations increases to infinity, it does not provide information about the distribution structure for smaller numbers of samples. For this purpose, two large deviations inequalities are used, Hoeffding's Inequality [53] and McDiarmid's Inequality [81], to bound the probability of (2.8) deviating from its expected value by more than some specified amount. These are often referred to as concentration inequalities.

Theorem 2.1 (Hoeffding's Inequality) Given n observations $x_1, \ldots, x_n$ drawn i.i.d. from distributions X_i with respective bounded output ranges $p(x_i \in [a_i, b_i]) = 1$. Define $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Then,

$$Pr(|\bar{X} - \mathbb{E}[\bar{X}]| \geq \epsilon) \leq 2\exp\left(-\frac{2n^2\epsilon^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{2.9}$$

In the case that all observations are drawn from the same distribution X, with $V_m = |b - a|$

$$Pr(|\bar{X} - \mathbb{E}[\bar{X}]| \geq \epsilon) \leq 2\exp\left(-\frac{2n\epsilon^2}{V_m^2}\right) \tag{2.10}$$

Hoeffding's Inequality states that the worst case probability that the empirical mean of a group of i.i.d. observations will deviate from the expected value of the mean by more than some ϵ is bounded by a quantity determined by the number of observations, ϵ, and the

range of the output. Hoeffding's Inequality is powerful in that it makes no assumptions about the structure of the distribution, and does not require observations to be drawn from the same distribution. McDiarmid's Inequality is a generalization of the Hoeffding bound. Instead of analyzing the behavior of the empirical mean, McDiarmid's Inequality bounds the probability of a general multi-variate function deviating from its expected value by some ϵ.

Theorem 2.2 (McDiarmid's Inequality) Consider n observations $x_1, \ldots, x_n$ drawn i.i.d. from distributions X_i and a function $f(x_1, x_2, \ldots, x_n)$ which satisfies the following constraint:

$$\sup_{x_1, x_2, \ldots, x_n, \hat{x}_i} |f(x_1, x_2, \ldots, x_n) - f(x_1, x_2, \ldots, \hat{x}_i, \ldots, x_n)| \leq c_i \quad \forall i \tag{2.11}$$

That is, changing one x_i cannot change $f(x_1, x_2, \ldots, x_n)$ by more than c_i. Then

$$Pr(|f(x_1, x_2, \ldots, x_n) - \mathbb{E}[f(x_1, x_2, \ldots, x_n)]| \geq \epsilon) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n} c_i^2}\right) \tag{2.12}$$

2.4 Reinforcement Learning

An RL environment is modeled as a deterministic Markov Decision Process [99] $M = \langle S, A, R, T, \gamma \rangle$ with a potentially infinite set of states S, finite actions A, and discount factor $\gamma < 1$. Each step elicits a reward $R(s, a) \mapsto [R_{\min}, R_{\max}]$ and a deterministic transition to state $s' = T(s, a)$. Every MDP has an optimal value function $Q^*(s, a) = R(s, a) + \gamma V^*(s')$ where $V^*(s) = \max_a Q^*(s, a)$ and corresponding optimal policy $\pi^* : S \mapsto A$. Note the bounded reward means $V^* \in [V_{\min}, V_{\max}]$. The algorithm's current estimate of the value function is denoted as $\hat{Q}(s, a)$.

In RL, an agent is given S,A, and γ and then acts in M with the goal of enacting π^*. For value-based methods (as opposed to policy search [63]), there are roughly two classes of RL algorithms: model-based and model-free. Model-based algorithms, such as KWIK-Rmax [77], build models of T and R and then use a planner to find Q^*. Many model-based approaches have *sample efficiency* guarantees, that is bounds on the amount of *exploration*

they perform.

By contrast, *model-free* methods such as Q-Learning [127] build Q^* directly from experience without explicitly representing T and R. Generally model-based methods are more sample efficient but require more computation time for the planner. Model-free methods are generally computationally light and can be applied without a planner, but need (sometimes exponentially) more samples, and are usually not PAC-MDP. There are also methods that are not easily classified in these categories, such as C-PACE [95], which does not explicitly model T and R but performs a fixed-point operation for planning.

2.4.1 Non-stationary Reward Functions

This thesis also consider MDPs where the reward function may switch at changepoints. This thesis focuses on the class of reward functions $\mathcal{R}$ that satisfy the same assumptions in Section 2.2: *realizability* (Assumption 2.1), *separability* (Assumption 2.2), and *Gaussianity* (Assumption 2.3). In addition, the following assumptions are adopted from [14] to continuous spaces: the region for which functions are well separated is sufficiently large, i.e. it is observable by the agent, and each MDP in the class of possible MDPs $\mathcal{M}$ has a known upper bound on the diameter.

Assumption 2.4 (Separability) There is a known gap Γ of model difference in $\mathcal{M}$: for all $M_i, M_j \in \mathcal{M}$, there exists some compact region of the input region $U \subseteq S$ for which for all $(s,a) \in U$ such that $|\bar{r}_i(s,a) - \bar{r}_j(s,a)| > \Gamma$. Furthermore, U contains at least one voronoi region of the covering set.

The notion of a diameter was first introduced in [5] for discrete state spaces to quantify how long it takes an agent to traverse the entire state space. The diameter intuitively measures the maximum number of steps that an agent must take to move from one state to any other arbitrary state. Here, the notion of a diameter is generalized to continuous spaces by using the covering set. The modified definition says that given any initial state, the diameter is the maximum number of steps required on average to reach any *region* of the state space. This assumption is effectively a reachability assumption. That is, the agent

can reach any region of the state space in order to detect and model the reward function in a finite number of samples.

Definition 2.3 Consider the stochastic process defined by a stationary policy $\pi : S \mapsto A$ operating on an MDP M with initial state s. Let $T(s' \mid M, \pi, s)$ be the random variable for the first time step in which any state $s' \in V_i$, where V_i is the Voronoi region around element $(s_i, a_i) \in \mathcal{N}_{S\times A}(r)$ is reached in this process. Then the *diameter* of M is defined as

$$D(M) := \max_{s \neq s'} \min_{\pi : S \mapsto A} \mathbb{E}[T(s' \mid M, \pi, s)] \tag{2.13}$$

Assumption 2.5 (Known Diameter) There is a known diameter D, such that for every MDP in $\mathcal{M}$, any region of states s' is reachable from any state s by at most D steps *on average*.

2.5 Probably Approximately Correct in MDPs Framework

In the following section, the Probably Approximately Correct in MDPs (PAC-MDP) framework is reviewed. Definitions and Theorems are reproduced from [114]. The PAC-MDP framework is a methodology for analyzing the learning complexity of a reinforcement learning algorithm. Learning complexity refers to the number of steps for which an algorithm must act sub-optimally, or explore, in order to determine the (approximate) optimal policy. In the analysis of RL algorithms, two additional non-negative parameters ϵ and δ are used. ϵ controls the quality of the solution that the algorithm is required to return, and δ controls the confidence in that solution, i.e. the probability that the algorithm functioned correctly and returned a solution within the performance bounds. As these parameters approach zero, more exploration and learning is required in order to return a more accurate solution.

In this context, an algorithm is a non-stationary policy that, on each timestep, takes as input the entire history or trajectory through an MDP, and outputs an action which the agent executes. Given this view of an algorithm, the sample complexity of exploration is formally defined as follows:

Definition 2.4 Let $c = (s_1, a_1, s_2, a_2, \ldots)$ be a random path generated by executing an algorithm $\mathcal{A}$ in an MDP M. For any fixed $\epsilon > 0$, the **sample complexity of exploration (sample complexity)** of $\mathcal{A}$ is the number of timesteps t such that the policy at time t, $\mathcal{A}_t$, satisfies $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$.

Intuitively, this definition states that the sample complexity of an algorithm is the number of steps before the algorithm models the value function accurately (within ϵ) everywhere. Next, an efficient PAC-MDP algorithm is defined to formalize the notion of an efficient learning algorithm.

Definition 2.5 An algorithm $\mathcal{A}$ is said to be an **PAC-MDP** (Probably Approximately Correct in Markov Decision Processes) algorithm if, for any $\epsilon > 0$ and $0 < \delta < 1$, the sample complexity of $\mathcal{A}$ are less than some polynomial in the relevant quantities $(S, A, \frac{1}{1-\gamma}, \frac{1}{\epsilon}, \frac{1}{\delta})$ with probability at least $1 - \delta$.

Now that the definition of PAC-MDP has been established, some definitions and lemmas are introduced from [114]. The main result of [114] shows that if an RL algorithm satisfies certain properties of optimism, high accuracy, and learning rates, then the algorithm is PAC-MDP. These results are used to prove that UCRL-GP is PAC-MDP in Section 4.2.1.

To begin, definitions for a greedy algorithm and the notion of a known state-action pair are presented

Definition 2.6 Suppose an RL algorithm $\mathcal{A}$ maintains a value, denoted $\hat{Q}(s, a)$ for each state-action pair $(s, a) \in S \times A$. Let $\hat{Q}_t(s, a)$ denote the estimate for (s, a) immediately before the t^{th} action of the agent. It is said that $\mathcal{A}$ is a **greedy algorithm** if for the t^{th} action ofid $\mathcal{A}$, a_t is $a_t = \arg\max_{a \in A} \hat{Q}_t(s, a)$, where s_t is the t^{th} state.

Definition 2.7 Let $M = \langle S, A, T, R, \gamma \rangle$ be an MDP with a given set of action values, $\hat{Q}(s, a)$, for each state-action pair (s, a), and a set K of state-action pairs, called the **known state-action pairs**. The **known state-action MDP** $M_K = \langle S \cup \{z_{s,a} | (s, a) \notin K\}, A, T_K, R_K, \gamma \rangle$ is defined as follows. For each unknown state-action pair, $(s, a) \notin K$, a new state $z_{s,a}$ is added to M_K, which has self-loops for each action ($T_K(z_{s,a} \mid z_{s,a}, \cdot) = 1$).

For all $(s,a) \in K$, $R_K(s,a) = R(s,a)$ and $T_K(\cdot \mid s,a) = T(\cdot \mid s,a)$. For all $(s,a) \notin K$, $R_K(s,a) = \hat{Q}(s,a)(1-\gamma)$ and $T_K(z_{s,a} \mid s,a) = 1$. For the new states, the reward is $R_K(z_{s,a},\cdot) = \hat{Q}(s,a)(1-\gamma)$.

Intuitively, K is the set of state-action pairs for which the agent has sufficiently accurate estimates of the reward and dynamics. Finally, the main theorem of [114] is presented.

Theorem 2.3 ([114]) Let $\mathcal{A}(\epsilon,\delta)$ be any greedy learning algorithm such that, for every timestep t, there exists a set K_t of state-action pairs that depends only on the agent's history up to timestep t. It is assumed that $K_t = K_{t+1}$ unless, during timestep t, an update to some state-action value occurs. Let M_{K_t} be the known state-action MDP and $\pi_t(s)$ be the current greedy policy, that is, for all states s, $\pi_t(s) = \arg\max_a \hat{Q}_t(s,a)$. Furthermore, assume $\hat{Q}_t(s,a) \leq V_{max}$ for all t and (s,a). Suppose that for any inputs ϵ and δ, with probability at least $1-\delta$, the following conditions hold for all states s, actions a, and timesteps t: (1) $V_t(s) \geq V^*(s) - \epsilon$ (optimism), (2) $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$ (accuracy), and (3) the total number of updates of action-value estimates can occur is bounded by $\zeta(\epsilon,\delta)$ (learning complexity). Then, when $\mathcal{A}(\epsilon,\delta)$ is executed on any MDP M, it will follow a 4ϵ-optimal policy from its current state on all but

$$O\left(\frac{V_{max}\zeta(\epsilon,\delta)}{\epsilon(1-\delta)}\log\left(\frac{1}{\delta}\right)\log\left(\frac{1}{\epsilon(1-\gamma)}\right)\right) \tag{2.14}$$

timesteps, with probability at least $1-2\delta$.

The key intuition behind the theorem relates to the "optimism in the face of uncertainty" paradigm. This states that if a greedy algorithm remains optimistic, then it will explore regions with low certainty over those with known values. In this case, the greedy algorithm will either act optimally or learn something new about the environment. Since the number of samples required before learning every state action pair in bounded by a polynomial quantity, it follows that a greedy algorithm will learn the value function in a polynomial number of exploratory steps.

2.6 Summary

This chapter reviewed the many technical frameworks from which the algorithms in this thesis are based: Gaussian processes, nonstationary prediction, reinforcement learning, large number asymptotic theory, and sample complexity definitions. The problem statements in this thesis are based in the nonstationary prediction and reinforcement learning frameworks, and these frameworks provide a method for analyzing performance in a meaningful manner. The Gaussian process, large number asymptotic theory, and sample complexity sections provide the machinery required to prove the main theoretical results of this thesis.

Chapter 3

Nonstationary Learning and Prediction

This chapter presents an algorithmic solution to the problem of performing regression and model learning when faced with nonstationary data, i.e. data containing changepoints. Section 3.1 introduces an algorithmic solution to this problem, GP-NBC. In Section 3.2, the algorithm is proven to be theoretically sound in limiting the number of mistaken predictions. In Section 3.3, GP-NBC is shown empirically to perform orders of magnitude faster than existing state of the art algorithms without losing predictive accuracy.

3.1 GP-NBC Algorithm

This chapter considers the problem of performing accurate regression when learning from data containing changepoints. This section presents an algorithm as a possible solution to this problem, the Gaussian Process Non-Bayesian Clustering algorithm, GP-NBC. The full algorithm, GP-NBC, including a component for utilizing previously identified models (or *clusters*), is presented in Algorithm 1. GP-NBC can be initialized with or without a set of prior models. The main idea of the approach to the problem of online non-stationary prediction and clustering is to decouple the problems of prediction, CPD, and model reidentification.

The algorithm begins with a (newly initialized) working model GP_w. For the prediction problem, an algorithm is given x_t and GP_w makes a prediction $\hat{\mu}_w(x_t)$ and is then updated with an observation y_t. Next, for the changepoint detection problem, an LRT (Algorithm 2)

is used, so the algorithm needs to construct a set of points S that the LRT can compare to the current model (Line 6). While a sliding window may suffice when successive generating functions differ at all points in the domain, if there is sufficient overlap between functions one of the following more nuanced approaches is needed:

- The *filtered sliding window* rule uses a buffer of size m_S for S and adds points when their log-likelihood is below a given threshold θ.
- The *local sliding window* rule adds points to local version of S, denoted $S_x(d_x)$, that contains the previous m_S measurements within a given distance of x.

The number of points m_S is domain specific, but generally small sets on the order of $m_S \approx 10$ suffice for most domains. It is later shown the latter method produces a bound on the number of mistakes made by GP-NBC, even in the case of adversarially chosen inputs. However, the experiments section uses the filtered sliding window as it is simpler to implement and often approximates the local sliding window well, especially for real-world data with some (though not strictly i.i.d.) regularities in the inputs.

After this set has been created, a new candidate GP model, GP_S, is created from these points, S, using the same hyperparameters and kernel as GP_w (Line 3), and the LRT is calculated for these points with respect to the two models. If the average of the last m LRT values (L_m) differs significantly from the average LRT values up until that point, then a changepoint is declared and a new model is initialized with an empty S. In the case that a local sliding window is used, the last m LRTs corresponding to the points in that region are used. If a changepoint is detected, the last m points are deleted from GP_w, restoring $GP_w \leftarrow GP_w^{t-m}$. This prevents points which were potentially generated after a changepoint from being included in the old model. Using a sparse GP requires only $O(m|\mathcal{BV}|^2)$ storage for this operation. After a new model is created, an initial burn-in period of approximately $R \in [2m, 4m]$ is used before calculating the LRT again (Line 8).

In practice, the changepoint detection algorithm of GP-NBC works very robustly based on the following intuition. In the first step, the filtered sliding window allows the algorithm to focus on anomalous points while ignoring points that fit the current model well, i.e. were likely generated from the current model. In the second step, a new candidate GP model,

Algorithm 1 GP-NBC

1: **Input**: (Optional) Set of previous models $\{GP_1, \ldots, GP_N\}$
2: Initialize working model GP_w
3: **while** Input/Output $\langle x_t, y_t \rangle$ available **do**
4: Predict $\hat{\mu}_w(x_t)$
5: Update GP_w with $\langle x_t, y_t \rangle$ using [34]
6: Add $\langle x_t, y_t \rangle$ to S according to rules in Sec 3.1
7: Save GP_w to array $\{GP_w^t, \ldots GP_w^{t-m}\}$
8: **if** GP_w has $> R$ measurements **then**
9: Call Algorithm 2
10: Call Algorithm 3
11: **end if**
12: **end while**

GP_S, is created from these points, S.

If the points are anomalous simply because of noise in the output, then on average, the new GP model created from these points will be similar to the current model, and the joint likelihood given the new model will not be substantially different from that of the current model. If the points are anomalous because they were drawn from a new function, then on average, the GP model created from these points will be substantially different from the current model. In this case, the joint likelihood of these points will be much higher for the new model relative to the current model. Lastly, instead of making a decision on a single LRT, the last m LRTs are averaged and compared to the average LRT values seen since the last changepoint. In practice, the LRT may have some offset value due to modeling error. Looking at the difference between the last m values and the average LRT values makes the algorithm robust to this problem. In particular, in Section 3.2, theoretical guidelines for setting m are given. In practice, the algorithm is robust to the selection of parameters m and η, as shown in the empirical section.

To store and reuse previous models, the following modifications are made. Once a model has at least R data points, a test is performed to see if GP_w matches a previous GP model using an LRT (Algorithm 3). In this case, the mean values at the basis vector set $\hat{\mu}_{n+1}(BV_{n+a})$ are used as an artificial data set. If the LRT values indicate that the mean of GP_{n+1} is not substantially different from an old model GP_i, the new model is deleted and the previous model with the lowest LRT score becomes the working model. The sparse

Algorithm 2 Changepoint Detection

Input: Set of points $\mathcal{S}$
$l_1 = \log p(\mathcal{S} \mid GP_w)$
Create new GP $GP_{\mathcal{S}}$ from $\mathcal{S}$
$l_2 = \log p(\mathcal{S} \mid GP_{\mathcal{S}})$
Calculate LRT $L_i(y) = \frac{1}{m}(l_2 - l_1)$
Calculate average of last m LRT: $L_m = \frac{1}{m}\sum_{j=i-m}^{i} L_j(y)$
Calculate average of LRT after changepoint: $L_{ss} = \frac{1}{i-m-1}\sum_{j=1}^{i-m-1} L_j(y)$
$i = i + 1$
if $L_m - L_{ss} \geq \eta$ **then**
    Restore GP_w to previous version
    Initialize GP_{n+1}, set $GP_w = GP_{n+1}$, set $i = 1$
end if

Algorithm 3 Compare to Previous Models

Input: Basis vectors BV_w, prediction values $\hat{\mu}_w(BV_w)$, and library of models
for Each model j in library **do**
    $l_1 = \log P(\hat{\mu}_w(BV) \mid GP_j)$
    $l_2 = \log P(\hat{\mu}_w(BV) \mid GP_w)$
    if $\frac{1}{|\mathcal{BV}|}(l_2 - l_1) \leq \eta$ **then**
        Delete current model and set $GP_w = GP_j$
    end if
end for

online GP implementation ensures the basis vectors are chosen to adequately cover the input domain avoiding over-fitting to one region of the input domain.

In summary, the computational complexity for the LRT is given by $O(|\mathcal{BV}|^2 m)$ for variance calculations. The computational complexity for comparing a current model to all previous models is $O(|\mathcal{BV}|^3 N)$. Note that $|\mathcal{BV}|$ is user tunable, and for most domains a size between 10-200 is sufficient. Hence, the GP-NBC algorithm is real-time computable, even on resource constrained platforms.

3.2 Theoretical Contributions

This section prove bounds on the sample complexity of GP-NBC, specifically how many mistakes (predictions with large error) it will make with high probability. The section begins by reviewing fundamental results to motivate the use of hypothesis testing for non-

stationary model detection. In Section 3.2.1, the key theoretical results determine the maximum number of mistakes a GP can made when learning a single generating function by determining a sufficient condition on the variance (Lemma 3.1) and the rate at which the variance meets that condition (Theorem C.1). In Section 3.2.2, the nonstationary case is analyzed by showing that if certain assumptions are met on the class of functions, the expected value of the LRT (Lemma 3.22) can be lower bounded, and given sufficiently large m (Lemma 3.3), it is proven that GP-NBC will either detect a changepoint or predict accurately in a bounded number of samples (Theorem 3.2).

Given any two hypotheses models H_0 and H_1, the theory of Large Deviations states that if a set of m samples generated i.i.d. from H_1 are observed, the probability that those samples will appear to have been drawn from H_0 decays with $\exp(-mD(H_1\|H_0))$. In the case that the two models are known a priori and the true model is H_1, the LRT values will tends towards the KL-Divergence $D(H_1\|H_0)$ exponentially fast. This implies that even a small set of points will give good performance for identifying the correct hypothesis. In online learning, H_1 is not known a priori and must be approximated using a subset of the current data, so the LRT will tend towards

$$\mathbb{E}[L_{H_1|H_0}(y)] = D(H_1\|H_0) - D(H_1\|\hat{H}_1) \tag{3.1}$$

which is the KL-divergence between the current model and the true distribution minus the approximation error of using a model $\hat{H}_1$ instead of H_1.

The theory of large deviations depends on i.i.d. observations of y_i, but the inputs may not be i.i.d. Therefore, the expectation operator is conditioned on x such that if H_1 is implicitly a function of x, (3.1) holds for each x. It is further assumed that the output range is bounded, $V_m = y_{max} - y_{min}$, the observable input locations belong to a compact domain $x \in U \subset \mathbb{R}^n$, and the mean of the output distribution $f_i(x) = \mathbb{E}[p_i(y \mid x)]$ is Lipschitz over x. Analysis in the stationary case holds for any arbitrary output distribution satisfying these assumptions. In the nonstationary case, further restrictions are required.

The analysis assumes worst case inputs x_t, i.e. samples are chosen adversarially to maximize the number of possible mistakes, without distributional assumptions on the in-

puts. Hence, it is applicable to many real-life applications, such as pedestrian tracking and reinforcement learning, which contain temporally correlated inputs that are ill suited for the simpler i.i.d. setting.

3.2.1 Stationary GP Sample Complexity

The analysis begins by considering the number of mistakes, as defined in Section 1.2.1, possible within a given phase without switching. This limits the rate at which GPs can learn and therefore how frequent changepoints can occur while learning seperate models. Sample complexity results for GPs under a stationary, i.e. without changepoints, setting are derived using adversarial inputs, which is equivalent to worst case analysis. The following lemma links the prediction variance of a GP to the probability that it will make a mistake on a given input.

Lemma 3.1 Consider a GP trained on samples $\vec{y} = [y_1, \ldots, y_t]$ which are drawn from $p_i(y \mid x)$ at input locations $X = [x_1, \ldots, x_t]$, with $\mathbb{E}[p_i(y \mid x)] = f_i(x)$; if the predictive variance of the GP at $x' \in X$ is

$$\sigma^2(x') \leq \sigma_{tol}^2 = \frac{2\omega_n^2 \epsilon_E^2}{V_m^2 \log(\frac{2}{\delta_1})} \tag{3.2}$$

then a mistake at x' is bounded in probability: $Pr\left\{|\hat{\mu}(x') - f_i(x')| \geq \epsilon_E\right\} \leq \delta_1$.

Proof McDiarmid's Inequality states that

$$Pr\{|f(x_1, \ldots, x_n) - \mathbb{E}[f(x_1, \ldots, x_n)]| \leq \epsilon\} \leq 2\text{exp}\left(-\frac{2\epsilon^2}{\sum_i c_i^2}\right) = \delta \tag{3.3}$$

where $c_i = \sup f(x_1, \ldots, x_i, \ldots, x_n) - f(x_1, \ldots, \hat{x}_i, \ldots, x_n)$. I.e. replacing x_i by some other value $\hat{x}_i$ can result in a change in the output $f(x_1, \ldots, x_n)$ no larger than c_i. In the case of an average of the variables, McDiarmid's Inequality becomes Hoeffding's Inequality. Consider the general GP regression equations

$$\mu(X) = K(X, X)(K(X, X) + \omega_n^2 I)^{-1} y \tag{3.4}$$

where $y \in [0, V_m]$ and $\text{Var}(y) \leq V_m^2$. $K(X,X)$ is symmetric and positive semi-definite, so its eigenvectors are orthonormal to eachother and all of its eigenvalues are nonnegative. It can be shown that $K(X,X)$ and $(K(X,X) + \omega_n^2)^{-1}$ have the same eigenvectors. Performing eigendecomposition,

$$\mu(X) = Q\Lambda Q^T Q(\Lambda + \omega_n^2 I)^{-1} Q^T y \tag{3.5}$$

$$\mu(X) = Q\Lambda(\Lambda + \omega_n^2 I)^{-1} Q^T y \tag{3.6}$$

Consider performing prediction only at the first input location x_1 by pre-multiplying using a unit coordinate vector $e_1 = [1, 0, \ldots, 0]^T$.

$$\mu(x_1) = e_1^T Q\Lambda(\Lambda + \omega_n^2 I)^{-1} Q^T y \tag{3.7}$$

This is just a weighted sum of the observations y, with weights given by

$$\alpha = e_1^T Q\Lambda(\Lambda + \omega_n^2 I)^{-1} Q^T \tag{3.8}$$

It follows that $\sum_i c_i^2 = \|\alpha\|_2^2 V_m^2$. Then,

$$\|\alpha\|_2^2 = e_1^T Q\Lambda(\Lambda + \omega_n^2 I)^{-1} Q^T Q(\Lambda + \omega_n^2 I)^{-1} \Lambda Q^T e_1 \tag{3.9}$$

$$\|\alpha\|_2^2 = q_1 \Lambda(\Lambda + \omega_n^2 I)^{-1}(\Lambda + \omega_n^2 I)^{-1} \Lambda q_1^T \tag{3.10}$$

where $q_1 = [Q_{11} \ldots Q_{1n}]$ is the first row of Q. Therefore,

$$\|\alpha\|_2^2 = \sum_i q_{1i}^2 \left(\frac{\lambda_i}{\lambda_i + \omega^2} \right)^2 \tag{3.11}$$

However, by evaluating (3.8), the weight α_1 which corresponds to (x_1, y_1) is given by

$$\alpha_1 = \sum_i q_{1i}^2 \frac{\lambda_i}{\lambda_i + \omega^2}. \tag{3.12}$$

Since every term in (3.12) is greater than every respective term in the sum of (3.11), it

follows that,

$$\|\alpha\|_2^2 \leq \alpha_1 \tag{3.13}$$

In order to finish the proof, an upper bound α_1 is derived. Consider that a GP prediction is equivalent to the MAP estimate of a linear Gaussian measurement model with a Gaussian prior.

$$\mu_{MAP}(x_1) = \frac{\sigma_0^2(x_1)}{\omega^2 + \sigma_0^2(x_1)} y_1 + \frac{\omega^2}{\sigma_0^2(x_1) + \omega^2} \mu_0(x_1) \tag{3.14}$$

In this case, the prior mean $\mu_0(x_1)$, and variance $\sigma_0^2(x_1)$ are given by the GP estimate before including (x_1, y_1), and the weight of the new observation is given by

$$\alpha_1 = \frac{\sigma_0^2(x_1)}{\omega^2 + \sigma_0^2(x_1)} \leq \frac{\sigma^2(x_1)}{\omega_n^2} \tag{3.15}$$

Using this bound on α_1, McDairmid's Inequality states that if

$$\frac{1}{\sigma^2(x_1)} = \frac{V_m^2}{2\omega_n^2 \epsilon_1^2} \log(\frac{2}{\delta}) \tag{3.16}$$

then the prediction is within ϵ_1 of the expected value of the GP prediction with probability $1 - \delta$. This proves that the estimate of the GP concentrates around its expected value with high probability. Since GPs are consistent estimators [102], it follows that the expected value of the GP is the expected value of the distribution $f(x)$. Therefore, it follows that if (3.16) holds, then the estimate of the GP is within ϵ_1 of $f(x)$.

Lemma 3.1 gives a sufficient condition to ensure, with high probability, that the mean of the GP is within ϵ_E of the $f_1(x)$. If the variance is greater than σ_{tol}^2, then one cannot say with certainty that the algorithm will not make a mistake at a point.

Theorem C.1 states that given a compact domain U, a GP can make erroneous predictions no more than n times before driving the variance below $\frac{1}{4}\sigma_{tol}^2$ everywhere. The theorem uses definitions of the covering number of a set (adapted from [95]) and the equivalent distance (Section 2.2)

Theorem 3.1 Consider a sparse GP model with linear independence test threshold $\epsilon_{tol} = \min\{\frac{\epsilon}{2V_m}, \frac{1}{2}\sigma_{tol}^2\}$, trained over a compact domain $U \subset \mathcal{R}^n$ such that $\mathcal{BV}$ adequately covers U, i.e. $\forall x \in U, \gamma(x) = k(x,x) - k(\mathcal{BV},z)^T K(\mathcal{BV},\mathcal{BV})^{-1} k(\mathcal{BV},z) \leq \epsilon_{tol}$. Let the observations $\vec{y} = [y_1, \ldots, y_n]$ be drawn from $p_i(y \mid x)$, with $\mathbb{E}[p_i(y \mid x)] = f_i(x)$, and x drawn adversarially. Then, the worst case bound on the number of samples for which

$$Pr\left\{|\hat{\mu}(x) - f_i(x)| \geq \epsilon_E\right\} \leq \delta_1, \forall x \in U \tag{3.17}$$

is at most

$$n = \left(\frac{4V_m^2}{\epsilon_E^2} \log\left(\frac{2}{\delta_1}\right)\right) |\mathcal{BV}|, \tag{3.18}$$

and the number of basis vectors $|\mathcal{BV}|$ grows polynomially with $\frac{1}{\epsilon_1}$, V_m for the Radial Basis Function (RBF) kernel.

Proof Accounting for the sparsification error of using [34], $\epsilon_{tol} V_m \leq \frac{\epsilon_1}{2}$, it is required that $\sigma^2(x) \leq \frac{1}{4}\sigma_{tol}^2, \forall x \in U$. By partitioning the domain into the Voronoi regions around the set $\mathcal{BV}$, it can be shown using the covariance equation, that given n_V samples in the Voronoi region $\mathcal{BV}_i$, then $\sigma^2(x) \leq \frac{n_V \epsilon_{tol} + \omega^2}{n_V + \omega^2}$ everywhere in the region.

To show this, for all points x in the Voronoi region of point $\bar{x} \in \mathcal{N}_c$, $k(\bar{x},\bar{x}) - k(\bar{x},x)^T K(x,x)^{-1} k(\bar{x},x) \leq \epsilon_{tol}$. Define the correlation coefficients between two points as $\rho = k(x_i, x_j)$. Using the linear independence test, ϵ_t is related to ρ as $\epsilon_t = 1 - \rho^2$. Using bayes law, it can be shown that given a point $\bar{x}$ with prior uncertainty $\sigma_0^2 = 1$ and m measurements at another location x_i with correlation coefficient ρ, the posterior variance is given by $\sigma_n^2 = 1 - \frac{n\rho^2}{n+\omega^2}$. Therefore, in a Voronoi region, the slowest that the variance can reduce at the center of the volume $\bar{z}$ is given by,

$$\sigma_n^2 \leq \frac{n\epsilon_{tol} + \omega^2}{n + \omega^2} \leq \frac{n\epsilon_{tol} + \omega^2}{n} \tag{3.19}$$

Plugging in, $\frac{n_V \epsilon_{tol} + \omega^2}{n_V + \omega^2} \leq \frac{1}{4}\sigma_{tol}^2$ and solving for n_V, $n_V = \left(\frac{4V_m^2}{\epsilon^2} \log\left(\frac{2}{\delta_1}\right)\right)$ points drive the variance at $\mathcal{BV}_i$ below σ_{tol}^2. Therefore, the total number of points that can be sampled anywhere in U before driving the variance below $\frac{1}{4}\sigma_{tol}^2$ everywhere is equal to the sum of points n_V over all regions,$|\mathcal{BV}|$.

It can now be proven that $|\mathcal{BV}|$ grows polynomially with $\frac{1}{\epsilon_1}$, V_m, for the RBF kernel. The covering number $\mathcal{N}_U(r(\frac{1}{2}\sigma_{tol}^2))$ can be bounded by loosely by creating a hyper-parallelopiped which contains the entire state space X, with dimensions $l_1, l_2, \ldots, l_d$, where d is the dimension of the space. The covering number is then loosely bounded by divided the volume of the hyper-parallelopiped by hyper-cubes of dimension $r(\frac{1}{2}\sigma_{tol}^2)$, which is a strictly smaller than the true volume of each Voronoi region. Plugging in,

$$\mathcal{N}_U(r(\frac{1}{2}\sigma_{tol}^2)) = \frac{l_1 l_2 \ldots l_d}{r(\frac{1}{2}\sigma_{tol}^2)^d} \tag{3.20}$$

In the case of the RBF kernel $k(x, x') = \exp(-\frac{\|x-x'\|^2}{2\theta})$, the equivalent distance map is given by

$$r(\epsilon_{tol}) = \theta \left(\log \left(\frac{1}{1-\epsilon_{tol}} \right) \right)^{\frac{1}{2}}. \tag{3.21}$$

$\frac{1}{r(\epsilon_{tol})}$ grows polynomially with $\frac{1}{\epsilon_{tol}}$. Additionally, $\frac{1}{\epsilon_{tol}} = \frac{1}{2}\sigma_{tol}^2 = \frac{V_m^2 \log\left(\frac{2}{\delta}\right)}{\omega^2 \epsilon_1^2}$, so it follows, $\mathcal{N}_U(r(\frac{1}{2}\sigma_{tol}^2)) \sim O\left(f^p(V_m^2, \frac{1}{\epsilon_1^2}, \log\left(\frac{1}{\delta}\right))\right)$, where $f^p(\cdot)$ is some function bounded by a polynomial of order p.

Note that in the adversarial setting, mistakes may occur at any time, but, Theorem C.1 states that the cumulative sum of those mistakes may not exceed some value n which is polynomial in relevant quantities.

3.2.2 Nonstationary Sample Complexity

Now that the number of mistakes using a GP has been bounded in the stationary case, nonstationarity is introduced by having $f(x)$ change between phases. Since the algorithm is online, it must infer this change from nothing other than sequentially observed changed outputs.

In the next lemma, the absolute distance between functions is related to the minimum change in the KL-divergence between a GP_w and the generating distribution. Intuitively, this gives a lower bound on the expected change in LRT values given there is a changepoint. The assumptions listed in Section 2.2 are used.

Lemma 3.2 Consider a GP, GP_w, trained on $p_i(y \mid x)$ with $\sigma_x^2(x) \leq \sigma_{tol}^2 \; \forall x \in \tilde{U}$, and a set of distributions $p_j(y \mid x) \in \mathcal{P}$ satisfying Assumptions 2.1, 2.2, 2.3; then $\forall x \in \tilde{U}$, w. p. $1 - \delta_1$

$$D(p_j(y \mid x) \| GP_w) - D(p_i(y \mid x) \| GP_w) \geq \eta \tag{3.22}$$

where,

$$\eta = \frac{1}{2}\left(\frac{\omega^2 + \sigma_{tol}^2 - \omega_{min}^2}{\omega^2 + \sigma_{tol}^2} - \frac{\epsilon_E^2}{\omega^2} + \log\left(\frac{\omega_{min}^2}{\omega^2}\right) + \frac{(\epsilon_F - \epsilon_E)^2}{\sigma_{tol}^2 + \omega^2}\right) \tag{3.23}$$

Proof The KL-divergence between two normal variables [96] is given by

$$D(p_0 \| p_1) = \frac{1}{2}\left(\frac{\sigma_0^2}{\sigma_1^2} - \log\frac{\sigma_0^2}{\sigma_1} - 1 + \frac{(\mu_0 - \mu_1)^2}{\sigma_1^2}\right) \tag{3.24}$$

The minimum that the first two terms can equal 1 is when $\sigma_1 = \sigma_0$; the maximum is when $\sigma_1^2 = \omega^2 + \sigma_{tol}^2$ and $\sigma_0 = \omega_{min}^2$. By minimizing the variance related terms in $D(p_2(y \mid x) \| GP_w)$, maximizing the variance related terms in $D(p_1(y \mid x) \| GP_w)$, and bounding $|\hat{\mu}_w(x) - f_1(x)| \leq \epsilon_1$, w. p. $1 - \delta_1$, equation (3.23) is obtained.

The expected value of the LRT is KL-divergence between the current distribution and the working model, shifted by the approximation error of using a finite m. From Assumption 2.1, this value is bounded. Therefore, the problem of detecting a change in the generating distribution can be reduced to determining the point in time where the mean of the time series $L(y)$ changes significantly. In GP-NBC, GLR algorithm is used to detect changes. Given a probability of false detection or probability of missing a changepoint δ_L, the next lemma gives conditions on the number of measurements m required to ensure sufficient resolution of the LRT to achieve these specifications.

Lemma 3.3 Consider a, GP_w trained on n_1 samples from $p_1(y \mid x)$. Consider a second GP trained from a set of $m \ll n_1$ samples, which are drawn from $p_2(y \mid x)$ from region $\tilde{U}$, with property, $D(p_2(y \mid x) \| GP_w) - D(p_1(y \mid x) \| GP_w) \geq \eta + \epsilon_{DS}$, $\forall x \in \tilde{U}$. Then, in order to determine if $L_m(y) - L_{ss}(y) \geq \eta$, w. p. $1 - \delta_d - \delta_L$, the window size m must be of length

$$m \geq \frac{\frac{8V_m^4}{\omega^4}\log\left(\frac{4}{\delta_L}\right) n_1}{\eta^2 n_1 - \frac{8V_m^4}{\omega^4}\log\left(\frac{4}{\delta_L}\right)} \tag{3.25}$$

or, for large n_1

$$m \geq \frac{8V_m^4}{\omega^4 \eta^2} \log\left(\frac{4}{\delta_L}\right). \tag{3.26}$$

Furthermore, if these window size conditions are met, Algorithm 2 will detect a change-point w. p. $1 - \delta_d - \delta_L$.

Proof Hoeffding's inequality is used to bound the distance of the LRT from its expected value. Since the output domain is bounded, it follows that the LRT values $L_i(y)$ are bounded. To proceed, the maximal range of the LRT is bounded, and is given by

$$L(y) = \frac{1}{2}\left(\log\frac{\sigma_W^2(x_i)+\omega^2}{\sigma_S^2(x_i)+\omega^2} + \left(\frac{(y_i-\mu_w)^2}{\sigma_w^2(x_i)+\omega^2} - \frac{(y_i-\mu_S)^2}{\sigma_S^2(x_i)+\omega^2}\right)\right) \tag{3.27}$$

The variance is not affected by y_i so the first term is a constant. The last two terms are bounded by the V_m^2/ω^2 each. Therefore, it follows that the LRT $L_i(y)$ is bounded by $c_i = \frac{2V_m^2}{m\omega^2}$. Therefore, using Hoeffding's Inequality, one can bound the distance of the averages L_m, L_{ss} from their respective expected values, $|L_{(\cdot)}(y) - (D(p_i\|GP_w) - D(p_i\|GP_S))| \leq \epsilon_{(\cdot)}$ w. p. $1 - \delta_L/2$, for $(\cdot) \in \{m, ss\}$. Given n_1 and δ_L, ϵ_w can be determined, and the required number of samples m to drive $\epsilon_S \leq \eta - \epsilon_w$ can be solved for through algebraic manipulation.

Lemma 3.3 states that if the generating function switches such that the current model GP_w no longer correctly models the data, then a set of observations built from the new distribution of size m will detect the change with high probability. However, the quantity $D(p_2\|GP_w) - D(p_1\|GP_w) \geq \eta + \epsilon_{DS}$ is implicitly a function of x. In order to determine a maximum number of mistakes the algorithm may make before either detecting a change-point or performing accurate predictions, the selection rule must be considered. Next, consider the case where a local sliding window approach described in Section 3.1 is used.

Consider a KL-divergence change threshold η such that there is an associated absolute error threshold $\epsilon_F < \epsilon_E$. Define $U_E = \{x \in Us.t.|f_i(x) - f_j(x)| > \epsilon_E\}$, and $U_F = \{x \in Us.t.|f_i(x) - f_j(x)| > \epsilon_F\}$. The local sliding window S_x described in Section 3.1 is constructed by adding x_i to $S_x(d_x)$. If there are more than m points in $S_x(d_x)$ such that $d(x_i, x_j) \leq d_x$, then the point with the oldest timestamp in that region is discarded.

When a call is made to $S_x(d_x)$ for x_1, the m nearest points are returned for the LRT. The next theorem bounds the number of mistakes that the algorithm can make after a changepoint before it either detects a changepoint and relearns the function or performs accurate predictions with high probability with a number of mistakes proportional to the covering number $N_U(d_x)$.

Theorem 3.2 Consider GP_w built on samples from the generating distribution $p_i(y \mid x)$. Assume that at some time instant k, there is a changepoint and the generating distribution switches to $p_j(y \mid x)$. Consider a local sliding window $S_x(d_x)$ with $d_x \leq \frac{1}{K}(\epsilon_E - \epsilon_F)$ and m points per set S. Then, Algorithm 2 will detect a changepoint w. p. $1 - \delta_1 - \delta_L$ or predict accurately. Furthermore, the total number of mistakes per phase is bounded by $n + (m-1)\mathcal{N}_{U_E}(d_x)$, w. p. $1 - \delta_d - \delta_L$.

Proof The proof divides the input domain in which mistakes are made into three regions: 1) regions in which $\sigma^2(x) \leq \frac{1}{4}\sigma^2_{tol}$ and $|f_1(x) - f_2(x)| < \epsilon_E$, 2) regions which have moderate variance, $\sigma^2(x) \geq \frac{1}{4}\sigma^2_{tol}$, but if samples from $p_2(y \mid x)$ drive $\sigma^2(x) \leq \frac{1}{4}\sigma^2_{tol}$ then $|\mu_w(x) - f_j(x)| < \epsilon_E$, and regions which have moderate variance, but samples satisfying the previous condition do not guarantee $|\mu_w(x) - f_j(x)| \geq \epsilon_E$. From Lemma C.1 the number of mistakes the GP can make before $\sigma^2(x) \leq \frac{1}{4}\sigma^2_{tol}$ everywhere is n. Therefore, the GP can make no more than n mistakes due to high variance across two successive generating functions. Samples may be queried in an error region U_E or an accurate region. However, after querying $m-1$ samples in each region of the covering set, the next sample will give us m points from the new distribution satisfying (3.22) and Lemma 3.3 guarantees a changepoint will be declared. The most mistakes the GP can make after the changepoint in regions with low variance $\frac{1}{4}\sigma^2_{tol}$ is then bounded by $(m-1)\mathcal{N}_{U_E}(d_x)$. Therefore, combining with the number of mistakes possible due to high variance, n, the total number of mistakes per phase is $n + (m-1)\mathcal{N}_{U_E}(d_x)$, w. p. $1 - \delta_d - \delta_L$ It is enforced that $\epsilon_F < \epsilon_E$, since, without it, points may be sampled arbitrarily close on either side of the boundary of the region U_E, and the nonstationary shift will not be detected.

3.3 Empirical Results

GP-NBC is empirically compared to several state-of-the-art techniques on simulated and real-world datasets. The competing algorithms are: GP-CPD [106], the Dirichlet Process - Gaussian Process (DP-GP) [105], and a MCMC - sampling based Chinese Restaurant Process (MCMC-CRP) algorithm [113] which unlike DP-GP, samples new changepoint configurations, rather than sampling over individual data samples. The results show GP-NBC achieves similar or better performance to the batch algorithms in terms of classification and prediction error even though it sequentially processes individual data-points online. GP-NBC also outperforms the batch algorithms and GP-CPD in terms of speed by several orders of magnitude. GP-NBC is compared to the other algorithms on one synthetic example and two real domains. In the last domain, GP-NBC is combined with DP-GP and it is shown that given a prior set of models output by DP-GP during a training phase, GP-NBC's detection of new clusters leads to significant improvement over only using the prior clusters.

3.3.1 Benchmark Experiments Over A Synthetic Dataset

The first experiment compares the ability of the different methods to differentiate between two very similar functions: $y_1(x) = x^2 - \frac{1}{2} + \epsilon$, and $y_2(x) = \frac{1}{2}x^2 + \epsilon$ over the domain $x \in [-1, 1]$, where $\epsilon \sim \mathcal{N}(0, 0.2^2)$ is Gaussian white measurement noise, and x are drawn i.i.d. from a uniform distribution. In the second experiment, the functions are well separated, $y_1(x) = x^2 + \epsilon$, and $y_2(x) = -x^2 + 2 + \epsilon$, with x drawn i.i.d. This results in an easier problem for the changepoint detection methods, but highlights the downside of using a naïve sliding window approach instead of GP-NBC. In both experiments, the generating function switches from f_1 to f_2 at some time τ drawn uniformly from (75,125), and switches back to f_1 at some τ drawn uniformly from (175,225). The run ends at $t = 400$. Figure 3.3.1 shows these two functions plotted together. The GP-NBC parameters were set to window threshold $\theta = 1$, detection parameter $\eta = 0.5$, and $|S| = 10$. For GP-CPD, the hazard function was set to represent a Poisson process with parameter $\alpha = 0.01$. The DP-GP concentration parameter was set to $\alpha = 0.1$, which yielded the best performance

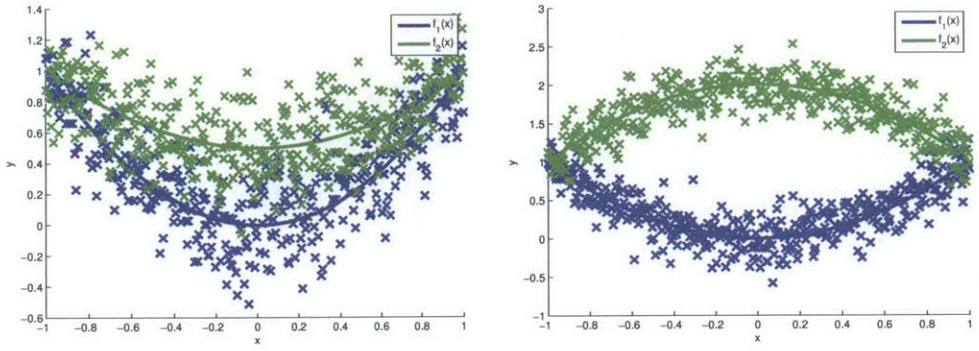


Figure 3-1: (left) f_1 (lower curve) and f_2 for the first experiment and (right) f_1 (lower curve) and f_2 for the second experiment. In the both experiments, a GP is trained on $75 - 125$ noisy samples from $f_1(x)$, the lower curve. The generating function is then switched to $f_2(x)$, the upper curve. After approximately 100 samples, the function is switched back.

Table 3.1: Errors, clusters and runtimes on Experiment 1

	Mean Abs. Error	St. Dev.	Max Error	Cluster No.	Runtime (s)
GP-NBC	**0.0484**	**0.0050**	**0.0624**	2.00 ± 0	3.16
GP-CPD	0.0679	0.0074	0.0848	–	108.4
DPGP	0.0802	0.0167	0.1109	3.066 ± 0.47	217.5
MCMC-CRP	0.0647	0.0469	0.1369	1.88 ± 0.49	158.6
GP-Forget	0.0685	0.0055	0.0843	–	**0.71**

over a manual search. MCMC-CRP used concentration parameter $\alpha = 1$. A naïve implementation of GP-regression with a forgetting factor, GP-Forget, is also compared.

Table 3.1 and Table 3.2 compare 75 runs of each algorithm on experiment 1 and 2, respectively. GP-NBC produces the best mean prediction across both experiments and correctly identifies the proper number of clusters (phases) more robustly than any of the competing methods. In addition, the runtime of GP-NBC is between one to two orders of magnitude faster than the batch methods. MCMC-CRP and DP-GP can achieve the lowest error in the best cases; however, the average errors are very similar. This is because both MCMC-CRP and DP-GP can converge to local optima. The GP-CPD algorithm does not perform as well as GP-NBC since it cannot reuse models and also takes 30 times longer to run. Finally, GP-Forget, runs quickly and performs fairly well on the first example. However, the low error is an artifact of the functions being very close.

In the second example, aliasing due to the sliding window results in extremely large errors for GP-forget while GP-NBC performs equally well in either case. To further demon-

Table 3.2: Errors, clusters and runtimes on Experiment 2

	Mean Abs. Error	St. Dev.	Max Error	Cluster No.	Runtime (s)
GP-NBC	0.0436	0.0050	**0.0662**	2.00 ± 0	3.16
GP-CPD	0.0591	**0.0044**	0.0674	–	108.4
DPGP	0.0755	0.0156	0.1104	4.38 ± 0.70	217.5
MCMC-CRP	**0.0403**	0.0156	0.0835	3.8 ± 1.61	158.6
GP-Forget	0.1556	0.022	0.2032	–	**0.71**

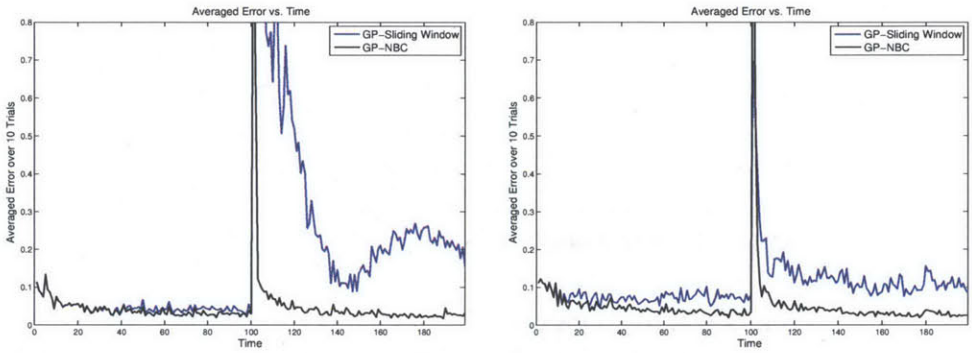


Figure 3-2: (Left) If the sliding window is *too large*, errors from changepoints will persist for long times. (Right) If the sliding window is *too small*, changepoints will be accomodated, but models will never be learned.

strate this problem, Figure 3-2 shows the error of GP-Forget, averaged over 50 trials, using a different sliding window length. If the window is set too large, then transient errors from a changepoint occurring will persist for a long time. Essentially, the algorithm must cycle through a significant amount of outdated data points before the new data can be used to make accurate predictions. Alternatively, if a short window is used, the error resulting from a changepoint will be small, since the algorithm can throw out old data quickly, however, the algorithm will never actually learn a model of the environment, and will have high steady state prediction error. On the other hand, GP-NBC is able to both learn the model, resulting in low steady state error, and also detect changepoints, resulting in consistently low prediction error when trained on nonstationary data.

In Figure 3-3, the performance of GP-NBC over various parameter settings from $\eta \in [0.1, 2]$ and $m \in [3, 50]$ is analyzed. For $\eta \geq 2$, the changepoint does not trigger a changepoint, for $\eta \leq 0.1$, false positives start occurring. For reasonable values of $\eta \approx 0.5$, $m \in [3, 30]$ results in similar performance. This demonstrates the robustness of GP-NBC to parameter selection.

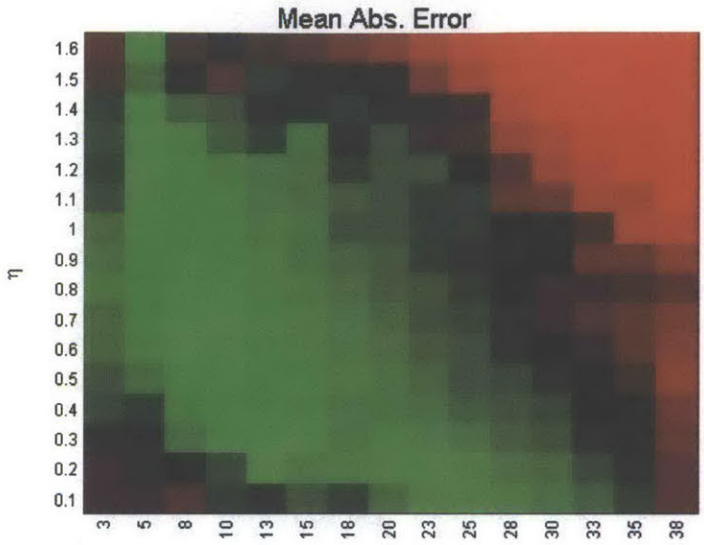


Figure 3-3: Heat map of the Mean Abs Error for various parameter settings for GP-NBC. Green corresponds to better MAE than the optimal parameter settings off GP-CPD, black to equivalent performance, and red to worse performance.

3.3.2 Real Datasets

In this section, GP-NBC is tested on two real datasets. The first dataset involves interaction data between a robot controlled by a human and an autonomous robot whose behavior changes at changepoints. The second dataset analyzes GP-NBC's ability to reidentify previously seen functions in a pedestrian trajectory prediction task.

Robot Interaction

The second experiment demonstrates the ability of GP-NBC to robustly identify multiple models on a real data set. In this experiment, a demonstrator drives a remote controlled iRobot Create, denoted GPUC-1. A second autonomous Create (GPUC-2) reacts to GPUC-1 in a one of four manners: 1) attempting to encircle GPUC-1 clockwise or 2) counterclockwise, 3) attempting to follow GPUC-1, or 4) tracking a point 1m behind GPUC-1. Position and heading measurements of the robots are collected at 5Hz using a VICON motion capture system, and robot velocities are calculated using a fixed point smoother. Every 200 seconds, the behavior of GPUC-2 changes to a new behavior or a previously exhib-

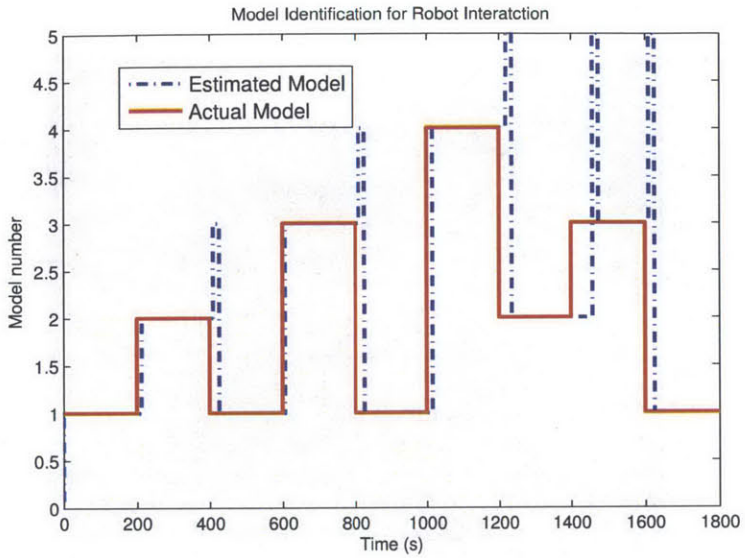


Figure 3-4: GP-NBC detects changepoints as well as reclassifying old models of robot interaction dataset.

ited behavior. The difference in the (x, y) position between GPUC-1 and GPUC-2, and the heading of GPUC-1 is used as the input to the algorithms, and the velocity of GPUC-2 is the output. Figure 3-4 shows a plot in which GP-NBC robustly identifies changepoints in GPUC-2's behavior as well as reclassifies previously seen behaviors. The GP-NBC algorithm took 2.9min to run and had a mean error of 0.0661; GP-CPD took 6.8h and had mean error 0.0785. GP-NBC outperforms GP-CPD since it is able to reuse previous models. MCMC-CRP and DP-GP were stopped after 6 hours. GP-NBC outperformed GP-CPD in terms of regression error as well as taking two orders of magnitude less time.

Pedestrian Behavior Classification

The final experiment demonstrates the ability of GP-NBC to utilize prior models while simultaneously detecting anomalous (new) functions. This capability is relevant to intent-aware prediction [6, 45] where a set of observed training behaviors may be given to an agent, but other unobserved behaviors may be unmodeled. The algorithm is able to robustly reclassify these anomalous behaviors when seen repeatedly, as demonstrated in the experiment. The setting consists of modeling pedestrian behaviors at an intersection, modeled after work by [6]. Real data of a pedestrian walking along one of four trajectories indoors was collected using LIDAR, as seen in Figure 3-5. The algorithm is trained on

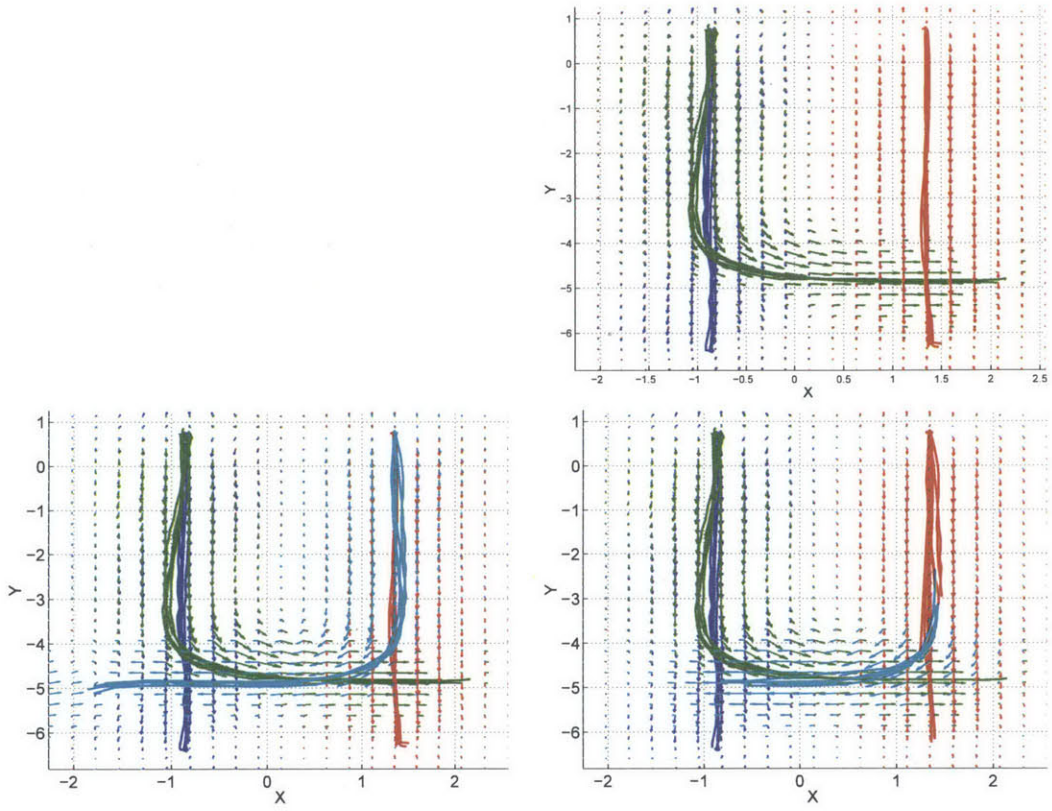


Figure 3-5: Top: Training data. Left: Test data. Right: Output of GP-NBC on test data. GP-NBC detects the new behavior and successfully reclassifies it.

three of the four behaviors (all but the teal trajectories) in batch using DP-GP. In the testing phase, a pedestrian may follow one of the three previously observed trajectories, or may follow a new trajectory, as seen in Figure 3-5 (left). Five trajectories, with an average of only 600 samples per trajectory, were tested with GP-NBC for each behavior. The pedestrian location and velocity were used as GP inputs, and the velocity at the next time step is used as the GP output. Figure 3-5 shows that GP-NBC correctly identifies when a pedestrian deviates from a behavior learned in the training phase and reclassifies the anomalous behavior as the fourth model.

The one-timestep ahead and ten-timestep ahead predictions of the pedestrian's location from GP-NBC are compared with DP-GP given the three training clusters. Prediction error is computed as the root mean square (RMS) difference between the true position and mean predicted position, where the mean position is the GP mean that incorporates the

uncertainty in previous predictions at each time step [48]. Prediction errors for each cluster are averaged across all trajectories at each time step. The prediction error computed by both algorithms for trajectories within the three training clusters was similar (mean difference of 10.25% and 2.47% for one and ten timesteps ahead), but GP-NBC reduced the error in the new fourth cluster by 27% and 44% for the one and ten timestep ahead predictions. As expected, using GP-NBC greatly improves the prediction capabilities of the algorithm, whereas simply using the original clusters from training fails to predict pedestrian behavior when new behaviors are observed. Besides providing better prediction error, GP-NBC fundamentally addresses the problem of dealing with new pedestrian trajectories that are observed in testing but not in training. This requirement is necessary for any sort of system that needs to predict different regression values for different operating behaviors that need to be identified online.

3.4 Conclusions

This chapter presented a computationally efficient algorithm with theoretical guarantees, GP-NBC, for GP regression in the presence of changepoints. GP-NBC acts orders of magnitude faster than other methods for nonstationary GP regression, making it capable of accurate prediction in real-time, which is essential in many online prediction, decision making, and control applications. GP-NBC also has strong theoretical guarantees, making it well suited to applications which require complexity or safety guarantees. These results significantly advance the state-of-the-art in non-stationary online predictive modeling and have significant implications for GP based autonomous decision-making algorithms, as well as regression over nonstationary generative models. Future steps in this project include developing methods for online hyperparameter optimization as well as making the algorithm robust to the selection of hyperparameters.

Chapter 4

Nonstationary Reinforcement Learning

This chapter presents the UCRL-GP-CPD algorithm. UCRL-GP-CPD is a multi-task reinforcement learning algorithm for continuous state spaces and discrete action spaces. First, the chapter introduces the basic UCRL-GP algorithm for single task learning which is inspired by the UCRL [5] and MBIE [116] algorithms. In Section 4.2.1, it is shown that the UCRL-GP algorithm is PAC-MDP, an important class of algorithms which learn the optimal policy in a polynomial number of exploratory steps. The results are among the first PAC-MDP results for continuous RL, following the work of C-PACE [95] and DGPQ and GP-Rmax [52].

Most PAC-MDP algorithms utilize a notion of "optimism in the face of uncertainty" to guide exploration to areas with low certainty. The UCRL and MBIE algorithms work by maintaining an optimistic upper bound on the reward function and the Q function using probabilistic error bounds. Other PAC-MDP algorithms, such as Rmax, maintain optimism by assuming that a state has a maximal value $Q(s,a) = V_{max}$ until enough data points have been observed in that state action pair such that the reward and transition estimates are highly accurate. GP-Rmax [60] is a previously proposed algorithm which generalizes Rmax to continuous domains by modeling the Q function with a GP and assuming that $Q(s,a)$ is V_{max} if the variance of the GP is too high. GP-Rmax was tested extensively empirically in [60], and it was proven in [52] that GP-Rmax is PAC-MDP.

In the worst case, UCRL-GP has the same complexity bounds as GP-Rmax when the reward function is equal everywhere. In practice, UCRL-GP will learn the optimal policy

in fewer steps than GP-Rmax because it uses the uncertainty in a more subtle way. Unlike GP-Rmax, which only uses reward and transition function information once it is highly accurate, UCRL-GP maintains an optimistic upper bound which is a function both of the uncertainty and the current reward estimate. For example, given two regions, one with high reward and one with low reward, GP-Rmax will explore both regions equally until the reward function is highly accurate everywhere. UCRL-GP, on the other hand, will stop exploring the region with low reward once the reward estimate is just accurate enough to differentiate that one region has higher reward.

After establishing sample complexity bounds in the single task learning case, UCRL-GP is extended to nonstationary reward functions UCRL-GP-CPD in Section 4.1 by running a modified version of GP-NBC at each step of the algorithm. It is proven that, in combination with a periodic exploration strategy, GP-NBC will detect changes in the reward function with high probability. UCRL-GP will then learn the new MDP. In order to identify previous MDPs and transfer previous model knowledge, the GP-NBC model reclassification step is modified, and it is proven that, with high probability, UCRL-GP-CPD will not transfer an incorrect model.

4.1 UCRL-GP Algorithm

This section first presents UCRL-GP for single task RL in Algorithm 4. Then, a modified version for multi-task RL, UCRL-GP-CPD, is proposed in Algorithm 6 which uses a modified version of GP-NBC as well as a periodic exploration strategy (Algorithm 8) to guarantee changepoint detection.

Similar to the presentation of GP-Rmax, the representation of the value function is left general in the analysis, but a specific form is proposed in Algorithm 5. For example, GP-Rmax uses GPs to model the reward and transition functions, but approximates the value function by discretizing the input space into a grid and performing Bellman updates at these discrete points. Bilinear interpolation is used to estimate the value function in between grid points. In UCRL-GP, Bellman updates are performed at discrete locations as well, however, these locations are chosen to be the Basis Vectors from the sparse GP representation. By

Algorithm 4 UCRL-GP

1: **Input:** GP reward, transition, and value function kernels $k_r(\cdot,\cdot)$, $k_t(\cdot,\cdot)$, $k_q(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, performance measures δ, ϵ, maximal output range of the reward function V_m
2: **for** $a \in A$ **do**
3: Initialize all reward GPs $GP_a^r = GP.\text{init}(\mu = R_{max}, k_r(\cdot,\cdot))$
4: Initialize all transition GPs $GP_a^t = GP.\text{init}(0, k_t(\cdot,\cdot))$
5: Initialize all value functions $\hat{Q}(s,a) = \frac{R_{max}}{1-\gamma}$
6: **end for**
7: **for** each timestep t **do**
8: Act greedily with respect to current model: $a_t = \arg\max_a \hat{Q}(s,a)$
9: Take action: $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
10: Update transition and reward models: $GP_{a_t}^r.\text{update}(s_t, r_t)$, $GP_{a_t}^t.\text{update}(s_t, s_{t+1})$
11: Learn new value functions $\hat{Q}(s,a)$ using optimistic value iteration algorithm (Algorithm 5)
12: **end for**

choosing basis vectors dynamically as opposed to using a fixed grid, fixed point operations are not performed in locations of the state space that are not accessible by the agent. It also provides an automatic way to determine the appropriate resolution of the points based on approximation error [34]. In addition, one does not need to manually make the mesh grid finer or coarser to control resolution.

In order to perform interpolation, a GP is used in a method similar to GP-DP [40]. At each iteration of the value iteration algorithm (Algorithm 5), a GP is trained using the values at the basis vectors. However, unlike GP-DP, instead of performing updates at every observation location, only the sparse basis vectors are used. This alleviates computational issues as the number of data points increases.

There are two keys differences between the standard Bellman update and the fixed point operation performed in Algorithm 5. In Line 10, an optimistic upper bound on the reward is calculated using Lemma 3.1 instead of using the mean of the GP. In Line 11, the maximum is taken over the value function plus an optimism bonus which accounts for error due to regularization.

In order to detect changes in the environment and transfer past models, two additional steps are added to UCRL-GP. This results in UCRL-GP-CPD, presented in Algorithm 6. Similar to the assumptions of GP-NBC, it is assumed that there are sufficient samples in

Algorithm 5 Optimistic Value Iteration

Input: Input locations BV, reward and transition GP models GP_a^r, GP_a^t, performance measure δ_{VI}, range of reward function V_m, convergence threshold ϵ_{VI}, Lipschitz constant L_Q
for $a \in A$ **do**
  Initialize value function GPs $[GP_a^v]_0 = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k_v(\cdot,\cdot))$
end for
k = 0
while $|[GP_a^v]_{k+1} - [GP_a^v]_k|_\infty \geq \epsilon_{VI}$ **do**
  for $s \in BV$ **do**
    for $a \in A$ **do**
      Get next state after taking action a, $s' = GP_a^t.\text{Predict}(s)$
      $\hat{r}(s,a) = GP_a^r.\text{mean}(s) + \sqrt{\frac{V_m^2 \log\left(\frac{2}{\delta}\right) GP_a^r.\text{Var}(s)}{2\omega_n^2}}$
      $y_a(s,a) = \hat{r}(s,a) + \gamma \max_{a'} \left([GP_{a'}^v.\text{mean}(s')]_k + \frac{GP_{a'}^t.\text{Var}(s)}{\omega_n^2} L_Q \right)$
      if $y_a > V_{max}$ **then**
        $y_a = V_{max}$
      end if
      Replace value function GP with new GP: $[GP_a^v.\text{update}(s, y_a)]_{k+1}$
    end for
  end for
  k = k+1
end while

between changepoints. This assumption allows the decoupling of the problems of planning, changepoint detection, and model transfer. The key distinctions between UCRL-GP-CPD and UCRL-GP are given in the lines of Algorithm 6: Line 9, a set of least likely points is maintained to identify nonstationary phenomena, Line 10, an array of GP models is saved to allow reverting back to previous models in the case of a changepoint, Line 13, Changepoints are tested for using (Algorithm 2), in Line 14, a modified version of the Model Transfer Algorithm is run (Algorithm 7).

Lastly, in order to guarantee that changepoint will be detected, the state space must be explored periodically. In Line 9 of Algorithm 6, exploration is performed until either a changepoint has been detected or enough of the state space has been explored to declare with high probability no changepoint has occurred.

In order to prove that UCRL-GP-CPD will transfer models without negative transfer, a modified version of the model transfer algorithm is proposed (Algorithm 7). This algorithm works similarly to [14, 74]. In those algorithms, a set of candidate models is maintained which are similar to the current reward function. At each step, optimistic and pessimistic bounds are calculated for each model and the current model. If the lower bound of the candidate model is greater than the upper bound of the working model at any location, then with high probability, the algorithm knows that the current reward function is not the same as the candidate model (Line 6 of Algorithm 7). In this case, the candidate model can be removed from future comparison. Conversely, if the upper bound of the candidate model is less than the lower bound of the working model at any location, this candidate model can also be removed from consideration for transfer. Note, the model is not deleted from memory, but it is simply removed from consideration for model transfer. Lastly, if the confidence bounds of the candidate model are sufficiently close to those of the working model at all locations, the candidate model is transferred (Line 13 of Algorithm 7).

In [14], all of the models are available during a training phase, so the transfer algorithm is guaranteed to transfer the correct model with high probability. In [74], the models are not available a priori, but are determined at the beginning of each episode using a clustering method after an exploratory period. In UCRL-GP-CPD, it is assumed that reward functions are well-separated by some known quantity, and so if the current model looks sufficiently

Algorithm 6 UCRL-GP-CPD

Input: GP reward, transition, and value function kernels $k_r(\cdot,\cdot)$, $k_t(\cdot,\cdot)$, $k_q(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, performance measures δ,ϵ, maximal output range of the reward function V_m, changepoint detection parameters $R, m, \delta_{CP}, \epsilon_{CP}, \epsilon_E = \Gamma$.
for $a \in A$ **do**
  Initialize all reward GPs $GP_a^r = GP.\text{init}(\mu = R_{max}, k_r(\cdot,\cdot))$
  Initialize all transition GPs $GP_a^t = GP.\text{init}(0, k_t(\cdot,\cdot))$
  Initialize all value functions $\hat{Q}(s,a) = \frac{R_{max}}{1-\gamma}$
end for
for each timestep t **do**
  if Exploratory flag **then**
    Run exploratory algorithm such as Algorithm 8
  else
    Act greedily with respect to current model: $a_t = \arg\max_a \hat{Q}(s,a)$
    Take action: $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
    Update transition and reward models: $GP_{a_t}^r.\text{update}(s_t, r_t)$, $GP_{a_t}^t.\text{update}(s_t, s_{t+1})$
    Add $\langle s_t, r_t \rangle$ to least likely set $\mathcal{S}$ according to rules in Sec 3.1
    Save GP_a^r to array $\{[GP_a^r]^t, \ldots [GP_a^r]^{t-m}\}$
    for $a \in A$ **do**
      if GP_a^r has $> R$ measurements **then**
        Call Changepoint Detection Algorithm (Algorithm 2)
        Call Modified Model Transfer Algorithm (Algorithm 7)
      end if
    end for
    Learn new value functions $\hat{Q}(s,a)$ using optimistic value iteration (Algorithm 5)
  end if
end for

Algorithm 7 Modified Model Transfer

1: **Input**: Basis vectors BV_w, current working model GP_w, library of candidate models, model gap Γ, performance measures δ_{MT}.
2: Calculate optimistic upper bound on current reward function at Basis Vector locations: $r_w^{UCB}(BV_w) = GP_a^r.\text{mean}(s) + \sqrt{\frac{V_m^2 \log\left(\frac{2}{\delta_{MT}}\right) GP_a^r.\text{Var}(s)}{2\omega_n^2}}$
3: Calculate pessimistic lower bound on current reward function at Basis Vector locations: $r_w^{LCB}(BV_w) = GP_a^r.\text{mean}(s) - \sqrt{\frac{V_m^2 \log\left(\frac{2}{\delta_{MT}}\right) GP_a^r.\text{Var}(s)}{2\omega_n^2}}$
4: **for** Each model j in candidate library **do**
5: Calculate optimistic and pessimistic bounds on reward function at basis vector locations using lines 2-3: $r_j^{UCB}(BV_w)$, $r_j^{LCB}(BV_w)$
6: **if** $\exists s \in BV_w$, s.t. $r_j^{LCB}(BV_w) > r_w^{UCB}(BV_w)$ **then**
7: Remove model j from candidate library. Continue to next model.
8: **end if**
9: **if** $\exists s \in BV_w$, s.t. $r_j^{UCB}(BV_w) < r_w^{LCB}(BV_w)$ **then**
10: Remove model j from candidate library. Continue to next model.
11: **end if**
12: **if** $\forall s \in BV_w$, $|r_j^{LCB}(BV_w) - r_w^{UCB}(BV_w)| < \Gamma$ and $|r_j^{UCB}(BV_w) - r_w^{LCB}(BV_w)| < \Gamma$ **then**
13: Transfer Model: Delete current model and set $GP_w = GP_j$
14: **end if**
15: **end for**

close to an old model, the algorithm knows that the reward functions are not well separated, so they must be the same.

If a change occurs along the trajectory the agent follows, then UCRL-GP-CPD will detect the change. However, if a change occurs somewhere else in the environment, the agent must periodically explore. This concept was first introduced in [2] as a *fog of war* function. In UCRL-GP-CPD, the exploratory policy is left to be general, but a specific policy is proposed in Algorithm 8. Additionally, in the algorithm statement and analysis, the issue of when to explore is left general. In the experiments, a simple counter since the last exploration period is used. This is a naïve approach, but satisfies the conditions for determining if a changepoint occurs in Lemma 4.7.

Ideally, an exploration policy would investigate uncertain regions of the state space initially, and if the agent observes an anomalous point, the agent should explore this region further to determine if the reward function has truly shifted. This thesis takes a targeted exploration approach based on this principle in Algorithm 8. The algorithm initializes

Algorithm 8 Exploration Algorithm

Input: Current reward model GP_a^r, Error threshold ϵ_E
Initialize new temporary reward functions $\hat{GP}_a^r = GP.\text{init}(\mu = R_{max}, k(\cdot,\cdot))$
Solve for $\hat{Q}_{ex}(s,a)$ using reward function $e_a(s)$ from Algorithm 9, transition dynamics model GP_a^t, $\gamma = 1$, and using standard value iteration.
for each timestep t **do**
  Act greedily with respect to exploration model: $a_t = \arg\max_a \hat{Q}_{ex}(s,a)$
  Take action: $\langle r_t, s_{t+1}\rangle = Env.\text{takeAct}(a_t)$
  Update transition and reward models: $GP_{a_t}^r.\text{update}(s_t, r_t)$, $GP_{a_t}^t.\text{update}(s_t, s_{t+1})$
  Update temporary reward models: $\hat{GP}_{a_t}^r.\text{update}(s_t, r_t)$
  Add $\langle s_t, r_t\rangle$ to least likely set $\mathcal{S}$ according to rules in Sec 3.1
  Save GP_a^r to array $\{[GP_a^r]^t, \ldots [GP_a^r]^{t-m}\}$
  for $a \in A$ **do**
    if GP_a^r has $> R$ measurements **then**
      Call Changepoint Detection Algorithm (Algorithm 2)
      Call Modified Model Transfer Algorithm (Algorithm 7)
      if Changepoint Detected **then**
        Return
      end if
    end if
  end for
end for

a new temporary estimate of the reward function. At each step, a new MDP is defined using a error reward $e_a(s)$ (Algorithm 9), and solved to find the optimal path. After each observation, both the temporary reward models and the working reward model are updated. The function $e_a(s)$ acts as a binary error signal, in which the reward is 1 if the temporary reward model disagrees with the current model, and 0 is the temporary model is sufficiently close to the temporary model. Initially, the exploration algorithm will explore all regions of the state space. If a point is anomalous with regard to the working reward model, then the temporary model will update its mean to be close to the anomalous point. If this case, there will still be a large difference between the temporary model and the working model, so the agent will receive a reward for exploring this region further. If, on the other hand, the observed point is similar to the working model, the reward models will be similar, so no reward will be given for further exploring this region.

Algorithm 9 Function: Binary Error Signal

1: **Input**: Current reward model GP_a^r, temporary reward model $\hat{GP}_a^r$, model gap Γ, probability of failure δ_{BE}, state action pair (s, a)
2: $\Delta m = |\hat{GP}_a^r.\text{mean}(s) - GP_a^r.\text{mean}(s)|$
3: $\Delta s = \sqrt{\frac{V_m^2 \log\left(\frac{4}{\delta_{BE}}\right) GP_a^r.\text{Var}(s)}{2\omega_n^2}} + \sqrt{\frac{V_m^2 \log\left(\frac{4}{\delta_{BE}}\right) \hat{GP}_a^r.\text{Var}(s)}{2\omega_n^2}}$
4: Return $e_a(s) = 1$, if $\Delta m + \Delta s \geq \Gamma$, and $e_a(s) = 0$, o.w.

4.2 Theoretical Results

This section presents sample complexity results for UCRL-GP and theoretical results about negative transfer and successful detection of changepoints for UCRL-GP-CPD.

4.2.1 Stationary Learning

In order to prove that UCRL-GP is PAC-MDP, it is proven that UCRL-GP maintains all of the requirements to invoke Theorem 2.3 (originally from [114]): 1) optimism, 2) accuracy of known states, and 3) the total number of updates of action-value estimates, i.e. number of updates before all states are known, can occur is bounded by a polynomial function $\zeta(\epsilon, \delta)$.

First, it is proven that UCRL-GP remains optimistic with high probability. Pazis and Parr [95] prove that for the exact Bellman operator $BQ(s, a)$, if $Q(s, a)$ is initialized optimistically, and a fixed-point operation with termination threshold ϵ_{VI} is used, the resulting $Q(s, a)$ will be optimistic as well. This lemma is reproduced below.

Lemma 4.1 Let $\epsilon \geq 0$ be a constant such that $\forall (s, a) \in (S, A)$, $BQ(s, a) \leq Q(s, a) + \epsilon_{VI}$, where $BQ(s, a)$ denotes the exact Bellman operator. Then,

$$\forall (s, a) \in (S, A), Q^*(s, a) \leq Q(s, a) + \frac{\epsilon_{VI}}{1 - \gamma} \tag{4.1}$$

In the next lemma, it is proven that by setting the parameters δ_{VI} and ϵ_{VI} properly, then with high probability, UCRL-GP will remain optimistic during the value iteration stage.

Lemma 4.2 Setting $\delta_{VI} = \frac{\delta}{|BV||A|}$, $\epsilon_{VI} = \epsilon(1 - \gamma)$, UCRL-GP will remain optimistic

with high probability $1 - \delta$

$$\forall (s,a) \in (S,A), Q^*(s,a) \leq Q(s,a) + \epsilon, \text{ w.p. } 1\text{-}\delta \tag{4.2}$$

Proof Let $BQ(s,a)$ denote the exact Bellman operator and $\hat{B}Q(s,a)$ denote the approximate Bellman operator in Algorithm 5. If $\hat{B}Q(s,a) \geq BQ(s,a), \forall (s,a)$ at each iteration, it follows from Lemma 4.1 that UCRL-GP remains optimistic. It is said that the value iteration has *failed*, if $\exists (s,a)$, s.t. $\hat{B}Q(s,a) < BQ(s,a)$.

In order to maintain optimism, Algorithm 5 uses an optimism bonus in the reward function to account for stochasticity in the reward function as well as an optimism bonus in performing Bellman updates using the location $s' = T(s,a)$, when the transition function is unknown. In particular,

$$\hat{B}Q(s,a) - BQ(s,a) = \Delta r(s,a) + \gamma \Delta Q(s,a) \tag{4.3}$$

where

$$\Delta r(s,a) = GP_a^r.\text{mean}(s) + \sqrt{\frac{V_m^2 \log\left(\frac{2}{\delta}\right) GP_a^r.\text{Var}(s)}{2\omega_n^2}} - \bar{r}(s,a) \tag{4.4}$$

$$\Delta Q(s,a) = \max_{a'} \left([GP_{a'}^v.\text{mean}(s')]_k + \frac{GP_{a'}^t.\text{Var}(s)}{\omega_n^2} L_Q \right) - Q(T(s,a),a) \tag{4.5}$$

If $\Delta r(s,a), \Delta Q(s,a) \geq 0$, $\forall (s,a)$, it follows that Algorithm 5 remains optimistic, so UCRL-GP remains optimistic during the value iteration phase. Therefore, the error contributed by each term is analyzed separately.

Since the dynamics are deterministic, the error in $\Delta Q(s,a)$ is due to performing a backup at some state $s' \neq T(s,a)$ due to generalization error. The maximum error of propagating $Q(s',a')$ instead of $Q(T(s,a),a')$ is given by $\|s' - T(s,a)\|_\infty L_Q$, which is the distance from s' to the true transition point, multiplied by the Lipschitz constant L_Q. The distance $\|s' - T(s,a)\|_\infty$ is upper bounded by the regularization error $\frac{GP_{a'}^t.\text{Var}(s)}{\omega_n^2}$ (see Section A.2). Therefore, $\Delta Q(s,a) \geq 0$.

From Lemma 3.1, the probability that $|GP_a^r.\text{mean}(s) - \bar{r}(s,a)| > \sqrt{\frac{V_m^2 \log\left(\frac{2}{\delta_{VI}}\right) GP_a^r.\text{Var}(s)}{2\omega_n^2}}$

for any individual point is δ_{VI}. The probability of the operation failing is upper bounded by the sum, or union bound, of the probabilities of an update failing at any location. Setting $\delta_{VI} = \frac{\delta}{|BV||A|}$, the total probability of failure is given by δ. Therefore, with probability $1-\delta$, $\hat{BQ}(s,a) > BQ(s,a), \forall(s,a)$, completing the proof.

In order to prove accuracy of a known state in Lemma 4.3, the following definition is used for a known state.

Definition 4.1 During the execution of UCRL-GP, a state action pair (s,a) is said to be *known* if

$$GP_a^r.\text{Var}(s) \leq \frac{2\omega_n^2\epsilon_K^2(1-\gamma)^2}{V_m^2\log(\frac{2}{\delta_K})} \tag{4.6}$$

and

$$GP_t^r.\text{Var}(s) \leq \frac{\epsilon_K\omega^2}{L_Q} \tag{4.7}$$

where $\epsilon_K = \frac{\epsilon(1-\gamma)}{2}$, and $\delta_K = \frac{\delta}{2|BV||A|}$

In the next lemma, Lemma 4.3, it is proven that a known state maintains highly accurate estimates of the reward and transition functions.

Lemma 4.3 During the execution of UCRL-GP, the estimate of the reward function and transition dynamics of a known state action pair (s,a) is highly accurate (within ϵ_K) with high probability $1-\delta_K$:

$$Pr\left\{|GP_a^r.\text{mean}(s) - \mathbb{E}[r(s,a)]| \geq \epsilon_K\right\} \leq \delta_K \tag{4.8}$$

$$Pr\left\{|GP_a^t.\text{mean}(s) - T(s,a)]| \geq \epsilon_K\right\} \leq \delta_K \tag{4.9}$$

Proof The first statement follows immediately from Lemma 3.1. The second statement follows from upper bounding the error due to regularization, given in Section A.2.

While Lemma 4.3 proves that the estimates at individual state action pairs are highly accurate, extra steps are required to prove that during the value iteration stage, this results in an estimate of the value function which is consistent with the current MDP model. The reader is referred to Section 2.5 for certain definitions from [114]. In particular, the notion

of known states K and the known state-action MDP M_K is used (Definition C.3). In the next definition, Event A1 is defined to be the event in which the value function of all known states is highly accurate given the current MDP.

Definition 4.2 (Event A1) For all stationary policies π, timesteps t, and states s during execution of the UCRL-GP algorithm on some MDP M, $|V^{\pi}_{M_{K_t}}(s) - V^{\pi}_{\hat{M}_{K_t}}(s)| \leq \epsilon$

Lemma 4.4 If UCRL-GP is executed on any MDP M, using the definition of a known state from Definition 4.1, then Event A1 will occur will probability $1 - \delta$.

Proof Event A1 occurs if UCRL-GP maintains a close approximation of its known state-action MDP. By Lemma 9 and Lemma 12 of [114], it is sufficient to obtain an $\epsilon(1-\gamma)$ approximation of the reward and transition functions to ensure $|V^{\pi}_{M_{K_t}}(s) - V^{\pi}_{\hat{M}_{K_t}}(s)| \leq \epsilon$. From Lemma 4.3, UCRL-GP maintains a $\epsilon(1-\gamma)$ approximation of the reward and transition functions at each state action pair with probability $1 - \delta_K$. Applying a union bound over the probability of failure of any location, it holds that with probability $1 - \delta$, UCRL-GP maintains a $\epsilon(1-\gamma)$ approximation everywhere.

Lastly, a bound on the number of events in which a new state action may become known is derived.

Lemma 4.5 Given an MDP M, with state action space $S \times A$, the maximum number of times that a state-action pair may become known is given by

$$\mathcal{N}_{S\times A}\left(r\left(\sigma^2_{tol}\right)\right), \tag{4.10}$$

where

$$\sigma^2_{tol} = \frac{\omega^2_n \epsilon^2_K (1-\gamma)^2}{V^2_m \log(\frac{2}{\delta_K})} \tag{4.11}$$

Proof Following the proof of Theorem C.1, the state space can be partitioned into Voronoi regions around the covering set. If the variance of a point drops below σ^2_{tol}, then every state within the Voronoi region of the covering set element satisfies the property $GP^r_a.\text{Var}(s) < 2\sigma^2_{tol}$. From Definition 4.1, if the variance at a point is below $2\sigma^2_{tol}$, it is defined as known.

Therefore, the maximum number of points that can become known is given by the number of Voronoi regions, i.e. the covering number.

Now that the key properties of optimism, accuracy, and bounded number of events in which a new state action pair may become known have been established, the general PAC-MDP Theorem of [114] is invoked.

Theorem 4.1 If UCRL-GP is executed on any MDP M, then with probability $1-\delta$, UCRL-GP will follow a 4ϵ-optimal policy from its current state on all but

$$O\left(\frac{V_m \zeta(\epsilon,\delta)}{\epsilon(1-\gamma)} \log\frac{1}{\delta} \log\frac{1}{\epsilon(1-\gamma)}\right) \tag{4.12}$$

timesteps, with probability at least $1-2\delta$, where

$$\zeta(\epsilon,\delta) = \left(\frac{4V_m^2}{\epsilon^2(1-\gamma)^2} \log\left(\frac{2|\mathcal{N}_{S\times A}\left(r\left(\sigma_{tol}^2\right)\right)|}{\delta}\right)\right) \mathcal{N}_{S\times A}\left(r\left(\sigma_{tol}^2\right)\right) \tag{4.13}$$

4.2.2 Non-Stationary Reward Functions

This section considers the analysis of non-stationary reward functions. The assumptions of separability and well-behavedness are as presented in Section 2.2. An additional assumption on the magnitude of the model gap between reward functions is added:

Assumption 4.1 For all $r_i, r_j \in \mathcal{R}$, $\exists U$, s.t. $|r_i - r_j| \geq \Gamma$ where $\Gamma = 2\epsilon_K$.

This assumption states that the magnitude of changes in the reward function is at least twice the resolution of a known state. Otherwise, the resolution of a known state is not fine enough to detect changes or transfer past models.

In this section, it is proven that if there is a changepoint and the reward function changes by some Γ over at least one Voronoi region of the covering set, UCRL-GP-CPD will detect this change or predict accurately, provided there is a sufficient number of samples between changepoints. To do this, it is first proven that if the UCRL-GP-CPD receives m samples from a region in which there is a change in the reward function, UCRL-GP-CPD will detect the change or predict accurately in a bounded number of samples (Lemma 4.6).

It is then proven that given the exploration strategy presented in Section 4.1, UCRL-GP-CPD will collect m such samples (Lemma 4.7) with high probability. Lastly, it is proven that the modified model transfer Algorithm 7 will not transfer an incorrect model with high probability.

In the following lemma, it is proven that UCRL-GP-CPD will detect a changepoint or predict accurately if it receives m samples in a region in which the reward function has changed by at least Γ.

Lemma 4.6 Consider UCRL-GP-CPD using GP-NBC with threshold η and window size m, defined below, trained on samples of MDP i with reward function $r_i(s,a)$. If a changepoint occurs at some time t, and the reward function switches to $r_j(s,a)$ satisfying the assumptions in Section 2.2 and Assumption 4.1, with region of separability U, then if the agent collects

$$m \geq \frac{8V_m^4}{\omega^4\eta^2}\log\left(\frac{4}{\delta_{CP}}\right). \tag{4.14}$$

where

$$\eta = \frac{1}{2}\left(\frac{\omega^2+\sigma_{tol}^2-\omega_{min}^2}{\omega^2+\sigma_{tol}^2} - \frac{\Gamma^2}{4\omega^2} + \log\left(\frac{\omega_{min}^2}{\omega^2}\right) + \frac{\Gamma^2}{4\sigma_{tol}^2+\omega^2}\right) \tag{4.15}$$

samples from U, then with probability $1-\delta_{CP}$, either UCRL-GP-CPD will declare a changepoint or predict accurately $|GP_a^r.\text{mean}(s) - r_j(s,a)| \leq \epsilon_K$

Proof Substituting $\Gamma = \epsilon_E$, $\epsilon_1 = \epsilon_K$, $\delta_1 = \delta_{CP}$ into (3.25) and (3.23) yields the above equations. The result follows immediately from Lemma 3.3.

In the next lemma, it is proven that if a changepoint occurs and the exploration Algorithm 8 is executed, UCRL-GP-CPD will collect m samples with high probability.

Lemma 4.7 Consider an execution of UCRL-GP-CPD with estimate of the reward functions $GP_a^r.\text{mean}(s)$ trained on all data since the last changepoint, and newly initialized estimates of the reward functions $\hat{GP}_a^r.\text{mean}(s)$. If at time t_C, a changepoint occurs, and the reward function switches to $r_j(s,a)$ satisfying assumptions in Section 2.2 and Assumption 4.1 and exploration strategy (Algorithm 8) is executed at time $t > t_C$, then UCRL-GP-CPD will declare a changepoint or predict accurately $|GP_a^r.\text{mean}(s) - \bar{r}(s,a)| \leq \Gamma$ with probability $1-\delta_{CP}$ within $m(\mathcal{N}_{S\times A}(r(\sigma_{tol}^2)))D$ steps, where D is the MDP diameter.

Proof Exploration Algorithm 8 terminates when UCRL-GP-CPD declares a changepoint or when the confidence bounds of the new reward model $\hat{GP}_a^r$ become sufficiently close to the old reward model GP_a^r. If a changepoint has occurred, the current GP model may either successfully learn the new reward function if insufficient samples were seen previously in a region, or the model will continue to predict inaccurately, $|GP_a^r.\text{mean}(s) - \bar{r}(s,a)| > \Gamma$.

The difference between the outer confidence bounds of these two models is given from (4) to be $\Delta m + \Delta s$. That is, $\Delta m + \Delta s$ represents the maximum distance between the current reward function and the reward model, with high probability. Therefore, the distance will fall below the error threshold, $\Delta m + \Delta s \leq \Gamma$ with probability δ_{CP}, and the exploration algorithm will terminate without predicting a changepoint. The probability that $\Delta m + \Delta s \geq \Gamma$ for all regions that changed is $1 - \delta_{CP}$. The exploration will not terminate unless $\Delta m + \Delta s \leq \Gamma$ for all (s,a), so the exploration algorithm will collect m samples from regions that are well separated. By Lemma 4.6, GP-NBC will then declare a changepoint. The expected maximum number of samples before the algorithm collects m samples from the region of interest is given by the covering number of the space $\mathcal{N}_{S\times A}(r(\sigma_{tol}^2))$ multiplied by m multiplied by the diameter of the MDP D.

Lastly, it is proven that the modified model transfer Algorithm 7 will not transfer a model which is inaccurate.

Lemma 4.8 Consider an execution of UCRL-GP-CPD with estimate of the reward functions $GP_a^r.\text{mean}(s)$ trained on all data since the last changepoint, and set of previous models $[GP_a^r]^j$. Algorithm 7 will not transfer a previous model which is inaccurate with high probability $1 - \delta_{MT}$, i.e.

$$|GP_a^j.\text{mean}(s) - \bar{r}(s,a)| \leq \Gamma \,, \forall s, a \tag{4.16}$$

Proof Model Transfer Algorithm 7 transfers a model when the maximum distance between the confidence bounds of the current model and the past model differ by no more than Γ anywhere. Therefore, the past model differs no more than Γ from the current reward function with high probability $1 - \delta_{MT}$.

4.3 Empirical Results

This section presents empirical results comparing UCRL-GP to state of the art RL algorithms for continuous state spaces with stationary reward functions as well as empirical results for UCRL-GP-CPD for various nonstationary reward domains.

4.3.1 Stationary Reward Functions

UCRL-GP is compared against several state of the art RL algorithms for continuous state spaces. One of the primary contributions of UCRL-GP is efficient (PAC-MDP) exploration in continuous state spaces using flexible nonparametric function approximation. UCRL-GP is compared to the only existing PAC-MDP algorithms for continuous state spaces in the literature: CPACE [95], an instance based algorithm which uses local averages to represent the Q function directly, DGPQ [52], a model-free algorithm which uses GPs, and GP-Rmax [60], a model-based algorithm which uses GPs. The algorithms are tested on two synthetic domains: puddle world [9] and a cart-pole balancing task [72].

In the first domain, puddle world, an agent must navigate a continuous planar world $S : [0, 1]^2$ from one corner to another while avoiding a puddle. The agent receives a reward of -1 for every step it takes, and receives an additional negative reward for stepping in the puddle proportional to the distance to the center of the puddle. A graphic of the puddle world domain is shown in Figure 4-1. At each step, the agent takes a step in one of the cardinal directions with magnitude $0.1 + \eta$, where $\eta \sim \mathcal{N}(0, 0.01^2)$ is white Gaussian noise. The agent starts in the corner $[0, 0]^T$ and ends when the agent reaches within 0.15 L_1 distance of the goal $[1, 1]^T$ or when the episode length exceeds 200 steps.

In Figure 4-2, the performance of each PAC-MDP algorithm is compared on puddle world. All of the model-based approaches, GP-Rmax, CPACE, and UCRL-GP learn the optimal policy in a similar number of episodes, while the model-free approach, DGPQ, takes substantially more samples. This is due to the inherent trade-off between computational efficiency and sample efficiency. DGPQ only performs a maximum of one Bellman update per iteration, resulting in computational efficiency, but at the cost of sample efficiency.

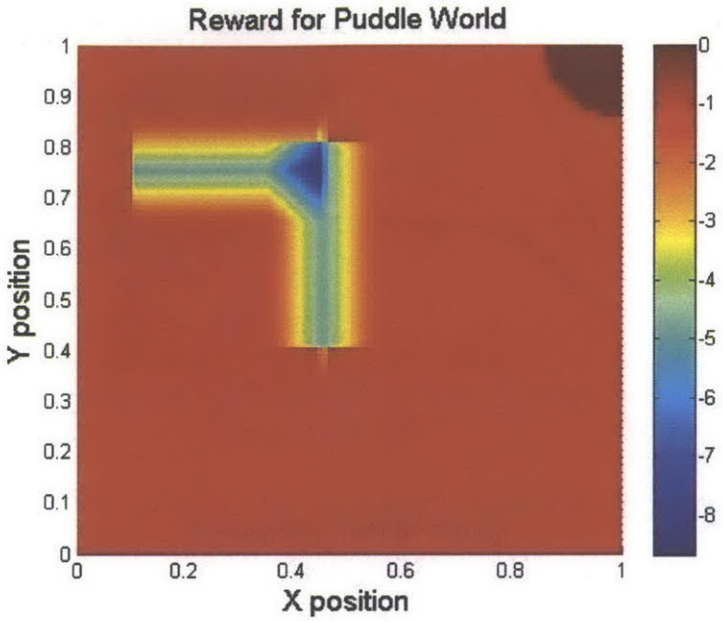


Figure 4-1: 2D depiction of the reward function for puddle world.

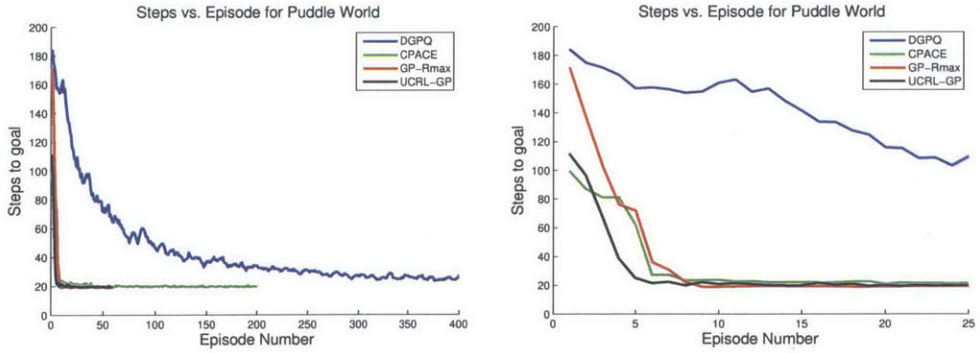


Figure 4-2: Learning curves for various PAC-MDP algorithms on the puddle world domain (number of steps to reach the goal vs. episode number. The optimal policy take approximately 20 steps to reach the goal. (left) The learning curve of all algorithms is visible. (right) The axes are enlarged to compare the speeds of the model-based methods.

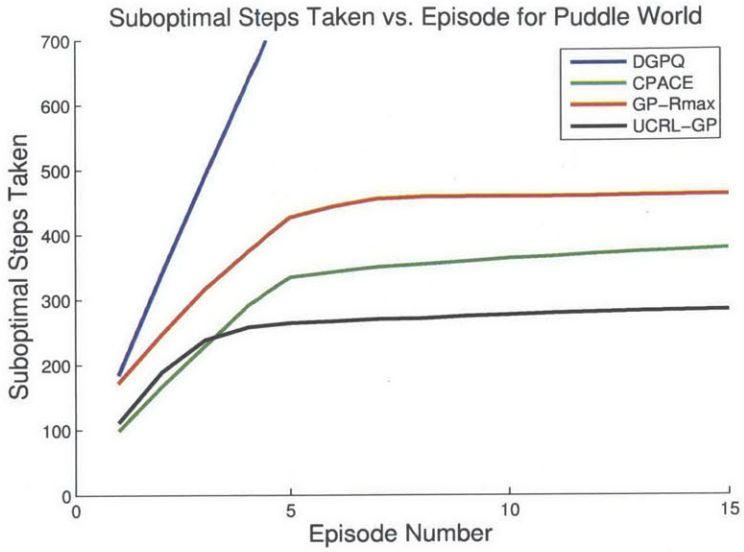


Figure 4-3: Number of suboptimal (exploratory) steps taken by various PAC-MDP algorithms to determine the optimal policy. UCRL-GP takes approximately 35% fewer steps than CPACE and 66% fewer steps than GP-Rmax.

Although Figure 4-2 shows similar performance in terms of the number of episodes to the optimal policy, UCRL-GP takes substantially less exploratory *steps* than the other algorithms. Figure 4-3 shows that UCRL-GP takes about 35% fewer suboptimal steps than CPACE and 66% fewer steps than GP-Rmax. While GP-Rmax treats uncertainty in the reward and transition functions as binary "known" or "unknown", UCRL-GP treats the uncertainty in a more nuanced fashion, which results in less suboptimal exploration steps. Additionally, C-PACE and DGPQ require tuning the Lipschitz constant which can dramatically affect the learning speed of these algorithms.

On the second domain, cart-pole balancing, the agent attempts to balance an inverted pendulum attached to a cart. The agent may apply a force of $F = \{-1, 0, 1\}N$ left or right on the cart in order to balance the pendulum. The parameters of the cart are mass of the cart $M = 0.5$ kg, mass of the pendulum, $m = 0.2$ kg, moment of inertia of the rod $I = 0.006$ kg m^2, and viscous damping $0.2\frac{\text{Ns}}{m}$. The reward signal is $r = -1 + \cos(\theta)$, where θ is the angle of the pole relative to the vertical line. The episode starts with the pole at $\theta = 0$ and ends when the height of the pole falls below $\pi/4$ radians below the horizontal, or when the episode length exceeds 400 timesteps. This system has four continuous dimensions cart

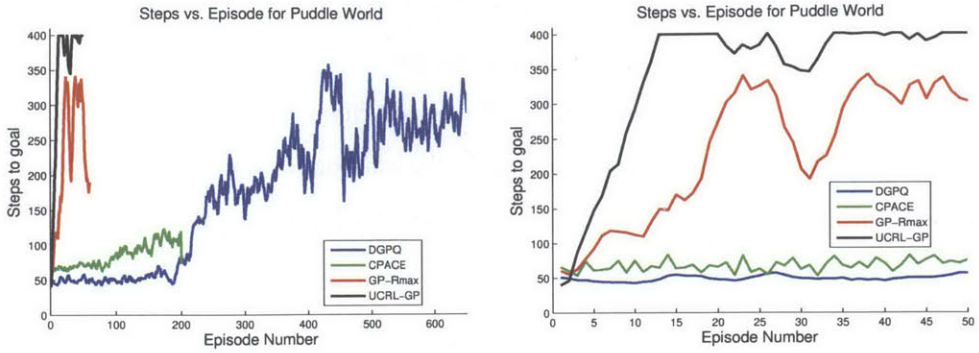


Figure 4-4: Learning curves for various PAC-MDP algorithms on the cart-pole balancing domain (number of steps that the algorithm can keep the pole from falling vs. episode number). The UCRL-GP algorithm can be seen to outperform the other methods by orders of magnitude in terms of learning speed. (left) The learning curve of all algorithms is visible. (right) The axes are enlarged to show the learning curve of UCRL-GP and GP-Rmax.

position x, velocity $\dot{x}$, pendulum angle θ, and angular rate $\dot{\theta}$. At each step the velocity $\dot{x}$ and angular rate $\dot{\theta}$ are perturbed with white Gaussian noise $\mathcal{N}(0, 0.01)$. The results of this are shown in Figure 4-4.

In this domain, UCRL-GP outperforms all of the methods even more dramatically than in the puddle world domain. Balancing the pole is a substantially more difficult control task than puddle world, and it is shown that while UCRL-GP starts learning the policy within the first few episodes, C-PACE and DGPQ takes hundreds of episodes. GP-Rmax starts learning the policy early on as well, but the learning speed is still substantially slower. Additionally, UCRL-GP is the only algorithm that is able to consistently balance the pole for the full length of the episode. The cart-pole domain is difficult because the system is highly unstable. Therefore, if an algorithm makes an incorrect action at any time instant, it may be difficult to recover and there is a high probability that the pole will fall. Additionally, to learn how to balance is made more difficult by the fact that in order to learn, the agent must explore more unstable regions of the state space in which the pole is close to falling.

4.3.2 Nonstationary Reward Functions

In this section, UCRL-GP-CPD is tested on two nonstationary reward function domains: puddle world with a moving puddle and a UAV pedestrian avoidance path planning prob-

lem. UCRL-GP-CPD is also compared to standard UCRL-GP to demonstrate the problem with using a simple stationary RL algorithm when there is non-stationarity in the reward or transition function. In the first application, the reward function changes everywhere such that it is sufficiently easy to detect a change in the environment without exploration. In the pedestrian avoidance application, however, the changes are more local, requiring an exploration strategy to see if anything has changed in the environment.

Simulated Domain: Nonstationary Puddle World

This experiment considers a variant on the puddle world experiment described in Section 4.3.1. The agent must again navigate a continuous planar world $S : [0, 1]^2$ from one corner $s = [0, 0]^T$ to another $s = [1, 1]^T$ while avoiding a puddle. At each step, the agent takes a step in one of the cardinal directions with magnitude $0.1 + \eta$, where $\eta \sim \mathcal{N}(0, 0.01^2)$ is white Gaussian noise. The agent starts in the corner $[0, 0]$ and ends when the agent reaches within 0.15 L_1 distance of the goal $[1, 1]$ or when the episode length exceeds 200 steps. In this case, the penalty associated with stepping in the puddle is modeled by a Radial Basis Function (RBF) $r = -2\exp\left(-\frac{\|s-\vec{p}\|^2}{0.125}\right)$, where s is the current position of the agent, and $\vec{p}$ is the center of the puddle. There are 4 MDPs in this world, with puddles at respective positions $\vec{p} \in \{[1, 0.5]^T, [0.5, 1]^T, [0, 0.5]^T, [0.5, 0]^T\}$. Every 20 episodes, the puddle moves to the next location, effectively switching the MDP.

UCRL-GP-CPD is able to identify changepoints quickly and reliably in this domain, as seen in Figure 4-5. This leads to improved learning speed overly naïvely relearning an MDP from scratch on every episode. As also seen in Figure 4-5, simply using a stationary RL algorithm, such as UCRL-GP, results in poor performance across many tasks. UCRL-GP is able to learn the optimal policy for task 1 (episodes $1 - 20$, $81 - 100$, and $161 - 180$), but performs quite poorly on the other tasks. The algorithm is unable to learn the new reward function, and learns a suboptimal policy as a result.

UCRL-GP-CPD is better suited to this sort of problem than Bayesian methods since the MDPs are not drawn i.i.d. from a distribution. That is, there is a strong correlation between receiving a MDP of one type and encountering it the next episode as well. This strong correlation makes Bayesian methods ill-suited for two main reasons. Firstly, if the

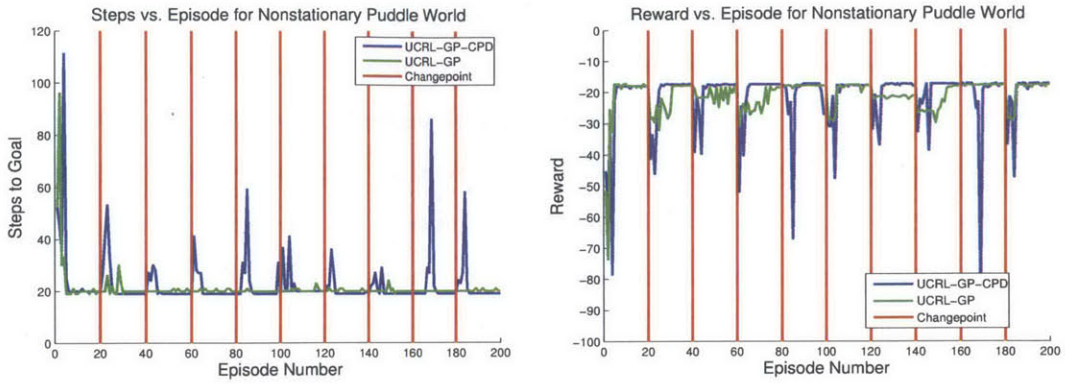


Figure 4-5: Steps to goal (left) and reward (right) per episode for nonstationary puddle world for UCRL-GP-CPD and UCRL-GP. UCRL-GP-CPD is able to accommodate changepoints and reuse previous models. This results is reliable changepoint detection and transfer of previous models to increase learning speeds of new MDPs. UCRL-GP-CPD is able to detect changes extremely fast.

prior density over MDPs is estimated correctly, then the probability of observing a new task goes to zero as the number of tasks drawn from MDP 1 are encountered. For example, if one encounters 100 episodes from a single MDP type, the probability density function of observing a new MDP is $p(\text{MDP} \neq 1) = \frac{\alpha}{100+\alpha}$, where α is typically set around values of 1 or less, when using the Dirichlet prior. Thus, it becomes increasingly difficult to determine a changepoint if the algorithm encounters many episodes of the same MDP. Secondly, Bayesian methods tend to overestimate the true number of MDPs, or components, logarithmically with the number of episodes [83]. That is, a Bayesian method using the Dirichlet prior will not tend to estimate the true prior density over tasks, but will tend to create many identical or similar models with different task labels.

4.3.3 Experimental Results

Problem Statement

In this section, UCRL-GP-CPD is demonstrated empirically on an real-world set-up in collaboration with the Distributed Autonomous Systems Lab (DAS) at Oklahoma State University. In this experiment, a variation of UCRL-GP-CPD is used in a Unmanned Aerial Systems (UAS) pedestrain avoidance problem. These results were originally pub-

lished in [2]. With loosening regulation from the FAA, UAS are expected to be deployed for several civilian applications, including package delivery, law enforcement and outdoor monitoring. However, most UAS are equipped with sensors such as cameras, which lead to growing concerns among the public that UAS on missions that require flying over inhabited areas could invade privacy by taking pictures of humans in private locations (see e.g. [46]). Human-aware UAS path planning algorithms that minimize the likelihood of a UAS flying over areas with high human density could potentially address this issue. Such algorithms would also be useful for covert military applications for conducting tactical missions without being detected, and in household robotics tasks for ensuring that robots do their jobs without interfering in human activity.

In order to create a human-aware UAS path, the UAS must be able to plan in anticipation of where humans could be. A naïve way of planning such paths might take into account available datasets of human population concentration as a function of built-up space, such as census maps maintained by counties or openly available maps such as those available from Google. However, such publicly available data may not be up to date and may not provide sufficient resolution to accurately predict population densities. Moreover, this approach fails to take into account the fact that human concentration is uncertain, and is often a function of the time of the day or the season.

Therefore, rather than relying solely on historic datasets, it is beneficial that the UAS maintain a real-time updated model of the population density. Such a model will inherently be non-stationary to account for time-dependent variations. Additionally, human behavior and density patterns often recur. For example, business districts in a city are busier on workdays. Ideally, if the UAS encounters a density pattern that it has seen before, it should be able to reclassify it as a previously-seen model and leverage a policy that it has already computed.

The problem is formulated as a nonstationary MDP, and a variation of UCRL-GP-CPD is used to solve the path planning problem. This variation does not use optimistic value iteration for exploration but still uses GP-NBC's changepoint detection and model transfer. This variation is referred to as GPC throughout this section. GPC learns and maintains a separate model for each distinguishable distribution of human density. While

in section 4.1, a simple counter is used to determine when to run an exploration strategy, a function called the *fog of war* is used is this section to determine when to explore. The addition of the fog of war function encourages the UAS to explore regions which it has not explored *recently*. This addition is crucial to ensure that the agent is able to detect changes in the environment.

In order to perform exploration, a new MDP is created using an *exploration reward* $\hat{r}_{e_{t_i}}$:

$$\hat{r}_{e_{t_i}}(x) = \hat{\Sigma}_{t_i}(x) + \Upsilon_{t_i}(x), \tag{4.17}$$

where $\hat{\Sigma}_{t_i}(x)$ is the covariance of the GP over $\mathcal{S}$, and Υ_{t_i} is the *fog of war* function. The latter increases the predictive variance as a function of time, so as to induce a tunable forgetting factor into (4.17). This functional is defined in this manner for maximal generality; a very simple choice however, and the one used in this section is $\Upsilon_{t_i}(x) := \max(C_{fow_1}(t - t_s), C_{fow_2})$, where t_s is the last time the model switched, and $C_{fow_1}, C_{fow_2} \in \mathbb{R}_+$ are user-defined constants. In order to determine when to switch to exploration, the quantity

$$s_e(t_i) = \kappa - \frac{1}{\mathrm{vol}(D)} \int_D \hat{r}_{e_{t_i}}, \tag{4.18}$$

is computed, where κ is the largest value of the variance (1 for the Gaussian kernel) and $D \subset \mathcal{S}$. The quantity s_e is a measure of the space explored at an instant t_i. If this is above a certain threshold, the agent should only exploit its knowledge of $\mathcal{S}$ instead of exploring. The use of predictive variance for exploration is similar to "knownness" based model-free MDP solvers [11] and information entropy maximizing active sensor placement algorithm [71].

Description of the Experiment

The mission scenario is for the agents to go from a predefined ingress location to a predefined egress location (referred to as the goal location) in the domain. The agent is free to choose the optimal path over the domain to perform its mission. Each path planning and execution instance is termed as a run. A large-scale experiment with a total of $5,500$ runs

across the environment is performed. During the experiment, the agents face four different population densities. The population densities switch periodically every 200 runs for the first 4 times for each model, and then randomly. The agents do not have an a-priori model of the underlying human population densities, nor do they know that these are expected to switch or know the total number of underlying models. Furthermore, stochasticity in transitions is induced by any tracking error the Quadrotor controller may have. The rewards samples are stochastic draws from the corresponding GP.

The urban arena is split into grids of dimension 50×50, and the agent action set consists of a decision to move to any directly adjacent grid. An efficient policy-iteration based planner [18] is used to compute the value function at the beginning of each run using the learned reward model. The resulting policy is further discretized into a set of waypoints, which are then sent to the agent as the desired path. To accommodate a large-number of runs $(5,500)$, simulated UAV agents are used in the experiment. The first four times the population density model is switched, 15 runs are performed by the real-UAV. Hence, the real UAV performs a total of 75 runs across the domain, with each run taking around 2 minutes from take-off to landing. Furthermore, since these runs are early in the learning phase of each model, the paths flown by the UAV are exploratory, and hence longer. The data collected by each agent is assumed to be fed to a centralized non-stationary MDP based planner, which does not distinguish between real and simulated agents. The simulated agents use physics based dynamics model for a medium sized Quadrotor and quaternion based trajectory tracking controllers [36, 55]. The real-UAVs fly in an experimental testbed; the testbed area is 16×12 ft and it is equipped with the Optitrack motion capture system and is designed to conduct real-time testing in an emulated urban-environment. The Optitrack cameras have a optical range between 18in to 433in, thus allowing an estimation volume of $11 \times 11 \times 4$ ft. The testbed consists of wooden buildings and several cars (all cars not shown in figure) that can move around, allowing us to simulate different population densities. The Optitrack is a set of ten infra-red cameras that collect data based on the markers attached to the body. The testbed further includes a monitoring facility with a Mobotix Q24 fish eye camera. The UAV used in this experiment is the a AR-Drone Parrot 2.0 Quadrotor controlled by a client program using the Robot Operating System (ROS) [101].

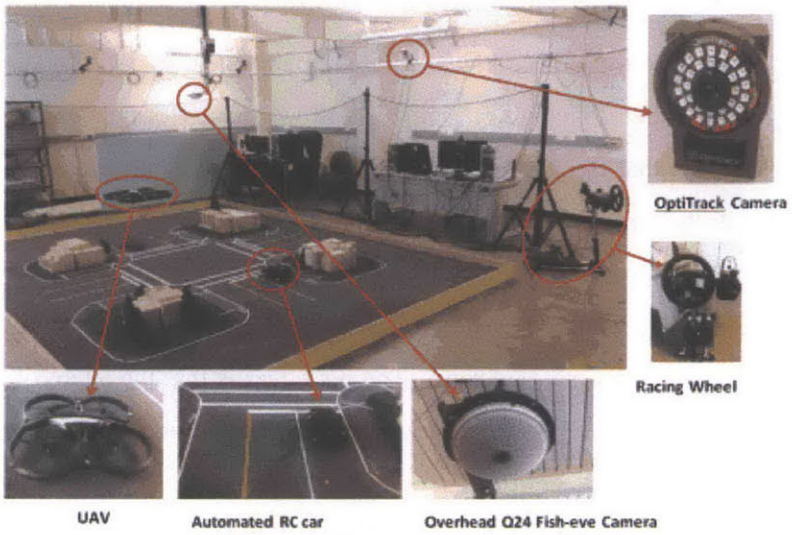


Figure 4-6: The flight-test bed at the DAS lab at OSU emulates an urban environment and is equipped with motion capture facility.

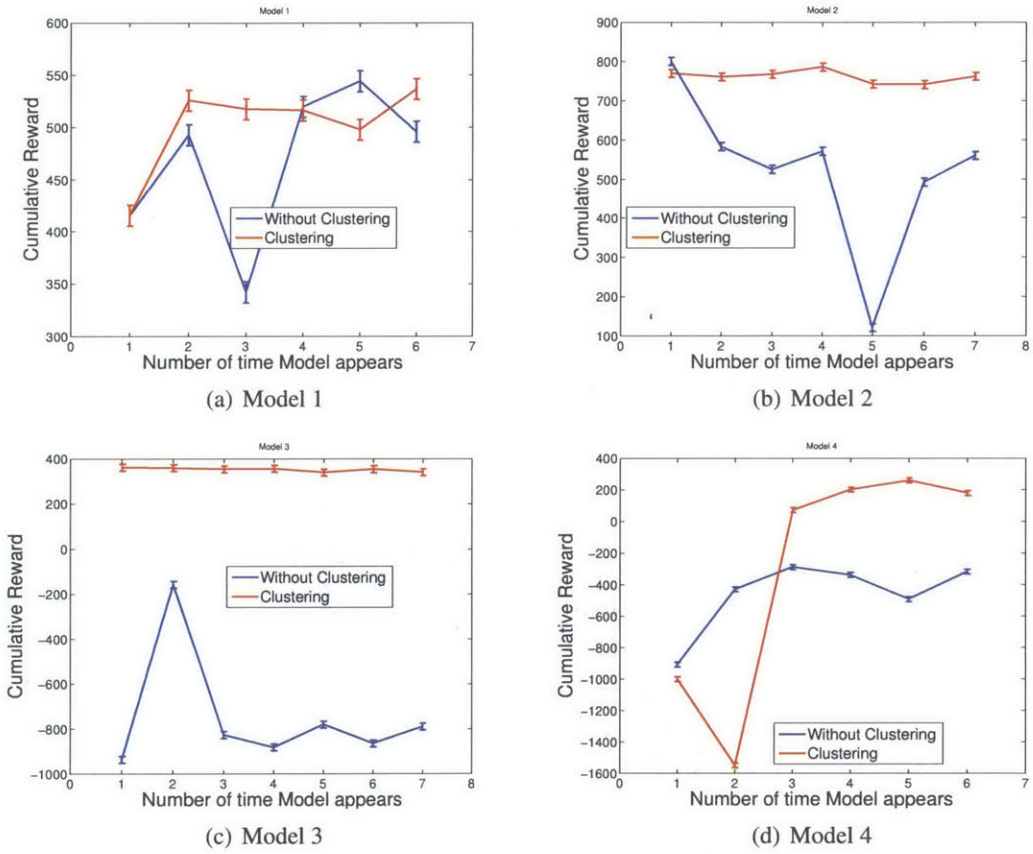
Discussion of Results

In the experiments, GPC is compared to RL using a GP representation but without any changepoint detection or model transfer. This algorithm is referred to as GPR (GP Regression). Figure 4-7 presents the average reward accumulated and its variance by both the planners for each of the four models. The horizontal axis indicates the number of times each model is active and the vertical axis indicates the cumulative reward the agent has accumulated over the time the underlying model was active. In model 1, both algorithms perform very similarly. One reason for this parity is that GPR tends to learn model 1 quickly as this is the model it was initialized in. In fact, the GPR predictive co-variance reduces heavily while learning this model for the first time, and even with the fog-of-war forgetting, the co-variance does not increase again. This results in the GPR (falsely) being confident in its model, and even though it is updated online with new reward samples, its estimate is highly biased towards the first model. However for models 2, 3, and 4, the clustering based GPC algorithm is clearly seen to have a better performance characterized by the consistency over which the GPC based planner finds the near optimal policy. Indeed it can be observed by noticing that total reward accumulated over the entire duration are

nearly constant, with the small variations being attributed to the stochasticity in the reward.

Figure 4-8 indicates total reward accumulated by the agent in each iteration, when the underlying reward model is changing. Figure 4-9 shows the performance of the GPC algorithm. It can be seen that the algorithm rapidly learns the underlying model, and quickly identifies whether the underlying model is similar to one it has learned before. As a result, the agent's estimate of the reward model converges quickly for each model. Furthermore, because the agent can recognize a previous model that it has already learned before, it does not have to spend a lot of time exploring. The net effect is that the agent obtains consistently high reward, and a long-term improvement in performance can be seen. Whereas, when the agent follows the non clustering based traditional GPR approach, it takes a longer time to find the optimal policy leading to a depreciation in its total accumulated reward over each model.

Exploration of the domain is required by any reinforcement learning algorithm to ensure that it obtains sufficient information to compute the optimal policy [15]. In this architecture, the exploration strategy is to compute a policy that guides the agents to areas of the domain with high variance, rather than using the estimate of the GP reward model. However, relying on the GP variance is not enough, because the GP variance updates do not take into account the non-stationarity in the domain. The fog of war forgetting introduced in (4.17) adds another metric on learned model confidence by ensuring that the agent revisits parts of the domain that it has not recently visited. However, exploration is costly, because this is when the agent is likely to accumulate negative reward. Therefore, the optimal long-term strategy should be to minimize exploration and maximize exploitation by identifying similarities in the underlying models. It becomes clear therefore that the performance of the GPC-based planner is superior because the GPC algorithm is able to cluster the underlying reward model with ones that it has seen before, and hence does not need to explore as much. The estimation performance of the GPC algorithm is visible by comparing the real and estimated models in Figures 4-10. This plot indicates that the algorithm did not spend time exploring areas where it cannot perform optimally. Furthermore, Figure 4-11 plots the value of the exploration reward (4.18) and the exploitation threshold (0.85). This figure shows that the GPC planner spends less time exploring. In fact, the dips in Figure 4-11



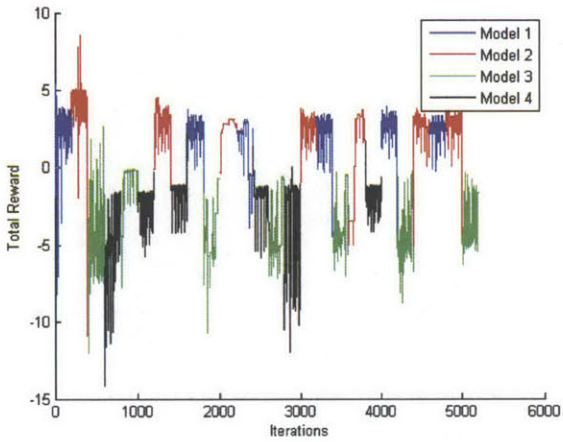
(a) Model 1

(b) Model 2

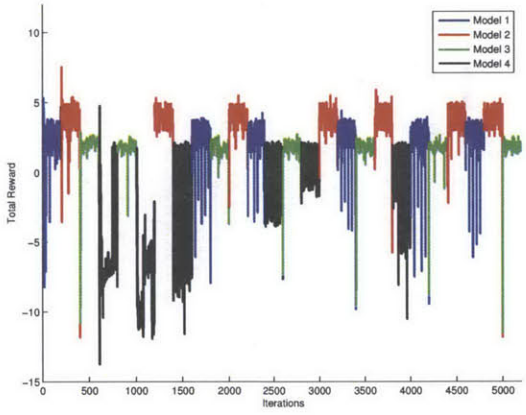
(c) Model 3

(d) Model 4

Figure 4-7: Comparison between Total Rewards for each model



(a) Cumulative Reward without Clustering



(b) Cumulative Reward with Clustering

Figure 4-8: Comparison of the accumulated rewards of GP clustering versus GPRegression; the agent accumulates more positive rewards when clustering models.

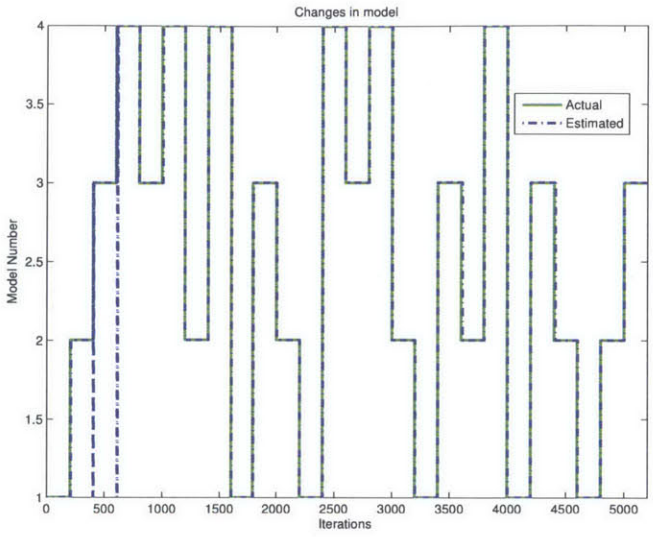
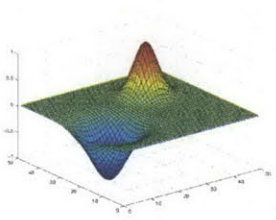


Figure 4-9: Plot indicating the actual model being tracked by the estimated model. At the 200^{th} and 400^{th} run new models are introduced, the algorithm quickly detects them after a brief misclassification. Note that the algorithm clusters the underlying reward model quickly and reliably afterwards.

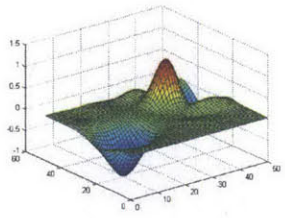
match the times when the agent encounters a new model for the first time. These step dips are desirable, because they indicate that the algorithm has detected that a new model has been encountered, and that it is likely to sufficiently explore the domain to learn that model. In contrast, only one such dip is seen at the beginning for the GPR based planner.

4.4 Conclusions

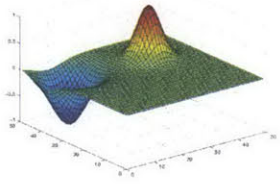
This chapter investigated the problem of RL when the reward function is non-stationary, i.e. may experience changepoints. First, Section 4.1 introduces the UCRL-GP algorithm for continuous RL with stationary reward functions. It is proven in Section 4.2.1 that the UCRL-GP algorithm is PAC-MDP, one of the first continuous RL algorithms with sample complexity guarantees. In Section 4.3.1, UCRL-GP outperforms state of the art RL algorithms in terms of sample complexity on several RL domains. This section secondly introduces the UCRL-GP-CPD algorithm for nonstationary reward functions. Section 4.2.2 proves that UCRL-GP-CPD will detect changes in the environment with high probability and that it is theoretically guaranteed against negative transfer. Section 4.3.2 demonstrated



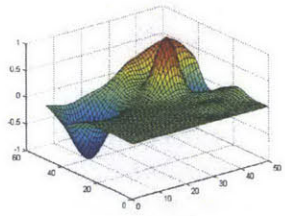
(a) Model 1 Actual



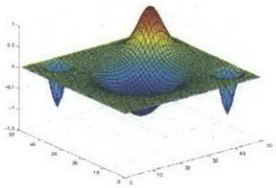
(b) Model 1 Estimated



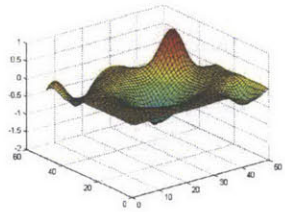
(c) Model 2 Actual



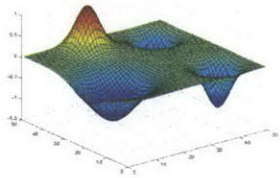
(d) Model 2 Estimated



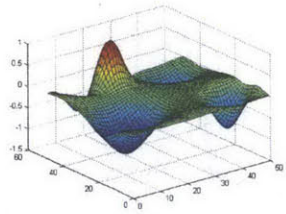
(e) Model 3 Actual



(f) Model 3 Estimated

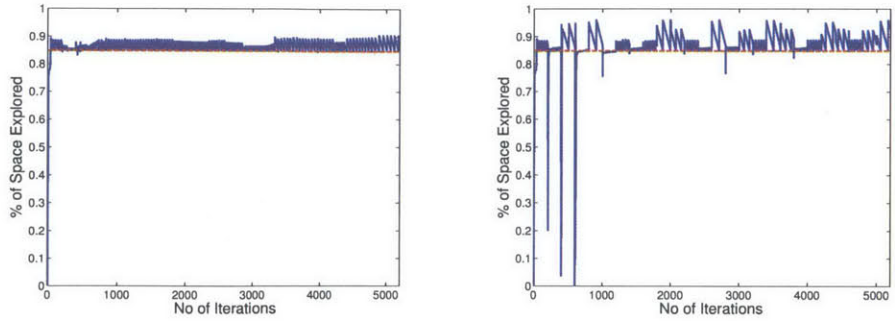


(g) Model 4 Actual



(h) Model 4 Estimated

Figure 4-10: Estimated and actual mean of a Gaussian Process reward generative model.



(a) Space Explored GPR (b) Space Explored GPC

Figure 4-11: Space Explored by each planner indicative of the variance

UCRL-GP-CPD empirically on a simulated and real-life domain.

Future work will aim at extending UCRL-GP to domains with stochastic dynamics. Planning in continuous spaces with stochastic dynamics is extremely difficult as it 1) requires taking the expected value of a continuous function with respect to an arbitrary distribution and 2) requires modeling an arbitrary p.d.f. which is parameterized by continuous variables. However, if certain structural assumptions are made on the transition dynamics, such as Gaussianity, it is possible to perform 1) in closed form with a Gaussian process representing the reward function [37]. Thus, while it is possible to perform planning with stochastic dynamics, the future steps are non-trivial.

Chapter 5

Conclusions and Future Work

The first part of this thesis introduces a novel algorithm, GP-NBC, for online prediction and learning in the presence of nonstationary data using Gaussian Processes. It was proven that GP-NBC is theoretically justified and will make a bounded number of erroneous predictions per phase, a property not shared by other algorithms using GPs. It was shown in several synthetic as well as real-life domains that GP-NBC requires orders of magnitude less computation than existing state of the art methods, while still providing accurate predictions. Additionally, unlike existing online methods, GP-NBC is capable of learning models and transferring past model knowledge when it becomes applicable again. The ability of GP-NBC to learn models and make accurate predictions in real-time makes it well-suited for application domains in Reinforcement Learning (RL), Adaptive Control, and decision making.

The second part of this thesis utilizes GP-NBC in a novel RL algorithm for continuous domains, UCRL-GP-CPD. First, an algorithm for stationary reward functions, UCRL-GP, is introduced. It is proven that UCRL-GP will learn an ϵ-optimal policy in a polynomial number of steps, i.e. it is PAC-MDP. UCRL-GP is one of the first algorithms for continuous RL which has been proven to be PAC-MDP. In the empirical section, it was shown that UCRL-GP learns the optimal policy in fewer exploratory steps than the state of the art RL algorithms for continuous domains.

GP-NBC is then combined with UCRL-GP to make a new algorithm, UCRL-GP-CPD, which is capable of detecting changes in the reward function and determining a policy when

necessary. Since changes in the reward function may be local, UCRL-GP-CPD employs a periodic exploration strategy to check to see if anything has changed in the environment. Using this periodic exploration, it was proven that if a changepoint occurs in the environment, UCRL-GP-CPD will detect it will high probability. Unlike existing algorithms, UCRL-GP-CPD does not make any distributional assumptions about the class of problems it may encounter, is capable of detecting changes in the environment at arbitrary points in time, and does not require a training phase in which all problems must be previously encountered. In the empirical section, UCRL-GP-CPD is tested on several domains such as a puddle world with moving puddle and UAV path planning while avoiding pedestrians. It was shown than UCRL-GP-CPD can successfully detect changes in the environment and transfer past model knowledge when applicable.

5.1 Discussion and Future Work

While a significant number of algorithms for nonstationary data have been recently proposed using Bayesian problem formulations, nonBayesian algorithms have been significantly less researched. Bayesian algorithms have theoretic and intuitive foundations, but the computation required to compute posterior distributions is often prohibitive for applications which require real-time decision making. Additionally, Bayesian algorithms require a prior belief over the distribution of changepoints. If a process has a well-defined transition structure that can be garnered from first principles, such as the Poisson process, then a Bayesian method will work well. However, in many applications, such as fault detection or pedestrian behavior analysis, it may be expensive or impossible to delineate a good prior function. As a result, this lack of a good prior often leads to ad-hoc feature selection, which can lead to poor performance.

NonBayesian algorithms, on the other hand, do not require a prior distribution, and are formulated instead on a notion of the probability of detecting an event and the probability of a false detection, which can be determined from the observation model alone. This reformulation results in a hard decision rule on changepoint detection rules instead of a soft posterior distribution. While there are some losses associated with hard decision rules,

a nonBayesian test for changepoints results in a scalable solution which takes orders of magnitudes less computation. A more thorough analysis and experimentation of Bayesian and nonBayesian methods in changepoint detection would be helpful to delineate what kinds of problems are better handled by one algorithm over another. In applications where changes are frequent or in which the changepoint times are known, it is likely that the Bayesian methods will perform better. However, in applications in which the changes are infrequent, this may violate the i.i.d. assumptions of the Bayesian formulation and cause problems.

Lastly, the theoretical guarantees derived in this thesis are based on the PAC framework, which is closely related to frequentist approaches. These bounds, however, are notoriously loose, and are more interesting for asymptotic analysis and theoretical justification of algorithms rather actual use in practice. PAC-Bayesian [80] is a class of methods which employ priors and perform posterior inference, making them Bayesian, but also employ some aspect of PAC proofs to theoretically prove that an algorithm is correct. It would be interesting to develop PAC-Bayesian versions of the GP-NBC and UCRL-GP-CPD algorithms to leverage the benefits of both schools of statistics.

Appendix A

Proofs

A.1 KL-divergence between estimated model and new distribution

In the case of online learning, H_1 is not known a priori and must be approximated using a subset of the current data. In this case, the LRT will not tend towards the KL-divergence of the true distributions, but rather it will tend towards,

$$\mathbb{E}[L_{H_1|H_0}(y)] = D(H_1\|H_0) - D(H_1\|\hat{H}_1) \tag{A.1}$$

which is the KL-divergence between the current model and the true distribution minus the approximation error of using a model $\hat{H}_1$ instead of H_1.

Proof

$$\begin{aligned}
\mathbb{E}[L_{H_1|H_0}(y)] &= \int_{p(y|H_1)} p(y \mid H_1) \log \frac{p(y \mid \hat{H}_1)}{p(y \mid H_0)} \\
&= \int_{p(y|H_1)} p(y \mid H_1) \log \frac{p(y \mid \hat{H}_1)}{p(y \mid H_1)} + \int_{p(y|H_1)} p(y \mid H_1) \log \frac{p(y \mid H_1)}{p(y \mid H_0)} \\
&= D(H_1\|H_0) - D(H_1\|\hat{H}_1)
\end{aligned}$$

A.2 Role of the Prior in Lemma 3.1

This section describes the affect of the GP prior on the learning result in Lemma 3.1.

While priors may improve early model performance, they may also slow learning if the initial estimate is far from the actual value. In order to consider the effect of the prior on GP inference, we examine the bias versus variance properties of the GP estimator for properties of the model. The mean function of a GP is equivalent to the MAP estimate of a linear combination of Gaussian beliefs, given by,

$$\hat{\mu} = \frac{\sigma_0^2}{\sigma^2 + \sigma_0^2} f(y_1, \ldots, y_n) + \frac{\sigma^2}{\sigma^2 + \sigma_0^2} \mu_0 \tag{A.2}$$

where μ_0, σ_0^2 are the prior mean and variance, respectively. $f(\cdot)$ is a function which maps the observations to a estimate. In the single variable case, this corresponds to an average. In Lemma 3.1, this corresponds to the function $\mu(x') = f(x_1, \ldots, x_n)$ which was analyzed. Therefore, the total error induced by using a GP is given by the sensitivity of $f(\cdot)$ to noise, which was analyzed in Lemma 3.1, and the error induced by having a prior bias. Here, we show that this bias-error is small, and can be effectively ignored in analysis.

The maximum error due to regularization is given by the upper bound on the second term

$$\epsilon_{pr} = \frac{\sigma^2}{\sigma^2 + \sigma_0^2} V_m \tag{A.3}$$

Plugging in σ_{tol} from Lemma 3.1,

$$\epsilon_{pr} = \frac{\omega_n^2 \epsilon_1^2 V_m}{\omega_n^2 \epsilon_1^2 + \frac{1}{2} V_m^2 \log(\frac{2}{\delta_1}) \sigma_0^2} \tag{A.4}$$

Defining $r = \frac{\sigma_0^2}{\omega_n^2}$,

$$\epsilon_{pr} = \frac{V_m}{1 + \frac{r}{2\epsilon_1^2} V_m^2 \log(\frac{2}{\delta_1})} \tag{A.5}$$

Interestingly, the role of the prior decreases as V_m, $\frac{1}{\epsilon_1}$, and $\frac{1}{\delta}$ increase. This is due to the fact that the variance required to average out noise scales as V_m^2 whereas the error induced by bias of the prior scales V_m. The effect of the prior increases as the ratio r decreases, i.e. our initial variance is very low, or the measurement noise is high. We argue that by

increasing r, we can make the error due to regularization arbitrarily small without effecting the reliability of our estimate. For example, by increasing σ_0^2 arbitrarily large, we have,

$$\hat{\mu} \approx \frac{\sigma_0^2}{\sigma^2 + \sigma_0^2} f(y_1, \ldots, y_n) \tag{A.6}$$

Effectively, by making the prior variance arbitrarily large, we have turned the GP into a function approximator which does not rely on any prior information. Therefore the guarantees still hold from the previous section. By creating a large prior variance, one might be concerned that the rate at which σ^2 decays will be slowed down. However, consider that the covariance at a point after a single observation is given by $\sigma^2 = (\sigma_0^{-2} + \omega^{-2})^{-1} \approx= \omega^2$. Thus, for any $\sigma_0^2 >> \omega^2$, the effect of the prior variance does not matter much. Since we can set r arbitrarily large by scaling the kernel function, we can therefore scale ϵ_{pr} to be arbitrarily small. Therefore, we neglect the role of the prior in all future analysis.

A.3 Proof that the variance decreases in the Voronoi region: Theorem 1

An ϵ_t-volume around a point $\bar{x}$ is defined as the set of all points which satisfy the following distance metric as $\{\epsilon_t - \text{Vol} : x \in S \mid k(\bar{x}, \bar{x}) - k(\bar{x}, x)^T K(x, x)^{-1} k(\bar{x}, x)\}$. Define the correlation coefficients between two points as $\rho = k(x_i, x_j)$. Using Bayes law, it can be shown that given a point $\bar{x}$ with prior uncertainty $\sigma_0^2 = 1$ and m measurements at another location x_i with correlation coefficient ρ, the posterior variance is given by $\sigma_n^2 = 1 - \frac{n\rho^2}{n+\omega^2}$. Using the linear independence test, we relate ϵ_t to ρ as $\epsilon_t = 1 - \rho^2$. Therefore, we have that in an ϵ_t-volume, the slowest that the variance can reduce at the center of the volume $\bar{z}$ is given by,

$$\sigma_n^2 \leq \frac{n\epsilon_t + \omega^2}{n + \omega^2} \leq \frac{n\epsilon_t + \omega^2}{n} \tag{A.7}$$

The ϵ_t-volume around a point $\bar{x}$ is identical to the voronoi region around a point in the covering set. $\epsilon_t \leq \frac{1}{2}\sigma_{tol}^2$ by the definition of the covering number.

Appendix B

Experimental Validation of Bayesian Nonparametric Adaptive Control using Gaussian Processes

This appendix presents a framework for Model Reference Adaptive Control (MRAC) using Gaussian Processes as well as extensive empirical results using this framework on a quadrotor. This work was initially published in [49] and represents work that I have done during my Masters which did not fit in thematically with the main text of the thesis.

B.1 Abstract

Many current model reference adaptive control (MRAC) methods employ parametric adaptive elements in which the number of parameters are fixed a-priori and the hyperparameters, such as the bandwidth, are pre-defined, often through expert judgment. As an alternative to these methods, a nonparametric model using Gaussian Processes (GPs) was recently proposed. Using GPs, it is possible to maintain constant coverage over the operating domain by adaptively selecting new kernel locations as well as adapt hyperparameters in an online setting to improve model prediction. This work provides the first extensive experimental flight results using GP-MRAC. Experimental results show that GP-MRAC outperforms traditional MRAC methods which utilize Radial Basis Function (RBF) Neural Networks

(NNs) in terms of tracking error as well as transient behavior on trajectory following using a quadrotor. Results show an improvement of a factor of two to three over pre-existing state of the art methods.

Additionally, many MRAC frameworks treat the adaptive element as being known exactly, and do not incorporate certainty of the prior model into the control policy. This appendix also introduces the notion of a *Bayesian scaling factor* which scales the adaptive element in order to incorporate the uncertainty of the prior model and current model confidence. The stability and convergence are proven using the Bayesian scaling factor in closed loop.

B.2 Introduction

In several aerospace and mechanical applications, it is often difficult or infeasible to obtain an exact model of the system dynamics, either due to cost or inherent model complexity. Model Reference Adaptive Control (MRAC) is a widely studied adaptive control methodology for Aerospace applications which aims to ensure stability and performance in presence of such modeling uncertainty. MRAC has been implemented on several experimental and in-service aerospace flight platforms, including the Georgia Tech GT-MAX [58], the J-DAM guided munition [110], F-36 aircraft [111], the X 33 launch vehicle [59]. It has also been used for fault-tolerant adaptive control research in the context of adaptive control in presence of loss of lifting surface or actuator uncertainties [22].

Several active directions in MRAC research exist, such as [17, 31, 73, 91, 108, 130], and a central element of these and many other MRAC architectures is a parametrized adaptive element, the parameters of which are tuned online by the MRAC architecture to capture the modeling uncertainty. A common approach for synthesizing the adaptive element is to use a deterministic weighted combination of a set of basis functions to estimate the associated parameters online. This approach has the benefit that it allows the designer to encode any physical knowledge about the modeling uncertainty within the MRAC framework. However, in many aerospace or mechanical problems of interest, the exact form of the adaptive element may not be available a priori, especially when the underlying uncer-

tainties are a result of highly stochastic effects such as unsteady aerodynamics, or nonlinear dynamics such as actuator limitations. To handle such "unstructured uncertainties" authors have previously proposed Neural Network based approaches. Indeed, following the pioneering work by Sanner and Slotine [107], Radial Basis Function Networks (RBFNs also referred to as RBF-Neural Networks) are perhaps the most widely used adaptive elements when a basis for the modeling uncertainty is unknown [67, 88, 120]. Unlike multi-layer perceptron Neural Networks [67, 75], the RBFNs are linear-in-parameters universal function approximators. The accuracy of an RBFN representation, however, greatly depends on the choice of RBF centers [93]. Typically, authors have assumed that the operating domain of the system is known and have pre-allocated a fixed quantity of Gaussian RBF centers over the presumed domain [23, 67, 88, 94, 125]. However, if the system operates outside of the domain of the RBF kernel's centers [93], or if the bandwidth of the kernel is not correctly specified [86, 109, 117], the modeling error will be high and stability cannot be guaranteed globally. In fact, RBFN based stability results require that the system operate enforce constraints such that the system does not leave a pre-specified compact domain. Another key limitation or RBFNs, and other NN based universal approximators, is that the update laws assume a deterministic representation of the uncertainty, when available measurements are actually stochastic. It is well-known that without a projection operator, gradient based update laws subject to noisy measurements can result in model error growing unbounded. Even with a projection operator, noisy measurements can result in over-learning or oscillatory behavior. Additionally, without a probabilistic representation of the uncertainty, it is difficult to gage the level of confidence of the adaptive element for use in a control law.

To overcome the aforementioned limitations of RBFNs and other fixed-parameter adaptive elements, Gaussian Process (GP) Bayesian nonparametric adaptive elements are used [20, 27, 28], creating a new class of BNP adaptive control in an architecture called GP-MRAC. Unlike traditional MRAC, GP-MRAC is nonparametric and makes no assumptions about the operating domain. Using only data available online, GP-MRAC is able to dynamically choose new kernel locations to maintain domain coverage as well as learn relevant kernel hyperparameters. The main benefits of GP-MRAC Bayesian nonparametric adap-

tive controllers include: no prior knowledge of the operating domain of the uncertainty is required, measurement noise is explicitly handled, and parameters such as the centers of RBFs need not be pre-allocated. GPs utilize a Bayesian framework which models uncertainties as a *distributions over functions*, which differs from the traditional deterministic weight-space based approaches [67, 68, 88, 125]. Furthermore, Bayesian inference in GP-MRAC overcomes the shortcomings of the standard gradient based MRAC parameter update laws, such as the lack of convergence guarantees and the possibility of bursting (parameters growing unboundedly) in presence of noise [4, 10, 87]. Efficient "budgeted" online algorithms were presented that ensure tractable computational complexity for online inference and efficient methods for optimizing hyperparameters online while ensuring stability. Furthermore, it was shown that GP-MRAC can be used for adaptive control in presence of time varying uncertainties[26].

This work presents results from an extensive flight-test study of GP-MRAC on a Quadrotor UAV in different flight conditions. Several trajectory following experiments demonstrate that GP-MRAC significantly outperforms traditional MRAC using fixed-parameter RBF-NNs in terms of tracking error, transient performance, and learning model uncertainty. GP-MRAC is also compared with the previously flight-tested Concurrent Learning MRAC (CL-MRAC) algorithm [24, 25, 30], as well as a variant of GP-MRAC which automatically learns hyperparameters in an online setting. The experiments show that the ability to dynamically place new kernel centers enables GP-MRAC to maintain excellent tracking performance even when the system operates outside the expected domain. On the contrary, it is shown that when using traditional RBF-NN adaptive elements, if the system leaves the intended operating domain due to overshooting or stochasticity, instability can arise.

Furthermore, this work present a novel modification to GP-MRAC called the *Bayesian scaling factor* which weighs the relative confidence of the prior model with that of the adaptive element model. This results in an learning-focused MRAC scheme which trusts the baseline conservative controller more when first learning the model uncertainty and entering into new regions of the operating domain, and trusts the representation of the adaptive element more when the confidence of the learned model, based on the GP predictive vari-

ance, is high. This represents a fundamental shift from many traditional adaptive control policies which assume that the adaptive element is a known deterministic quantity when feeding it back into the control policy. Using a Bayesian scaling factor, the relative confidence in multiple sources of information is weighed appropriately, and a control policy is computed relative to this confidence. The combined effect is a smoother transition from conservative baseline control to aggressive adaptive control based on the learned model. It should be noted that preliminary flight test results over a limited experiment were presented in a conference paper [19].

B.3 Approximate Model Inversion based Model Reference Adaptive Control

AMI-MRAC is an approximate feedback-linearization based MRAC method that allows the design of adaptive controllers for a general class of nonlinear plants (see e.g. [16, 58]). The GP-MRAC approach is introduced in the framework of AMI-MRAC, although it should be noted that it is applicable to other MRAC architectures (see e.g. [4, 57, 89, 121]). Let $x(t) = [x_1^T(t), x_2^T(t)]^T \in D_x \subset \mathbb{R}^n$, such that $x_1(t) \in \mathbb{R}^{n_s}$, $x_2(t) \in \mathbb{R}^{n_s}$, and $n = 2n_s$. Let $\delta \in D_\delta \subset \mathbb{R}^l$, and consider the following multiple-input controllable control-affine nonlinear uncertain dynamical system

$$\begin{aligned} \dot{x}_1(t) &= x_2(t), \\ \dot{x}_2(t) &= f(x(t)) + b(x(t))\delta(t). \end{aligned} \tag{B.1}$$

The functions $f(0)$, $f(0) = 0$ and b are partially unknown functions assumed to be Lipschitz over a domain $\mathcal{D}$ and the control input δ is assumed to be bounded and piecewise continuous, so as to ensure the existence and uniqueness of the solution to (B.1) over $\mathcal{D}$. Also assume that $l \leq n_s$ (while restrictive for overactuated systems, this assumption can be relaxed through the design of appropriate control assignment [42]). Further note that while the development here is restricted to control-affine systems, sufficient conditions exist to convert a class of non-affine in control nonlinear systems to the control-affine form in (B.1)

(see Chapter 13 in [65]), and the AMI-MRAC framework can also be extended to a class of non-affine in control systems [64, 66].

The AMI-MRAC approach used here feedback linearizes the system by finding a pseudo-control input $\nu(t) \in \mathbb{R}^{n_s}$ that achieves a desired acceleration. If the exact plant model in (B.1) is known and invertible, the required control input to achieve the desired acceleration is computable by inverting the plant dynamics. Since this usually is not the case, an approximate inversion model $\hat{f}(x) + \hat{b}(x)\delta$, where $\hat{b}$ chosen to be nonsingular for all $x(t) \in D_x$, is employed.

Given a desired pseudo-control input $\nu \in \mathbb{R}^{n_s}$ a control command δ can be found by approximate dynamic inversion:

$$\delta = \hat{b}^{-1}(x)(\nu - \hat{f}(x)). \tag{B.2}$$

Let $z = (x^T, \delta^T)^T \in \mathbb{R}^{n+l}$ for brevity. The use of an approximate model leads to modeling error Δ for the system,

$$\dot{x}_2 = \nu(z) + \Delta(z), \tag{B.3}$$

with

$$\Delta(z) = f(x) - \hat{f}(x) + (b(x) - \hat{b}(x))\delta. \tag{B.4}$$

Were b known and invertible with respect to δ, then an inversion model exists such that the modeling error is not dependent on the control input δ. A designer chosen reference model is used to characterize the desired response of the system

$$\begin{aligned} \dot{x}_{1_{rm}} &= x_{2_{rm}}, \\ \dot{x}_{2_{rm}} &= f_{rm}(x_{rm}, r), \end{aligned} \tag{B.5}$$

where $f_{rm}(x_{rm}, r)$ denotes the reference model dynamics, assumed to be continuously differentiable in x_{rm} for all $x_{rm} \in D_x \subset \mathbb{R}^n$. The command $r(t)$ is assumed to be bounded and piecewise continuous. Furthermore, f_{rm} is assumed to be such that x_{rm} is bounded for a bounded command.

Define the tracking error to be $e(t) = x_{rm}(t) - x(t)$, and the pseudo-control input ν to

be

$$\nu = \nu_{rm} + \nu_{pd} - \nu_{ad}, \tag{B.6}$$

consisting of a linear feed-forward term $\nu_{rm} = \dot{x}_{2_{rm}}$, a linear feedback term $\nu_{pd} = [K_1, K_2]e$ with $K_1 \in \mathbb{R}^{n_s \times n_s}$ and $K_2 \in \mathbb{R}^{n_s \times n_s}$, and an adaptive term $\nu_{ad}(z)$. For ν_{ad} to be able to cancel Δ, the following assumption needs to be satisfied:

Assumption B.1 There exists a unique fixed-point solution $\nu_{ad} = \Delta(x, \nu_{ad}), \forall x \in D_x$.

Assumption B.1 implicitly requires the sign of control effectiveness to be known [66]. Sufficient conditions for satisfying this assumption are available in [66].

Using (B.3) the tracking error dynamics can be written as

$$\dot{e} = \dot{x}_{rm} - \begin{bmatrix} x_2 \\ \nu + \Delta \end{bmatrix}. \tag{B.7}$$

Let $A = \begin{bmatrix} 0 & I \\ -K_1 & -K_2 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ I \end{bmatrix}$, where $0 \in \mathbb{R}^{n_s \times n_s}$ and $I \in \mathbb{R}^{n_s \times n_s}$ are the zero and identity matrices, respectively. From (B.6), the tracking error dynamics are then,

$$\dot{e} = Ae + B[\nu_{ad}(z) - \Delta(z)]. \tag{B.8}$$

The baseline full state feedback controller ν_{pd} is chosen to make A Hurwitz. Hence, for any positive definite matrix $Q \in \mathbb{R}^{n \times n}$, a positive definite solution $P \in \mathbb{R}^{n \times n}$ exists for the Lyapunov equation

$$0 = A^T P + PA + Q. \tag{B.9}$$

B.4 Adaptive Control using Gaussian Process Regression

Traditionally in MRAC, it has been assumed that the uncertainty, $\Delta(z)$, is a (smooth) deterministic function. This work offers an alternate view of modeling the uncertainty $\Delta(z)$ in a probabilistic framework using the a Gaussian Process (GP) Bayesian Nonparametric model. A GP is defined as a collection of random variables, any finite subset of which has

a joint Gaussian distribution [102] with mean $m(x)$ and covariance $k(x(t'), x(t))$. For the sake of clarity of exposition, it is assumed that $\Delta(z) \in \mathbb{R}$; the extension to the multidimensional case is straightforward. Note also that $z \in \mathbb{R}^{n+\delta}$ as before. Hence, the uncertainty $\Delta(z)$ here is specified by its mean function $m(z) = \mathbb{E}(\Delta(z))$ and its covariance function $k(z, z') = \mathbb{E}[(\Delta(z) - m(z))(\Delta(z') - m(z'))]$, where z' and z represent different points in the state-input domain. The following notation is used:

$$\Delta(z) \sim \mathcal{GP}(m(z), k(z, z')) \tag{B.10}$$

Ref. [102] provides a complete analysis of the properties of GPs.

B.4.1 GP Regression

See Section 2.1 for equations relating to GP regression.

B.4.2 Online Budgeted Inference

In general, traditional GP inference does not scale well with number of data points because it requires that a kernel center be added for every data point seen [104]. Hence, in order to enable efficient GP inference online on resource constrained platforms, modifications must be made that can ensure online tractability. The matrix inversion required for prediction scales with the number of data points n as $O(n^3)$. Csato [34] provides recursive, rank 1 updates for the weights α and covariance (B.13). However, even with rank-1 updates, prediction computation grows as $O(n)$, where n is the number of kernel points. Instead of creating a kernel at every data point, a restricted set of points referred to as the *active basis vector set* is used. Furthermore, the size of this set is limited by a user selectable "budget". When the number of points stored exceed the budget, new points can only be added by removing an existing point in a way that further enrich the active basis set. In order to determine the novelty of a new point, a linear independence test is performed [34]

as,

$$\gamma_{\tau+1} = k^*_{\tau+1} - k^T_{z_{\tau+1}} \alpha_\tau. \tag{B.11}$$

If $\gamma_{\tau+1}$ exceeds some threshold ϵ_{tol}, then the point is added to the data set. If the budget is exceeded, a basis vector element must be removed prior to adding another element. For this purpose, the KL divergence based scheme presented by Csato [34] is used. This scheme efficiently approximates the KL divergence between the current GP and the $(t+1)$ alternative GPs missing one data point each, then deletes removes the data point with the least score. To compute the updates in an online fashion, define the scalar quantities

$$q^{(\tau+1)} = \frac{y - \alpha^T_\tau k_{x_\tau}}{\omega^2_n + k^T_{x_\tau} C_\tau k_{x_\tau} + k^*_\tau}, \tag{B.12}$$

$$r^{(\tau+1)} = -\frac{1}{\omega^2_n + k^T_{x_\tau} C_\tau k_{x_\tau} + k^*_t}, \tag{B.13}$$

where α_τ, k_{x_τ}, and C_τ are defined in (B.12) and (B.13). Let $e_{\tau+1}$ be the $(\tau+1)$ coordinate vector, and let $T_{\tau+1}(\cdot)$ and $U_{\tau+1}(\cdot)$ denote operators that extend a τ-dimensional vector and matrix to a $(\tau+1)$ vector and $(\tau+1)\times(\tau+1)$ matrix by appending zeros to them. If $\gamma_{\tau+1}$ exceeds ϵ_{tol}, the GP parameters can be solved recursively by using the equations

$$\begin{aligned}
\alpha_{\tau+1} &= T_{\tau+1}(\alpha_\tau) + q^{(\tau+1)} s_{\tau+1}, \\
C_{\tau+1} &= U_{\tau+1}(C_\tau) + r^{(\tau+1)} s_{\tau+1} s^T_{\tau+1}, \\
s_{\tau+1} &= T_{\tau+1}(C_\tau k_{x_{\tau+1}}) + e_{\tau+1}.
\end{aligned} \tag{B.14}$$

If the threshold ϵ_{tol} is not exceeded, the point is not added to the budget, in which case the update equations are given by

$$\begin{aligned}
\alpha_{\tau+1} &= \alpha_\tau + q^{(\tau+1)} s_{\tau+1}, \\
C_{\tau+1} &= C_\tau + r^{(\tau+1)} s_{\tau+1} s^T_{\tau+1}, \\
s_{\tau+1} &= C_\tau k_{x_{\tau+1}} + K(Z,Z)^{-1} k(Z, z_{\tau+1}).
\end{aligned} \tag{B.15}$$

The inverse of the Gram matrix, dentoed by P, needed to solve for $\gamma_{\tau+1}$ is updated online through the equation

$$\begin{aligned} P_{\tau+1} &= U_{\tau+1}(P_\tau) \\ &\quad + \gamma_{\tau+1}^{-1} \left(T_{\tau+1}(\hat{e}_{\tau+1}) - e_{\tau+1}\right) \left(T_{\tau+1}(\hat{e}_{\tau+1}) - e_{\tau+1}\right)^T, \end{aligned} \tag{B.16}$$

where $\hat{e}_{\tau+1} := P_\tau k_{z_{\tau+1}}$. Finally, in order to delete an element, one computes the model parameters with the $(\tau+1)$-th point, and chooses the basis vector with the smallest score measure, given by

$$\epsilon_i = \frac{|\alpha_{\tau+1}(i)|}{P_{\tau+1}(i,i)}. \tag{B.17}$$

Alternatively, other methods for determining which basis vector should be deleted can be used, such as deleting the oldest basis vector [26]. Let ι be the basis vector chosen to be discarded by the score (B.17). Then the deletion equations are given by

$$\begin{aligned} \hat{\alpha} &= \hat{\alpha}^{\neg\iota} - \alpha^* \frac{P^*}{q^*}, \\ \hat{C} &= C^{\neg\iota} + c^* \frac{P^* P^{*T}}{q^{*2}} - \frac{1}{q^*} \left[P^* C^{*T} + C^* P^{*T}\right], \\ \hat{P} &= P^{\neg\iota} - \frac{P^* P^{*T}}{q^*}, \end{aligned} \tag{B.18}$$

where α^* is the ι^{th} component in the vector $\alpha_{\tau+1}$, and $\alpha^{\neg\iota}$ represents the remaining vector. Similarly, $C^{\neg\iota}$ ($P^{\neg\iota}$) represents the $\tau \times \tau$ submatrix in the $(\tau+1) \times (\tau+1)$ matrix $C_{\tau+1}$ ($P_{\tau+1}$) associated to the basis vectors being kept, c^* (q^*) represents the (ι,ι) index into the matrix chosen by the score measure, and C^* (P^*) is the remaining τ-dimensional column vector.

B.4.3 GP nonparametric model based MRAC

This section describes GP-MRAC with online hyperparameter optimization. Let $\hat{(\cdot)}$ denote a kernel with estimated hyperparameters. The adaptive element ν_{ad} is modeled as

$$\nu_{ad}(z) \sim \mathcal{N}(\hat{m}(z), \hat{\Sigma}_Z(z)), \tag{B.19}$$

where $\hat{m}(z)$ is the estimate of the mean of $\Delta(z)$ at the current time instant [20]. In this case, one can choose $\nu_{ad} = \hat{m}(z)$ or draw from the distribution in (B.19). It follows that if one chooses $\nu_{ad} = \hat{m}(z)$, then $\|\nu_{ad} - \Delta\|$ is bounded by the square root of the right hand side of (B.13) within one standard deviation.

B.4.4 Hyperparameter Optimization

This section presents methods for optimizing kernel hyperparameters online. An overview of hyperparameter optimization is presented to make the results intelligible to the reader, but for further details and proofs, the reader is referred to [50]. Regression using GPs requires the use of a kernel function $k(\cdot, \cdot)$, which typically has some hyperparameters that correspond to smoothness properties of the function as well as sensor noise levels. For example, consider one of the most popular choices of kernel functions, the radial basis function (RBF),

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\mu^2})\right) \tag{B.20}$$

In this case, μ is a hyperparameter which controls how correlated points are within the operating domain. A large μ results in smooth function approximation which may lose fine details, whereas a small μ will capture more fine details, but will also be more sensitive to noise. GPs may perform poorly if the design hyperparameters are far from the optimal values, and as such to ensure robust performance, hyperparameters should be optimized online Additionally, if the uncertainty is time varying, the optimal hyperparameters may be time varying, as well.

Traditionally in the GP literature, the hyperparameters are optimized in GPs by maximizing the likelihood, or equivalently, the log likelihood, of the data given the hyperparameters, θ. Taking the derivative with respect to θ yields

$$\frac{\partial}{\partial \theta_j} \log P(y \mid \theta, Z) = \frac{1}{2}\text{tr}\left((\bar{a}\bar{a}^T - \Sigma^{-1})\frac{\partial \Sigma}{\partial \theta_j}\right) \tag{B.21}$$

where $\Sigma_Z = (K(Z, Z) + \omega_n^2 I)$ and $\bar{a} = \Sigma_Z^{-1} y$. Using the closed form of the gradient, one can then choose any gradient based optimization method for maximization of the data

likelihood. Typically, optimization is performed in batch setting. Due to matrix inversion, gradient operations scale poorly with the number of data points, and performing hyperparameter optimization in an online setting using all of the data becomes intractable quickly. Instead, a subset of data is intelligently selected to alleviate computation costs while ensuring convergence to meaningful hyperparameters. Choosing the observations at the basis vectors y_Z as well as using current observations at each time instant i, y_i provides good performance as well as facilitates convergence analysis. Therefore, the hyper parameter update laws are,

$$\theta_j[i+1] = \theta_j[i] + b[i]\frac{\partial}{\partial \theta_j}\log P(y_Z, y_{i+1}|Z, Z_{i+1}, \theta) \tag{B.22}$$

with the following two candidate functions for $b[i]$,

$$b[i+1] = \frac{i}{i+1}b[i] \tag{B.23}$$

$$b[i+1] = (1-\delta)b[i] + \eta \tag{B.24}$$

where b is a dynamic variable which controls the learning rate of θ_j, and $\eta \ll 1$ and $\delta \ll 1$ are parameters chosen by the user to control convergence. (B.23) converges almost surely to a local optima of the modified likelihood function, while (B.24) will not converge, and is better suited to time-varying systems.

The budgeted GP regression algorithm [20, 34] assumed static hyperparameters. As the hyperparameter values are changed by the update equations, the values computed from the linear independence test (B.11) no longer reflect the current model. Therefore, some basis vector points which were previously thought to be roughly independent are now sufficiently correlated in the new model, and should be removed. However, naively recomputing the linear independence test requires $O(|\mathcal{BV}|^3)$ operations, so as an alternative, the algorithm checks to see if the conditional entropy between two points is below a certain threshold which requires $O(|\mathcal{BV}|^2)$ operations. In the specific case of an RBF kernel, this can be reduced to a distance calculation. These point-wise considerations will never remove too

Algorithm 10 Generic Gaussian Process–Model Reference Adaptive Control (GP-MRAC)

while new measurements are available **do**
  Call Algorithm 11
  Given z_{t+1}, compute γ_{t+1} by (B.11)
  Compute $y_{t+1} = \hat{\dot{x}}_{2_{t+1}} - \nu_{t+1}$
  if $\gamma_{t+1} > \epsilon_{tol}$ **then**
    Store $Z(:, t+1) = z(t+1)$
    Store $y_{t+1} = \hat{\dot{x}}_2(t+1) - \nu(t+1)$
    Calculate $K(Z, Z)$ and increment $t+1$
  end if
  if $|\mathcal{BV}| > p_{max}$ **then**
    Delete element in $\mathcal{BV}$ based on methods in Section B.4.2
  end if
  Update α recursively per methods in Section B.4.2
  Calculate $\hat{k}(Z, z(t+1))$
  Calculate $\hat{m}(z(t+1))$ and $\hat{\Sigma}(z_{t+1})$ using (B.12) and (B.13)
  Set $\nu_{ad} = \hat{m}(z(t))$
  Calculate pseudo control ν using (B.6)
  Calculate control input using (B.2)
end while

many points. After the budget is updated, the weights are re-initialized to

$$\alpha = \Sigma_Z^{-1} y \tag{B.25}$$

to account for the new hyperparameter and kernel function values.

Lastly, it is undesirable to recalculate the weights and kernel matrices if the hyperparameters change only slightly. Therefore, the constraint is added in practice that the model hyperparameters are only updated if the change exceeds some designated percentage. For example, in the experiments presented in Section B.6, a threshold of 5% was used. The GP-MRAC algorithm with hyperparameter update is presented in Algorithm 10.

B.5 Control Policy using Bayesian Inference

Typically in AMI-MRAC architectures, the control law $\nu = \nu_{rm} + \nu_{pd} - \nu_{ad}$ is used. At every time step, the algorithm queries the function approximator, such as a NN, and uses the output value as the adaptive element without regard for confidence levels of the

Algorithm 11 Update Hyperparameters

Calculate the derivative of the log likelihood per (B.21)
$\forall j$, Update $\theta_j[i+1]$ and $b[i+1]$ per (B.22)
if $\|\theta_j[i+1] - \theta_j[i]\| / \|\theta_j[i]\|$ **then**
  Update θ_j
  for all $z_i, z_j \in \mathcal{BV}$ **do**
    Calculate $\hat{\Sigma}^2_{i|j}$
    if $\hat{\Sigma}^2_{i|j} < \epsilon_{tol}$ **then**
      Delete z_i,y_i
    end if
  end for
  Recalculate covariance Σ_Z and weights $\alpha = \hat{\Sigma}_Z^{-}1y$
end if

function approximator. This control policy assumes that the ν_{ad} is well known and models $\Delta(z)$ sufficiently well. In fact, the NN does not readily provide any confidence metric on how well it has learned the modeling uncertainty. Hence, while this assumption might be true in the limit as time tends to infinity and sufficient data is presented to the NN through persistent excitation of the system, the assumption is typically not valid when little data has been received or if the data is not very rich. Therefore, in transient periods, it is desirable to take into account the confidence that the algorithm has on what it has learned to decide how much to trust the predictions from the adaptive element. The GP-MRAC framework presents an elegant and efficient way of handling transient learning.

In order to proceed, it is assumed that at each point (x, u), the value of $f(x, u)$ can be obtained through a draw from a normal distribution with an expected value of the approximate model $\hat{f}(x, u)$, and associate a confidence at each point in space $\sigma_f^2(x, u)$.

$$f(x, u) = \mathcal{N}(\hat{f}(x, u), \sigma_f^2(x, u)) \tag{B.26}$$

Thus, for every pair $z = (x, u)$, the expected estimate of $f(x, u)$ is $\hat{f}(x, u)$, but one expects that the true system may deviate from $\hat{f}(x, u)$ with some variance $\sigma_f^2(x, u)$ in practice. Using this formulation allows us to define regions in which one has high certainty or low confidence in the model $\hat{f}(x, u)$, and weigh the adaptive control element appropriately.

Using a prior confidence over the model, one can formulate the maximum a posteriori

(MAP) estimate of the true dynamics $f(x,u)$ using Bayes law. In particular, given a estimate $\mu(x,u)$ of $\dot{x} = f(x,u)$ from some model with a normal distribution and variance $\sigma_n^2(x,u)$, the MAP estimate of $f(x,u)$ is given by

$$f_{MAP}(x,u) = \frac{\sigma_n^2(x,u)}{\sigma_n^2(x,u)+\sigma_f^2(x,u)}\hat{f}(x,u) + \frac{\sigma_f^2(x,u)}{\sigma_n^2(x,u)+\sigma_f^2(x,u)}\mu(x,u) \tag{B.27}$$

which leads to the MAP estimate of the modeling error,

$$\Delta_{MAP}(x,u) = \frac{\sigma_f^2(x,u)}{\sigma_n^2(x,u)+\sigma_f^2(x,u)}(\mu(x,u) - \hat{f}(x,u)) \tag{B.28}$$

The Bayesian scaling factor is defined as, $\rho_{MAP} = \frac{\sigma_f^2(x,u)}{\sigma_n^2(x,u)+\sigma_f^2(x,u)}$. Plugging in, the control law becomes

$$\nu = \nu_{rm} + \nu_{pd} - \rho_{MAP}\nu_{ad} \tag{B.29}$$

In practice, ρ_{MAP} may jump around as the estimates of x are noisy. For this reason, instead of using ρ_{MAP} directly, a low pass filtered version is used

$$\rho_{MAP}[t+1] = \rho_{MAP}[t] + \epsilon\left(\frac{\sigma_f^2(x,u)}{\sigma_n^2(x,u)+\sigma_f^2(x,u)} - \rho_{MAP}[t]\right) \tag{B.30}$$

where $\epsilon \in [0,1)$ is some constant which determines filtering properties. If the confidence in the initial model is high or if the confidence in the measurement $\mu(x,u)$ is low, $\rho_{MAP}(x,u)$ will be small, so the adaptive element will be small, and the algorithm will rely on the prior model more. However, as the confidence in $\mu(x,u)$ increases, $\rho_{MAP}(x,u) \approx 1$, which results in the original MRAC law. Conveniently, GPs provide an estimate of predictive variance as well as a mean value, making them a suitable candidate for such posterior inference. As the GP model receives more measurements in a certain region, the variance $\sigma_n^2(x,u)$ decays and the algorithm rely on the adaptive element more than at the beginning. If the system starts to venture into new regions of the operating domain, or if the hyperparameters change, the variance of $\sigma_n^2(x,u)$ becomes large again, and the control policy falls back to the prior model. It is worth noting that since $\rho_{MAP} \in [0,1]$, all proofs which establish boundedness when using ν_{ad} also hold for the case when using $\rho_{MAP}\nu_{ad}$. Formally,

Theorem B.1 Consider the system in (B.1), the reference model in (B.5), the control law of (B.6) and (B.2). Let the uncertainty $\Delta(z)$ be represented by a Gaussian process as in (B.10), then GP-MRAC (Algorithm 1) and the adaptive signal $\nu_{ad}(z) = \rho_{MAP}\hat{m}(z)$ guarantees that the system is mean square ultimately bounded a.s. inside a compact set.

Proof Define $\hat{\nu}_{ad} = \rho_{MAP}\hat{m}(z)$. From Lemma 3 in Ref. [50], $\|\nu_{ad}\|$ is bounded for all time. Therefore, $\hat{\nu}_{ad}$ is bounded for all time. From Theorem 2 in [27, 28], if $\|\Delta(z) - \nu_{ad}\|$ is bounded, then the tracking error is mean square uniformly ultimately bounded inside of a compact set almost surely. If $\|\Delta(z) - \hat{m}(z)\|$ is bounded for all time, it follows that $\|\Delta(z) - \rho_{MAP}\hat{m}(z)\|$ is bounded. Therefore, since $\|\Delta(z) - \rho_{MAP}\hat{m}(z)\|$ is bounded for all time, the tracking error is mean square uniformly ultimately bounded inside of a compact set almost surely.

Theorem 1 states that including a Bayesian scaling factor in GP-MRAC will not destabilize the closed loop system. This is significant, because it allows for a natural and elegant method in which the predictive confidence of the adaptive element can be used to scale the adaptive input while maintaining boundedness.

B.6 Experimental Results

This section validates the GP-MRAC architecture through an extensive set of flight results. The results are evaluated through three separate experiments with multiple trials to ensure statistically significant finding. The tracking performance of GP-MRAC is compared with two variants of RBFN-MRAC as well as GP-MRAC with hyperparameter optimization. In all cases, GP-MRAC shows a significant performance advantage over traditional fixed parameter RBFN-MRAC and Concurrent-Learning based MRAC. The experiments are performed in the Aerospace Control Laboratory's RAVEN test environment at MIT [56]. The RAVEN test environment uses a motion capture system to obtain accurate estimates of the position and attitude of autonomous vehicles . Accelerations and angular rates were calculated using a fixed interval smoother. Position measurements are differentiated twice to

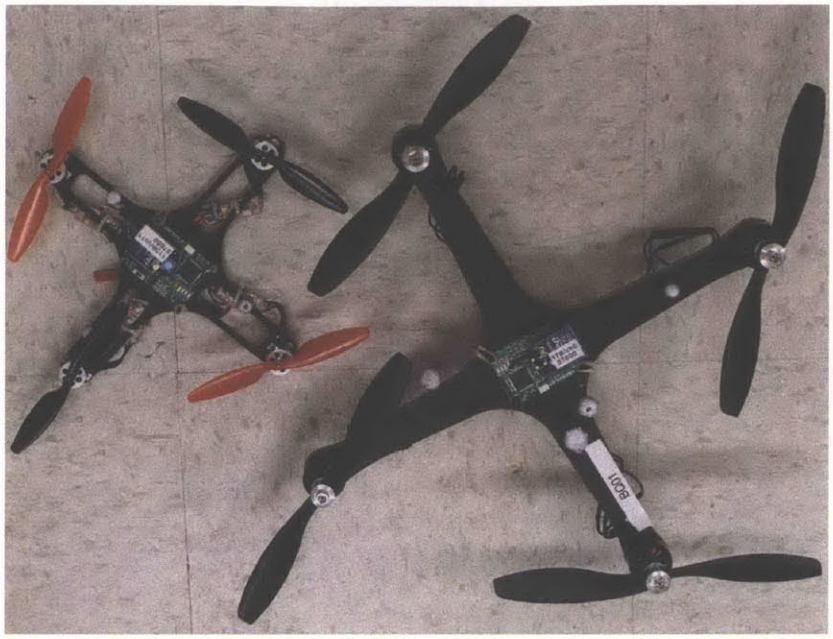


Figure B-1: Two MIT quadrotors equipped to fly in the ACL Real Time Indoor Autonomous Vehicle Test Environment (RAVEN) [54]. The baseline controller on both quadrotors is PID. The small quadrotor uses gains and thrust mappings from the bigger one, resulting in relatively poor trajectory tracking performance.

obtain accelerations, which leads to increased noise. The motion capture environment does provide high fidelity position measurements, which balances the noise introduced through numeric differentiation. However, it was observed that the GP was fairly robust to any noise introduced through numeric differentiation. In environments in which sensor noise levels are larger, the noise term ω_n^2 can be increased accordingly. This will result in the GP learning the model error slower and a Bayesian mixing factor which increases slower.

B.6.1 Hardware Details

The flight experiments in this work were performed on the smaller of the two quadrotors shown in Figure B-1. This vehicle weighs 96 grams without the battery and measures 18.8 cm from motor to motor. Both quadrotors utilize standard hobby brushless motors, speed controllers, and fixed-pitch propellers, all mounted to custom-milled carbon fiber frames. On-board attitude control is performed on a custom autopilot, with attitude com-

mands being calculated at 1 kHz. Due to limitations of the speed controllers, the smaller quadrotor motors only accept motor updates at 490 Hz. More details on the autopilot and attitude control are in [35, 36].

B.6.2 Augmentation of baseline linear control with adaptation

The outer loop (position and velocity) of the baseline proportional-integral-derivative controller (PID) on the quadrotor was augmented with AMI-MRAC adaptive control. The outer loop of the baseline controller generates desired acceleration to track a spline-based trajectory, these desired accelerations are then mapped to thrust and desired attitude (quaternion) commands [36]. The inner loop attitude controller is implemented onboard the vehicle and was left unaltered. The outerloop runs at 100 Hz on an off-board computer, and the inner loop attitude control runs at 500 Hz. It is assumed that the attitude control is sufficiently fast compared to the outer loop controller.

The experiments are performed in a rapid control transfer setting. In particular, the well tuned PID inner-loop and outer-loop controller from the larger quadrotor are implemented directly on the smaller quadrotor. The inner loop controller is not altered, while the outer loop controller is augmented in the framework of AMI-MRAC using GP-MRAC and other methods. The large differences in size and mass between the two quadrotors results in relatively poor position and velocity tracking with the small quadrotor when it uses the gains from the bigger quadrotor. In addition, the lack of knowledge of the mapping from unit-less speed controller commands to propeller thrust for the small quadrotor requires significant input from the altitude integral PID term and leads to poor vertical and horizontal position tracking. In the AMI-MRAC implementation, the inversion law is very approximate, with modeling errors on the order of $2-6\frac{m}{s^2}$ during experiments. Additionally, the PID are tuned only to achieve stability and moderate performance. Hence, any gains in performance over the baseline controller are due to the adaptive law.

In these experiments, a baseline model and controller is used which is qualitatively similar to the true model, albeit with scaling differences. It should be noted that since a GP is a nonparametric function approximator, the form of the model error does not matter. A

GP will learn the model error equally well regardless if the baseline model is qualitatively similar. This is different from parametric approaches to adaptive control, in which if the parameterization of the model error is incorrect, the learning capabilities of the adaptive element itself will be hindered. A similar baseline controller and PID law will, however, improve the initial tracking capabilities in either case.

In the following discussion, baseline PID refers to the outer-loop PID controller on the smaller quadrotors with gains directly transferred from the larger (well-tuned) quadrotor. MRAC refers to augmentation of the baseline law with a RBFN adaptive element and CL-MRAC refers to Concurrent Learning - MRAC, a learning-focused version of MRAC with RBFNs and gradient based updates which uses recorded and current data concurrently for adaptation and learning [21, 23, 30]. For details on how the adaptive element ν_{ad} is calculated using MRAC and CL-MRAC, the reader is referred to [30]. GP-MRAC will refer to a controller of the form (B.6) with $\nu_{ad} = \rho_{MAP}\hat{m}(z)$, in which hyperparameters are static, and GP-MRAC-HP will refer to a controller of the same form in which hyperparameters are updated online. In MRAC and CL-MRAC, a projection operator was used to ensure robustness, although the limit of the projection operation was not reached in experiments. The projection operator is not required for GP-MRAC and GP-MRAC-HP [20].

The controllers were separated into three different loops corresponding to x, y, z positions. The input to the RBF for MRAC was given by $z_t = [\dot{x}, \dot{y}, \dot{z}, \vec{q}]$. The input to the GP was $z_t = [\dot{x}, \dot{y}, \dot{z}, \vec{q}, \nu]$. In [19], the input to the GP was given by $z_t = [\dot{x}, \dot{y}, \dot{z}, \vec{q}]$. In this work, the pseudo-input ν is added to the GP input, which in practice reduced the tracking error significantly. It was attempted to add ν to the RBFN-MRAC, however, it was found that performance was degraded. One possible reason for this is the fact that it is difficult to preallocate RBF centers over the commanded signals ν, since it is not entirely known how ν will behave. Additionally, the number of RBFs needed to uniformly cover the input domain increases exponentially with the number of dimensions of the input. While GP-MRAC is able to dynamically allocate centers over only active regions of the input domain, MRAC with fixed centers cannot adequately cover the domain without an exponential number of centers.

Several different values of learning rates for RBFN-MRAC, and CL-MRAC were an-

alyzed through simulation and preliminary flight testing for MRAC. Note that GP-MRAC and GP-MRAC-HP do not use a learning rate. The results presented here correspond to values that resulted in good performance without over-learning. The best initial MRAC learning was found to be $\Gamma_W = 2$, and the learning rate of the adaptive law that trains on recorded data was found to be $\Gamma_{W_b} = 0.5$. Theoretically, MRAC and CL-MRAC learning rates can be kept constant in adaptive control, however stochastic stability results [79]) indicate that driving the learning rates to zero is required for guaranteeing convergence in presence of noise. The problem with this approach is that the effect of adaptation would be eventually removed. Therefore, in practice, the learning rate is set to decay to a small constant to avoid unstable or oscillatory behaviors. Learning rates for MRAC and CL-MRAC were decayed by dividing it by 1.5 for Γ_W and 2 for Γ_{W_b} each 20s. The decay limit of these learning rates are $\Gamma_W = 0.5$ and $\Gamma_{W_b} = 0.001$.

For MRAC and CL-MRAC 100 RBF centers were generated using a uniform random distribution over a domain where the states were expected to evolve. The centers for the position and velocity for the x,y axes were spread across $[-2, 2]$. For the vertical z axis, the position and velocity were spread across $[-0.5, 0.5]$ and $[-0.6, 0.6]$ respectively. The centers for quaternions were placed within $[-1, 1]$. The bandwidth for each radial basis function is set to $\mu_i = 1$. For GP-MRAC, centers are assigned dynamically by the algorithm itself (see Section B.4.2), and it not required to assign them a priori. The budget for the online dictionary of active bases was set to 100 and $\epsilon_{tol} = 0.0001$. An RBF kernel was used with initial bandwidth $\mu = 2.2$ and noise $\omega_n = 2$. These values were obtained by running GP-MRAC-HP for one trial and taking the final estimated optimal hyperparameters. For GP-MRAC-HP, the adaptation parameter $b[0] = 0.02$ was used. In each of the experiments, the initial bandwidth μ was varied in order to show that the performance of GP-MRAC-HP was not greatly affected by prior choice of μ. For the Bayesian mixing parameter, $\epsilon = 0.002$ was used, corresponding to a rise time of 5s. As the number of data points in a region increases, the variance of the GP decreases and the GP changes less with successive points in those regions. As opposed to the RBFN based approaches, one does not need to specify learning rates manually with GP-MRAC in order to avoid over fitting or oscillations.

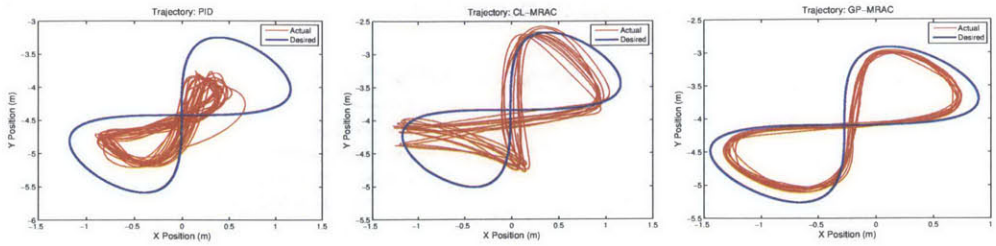


Figure B-2: Sample trajectories of the quadrotor following a figure eight pattern. The blue indicates the commanded path and the red indicates the actual path flown by the quadrotor. On the left, baseline PID is shown, in the middle, CL-MRAC, on the right, GP-MRAC. GP-MRAC follows the trajectory best in terms of both tracking error and qualitatively matching the shape of the figure 8 trajectory.

B.6.3 Flight-Test results

Figure 8 maneuvers

The quadrotor performed 5 trials consisting of 20 "figure 8" maneuvers with a period of 6.28s, taking approximately 125s per trial. In the case of GP-MRAC with hyperparameter estimation, the initial bandwidth was set to $\mu = 0.55, 1.1, 2.2, 4.4, 8.8$ for trials $1, 2, 3, 4, 5$, respectively. In figure B-2 a plot of the trajectory in space is presented. Using the initial PID controller, the quadrotor is unable to track the figure 8. CL-MRAC performs better, although it overshoots at points, resulting in sharp corners. Lastly, GP-MRAC performs the best in terms of tracking error as well as qualitatively matching the figure eight shape. GP-MRAC is also the most consistent of the controllers, producing little variation between iterations of the figure 8 shape.

Figure B-3 shows the windowed tracking error of each algorithm as a function of time. The baseline PID controller does not perform well in terms of tracking error. Due to a poor approximate model, the feed forward component of the control signal does very little in ensuring good tracking, and the majority of trajectory tracking performance is due to the feedback, leading to high error. RBFN-MRAC performs better over time, but the convergence rate is quite slow due to a lack of persistency of excitation (PE). CL-MRAC does not require PE and converges relatively quickly with less tracking error than traditional MRAC. However, GP-MRAC outperforms all three previously mentioned controllers substantially in terms of tracking error. GP-MRAC reduces the steady state error from the PID controller

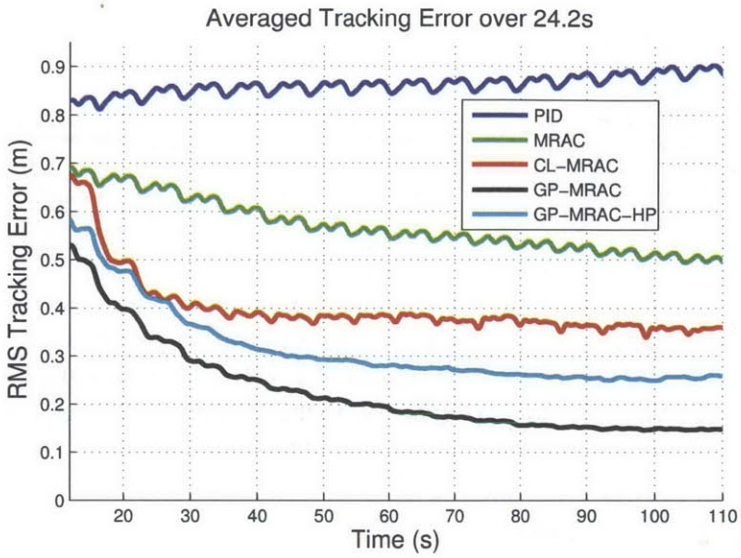


Figure B-3: GP-MRAC outperforms traditional MRAC in terms of RMSE tracking error for a figure 8 trajectory.

by approximately 85%, outperforms RBFN-MRAC by over a factor of three, and outperforms CL-MRAC by over a factor of two. GP-MRAC-HP with initial suboptimal bandwidth selection still performs well, although not as well as GP-MRAC. As GP-MRAC-HP adapts the hyperparameters, the certainty in the model decreases, leading to a lower ρ_{MAP} and a higher tracking error. As time tends to infinity, GP-MRAC-HP will converge to the optimal hyperparameters, leading to the same performances as GP-MRAC with initial optimal hyperparameters.

In figure B-4, the windowed RMSE of the difference $\|\Delta(z) - \nu_{ad}\|$ is plotted. GP-MRAC models the error $\|\Delta(z) - \nu_{ad}\|$ substantially better than RBFN-MRAC and CL-MRAC. GP-MRAC outperforms RBFN-MRAC by over a factor of three. This improvement in characterizing the uncertainty $\|\Delta(z) - \nu_{ad}\|$ is what fundamentally leads to the lower tracking error of GP-MRAC.

Pseudo Random Sum of Sines

In this experiment, the quadrotor tracked a pseudo random sum of sines signal. The frequencies used were aperiodic so that the resulting signal never repeats itself. The desired

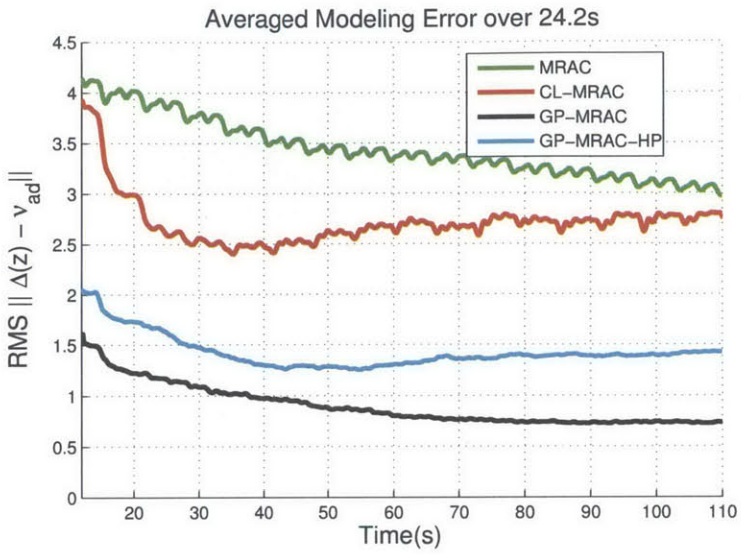


Figure B-4: GP-MRAC outperforms traditional MRAC in terms of RMSE modeling error for a figure 8 trajectory.

trajectories were given by $x_{rm}(t) = 0.7\sin(t) + 0.35\sin(0.33t)$, $y_{rm}(t) = 0.7\sin(1.1t + \pi/2) + 0.35\sin(2.11t)$, and $z_{rm}(t) = 0.1725\sin(1.33t) + 0.0875\sin(1.76t)$. A total of three trials were performed for each algorithm. GP-MRAC-HP was initialized using the bandwidths $\mu = 1.1, 2.2, 4.4$. Even when the signal is non repetitive, GP-MRAC can still learn the uncertainty better than MRAC and reduce tracking error reliably. Figure B-5 shows a plot of the trajectory in space. Figure B-6 shows the tracking performance of GP-MRAC and CL-MRAC in each dimension for a single trial.

Figure B-7 shows the windowed tracking error of each algorithm as a function of time. In figure B-8, the windowed RMSE of the difference $\|\Delta(z) - \nu_{ad}\|$ is plotted. Similar to the figure eight trajectory, MRAC and CL-MRAC improve the performance of the baseline controller, however both variants of GP-MRAC perform the best. In this case, the tracking performance of GP-MRAC and GP-MRAC-HP are nearly identical. GP-MRAC again outperforms MRAC by over a factor of 3.5 and outperforms CL-MRAC by about a factor of 2.5. In terms of model learning, GP-MRAC reduces the modeling error, $\|\Delta(z) - \nu_{ad}\|$, by a factor of 3-5 times over the RBFN adaptive element. Figure B-10 shows the GP prediction of the model uncertainty and the resulting control signal. These experiments show the superior capabilities of GPs to capture very fine variations in the model error while

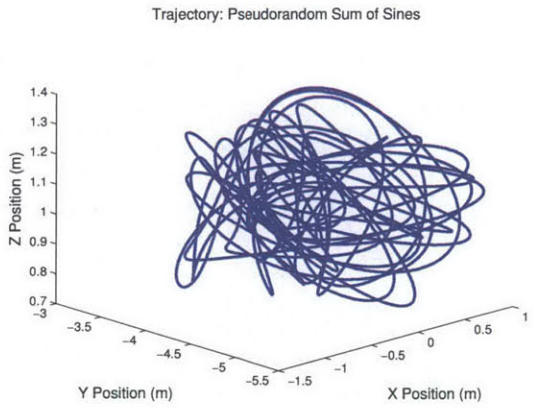


Figure B-5: Commanded pseudo-random sum of sines trajectory

simultaneously rejecting sensor noise.

Operation Outside Expected Operating Domain

In many cases, it is difficult to a priori designate the operating domain over which a system will operate. If the system leaves the domain covered by the adaptive element, than the system will not track the trajectory well and can cause instability. Even if a-priori bounds are known on the maximum flight envelope as estimated by experts, it is still possible that due to noise, external disturbances, or PID overshoot that the system will exit the covered domain. This experimental scenario demonstrates how overshooting can lead to instability in the case of fixed center RBFN-MRAC, and show that the ability of GP-MRAC to dynamically allocate centers and to use a Bayesian mixing factor ensure that the adaptive element is completely trusted only in high confidence regions of the state space, which ensures stability and good performance. In order to demonstrate this idea, a fictitious "training and testing" experiment is created. In this experiment, the adaptive element is trained on a pseudo-random sum of sines for 60 s, followed by a 10 s rest, followed by a test of 60 s of flying in a circle. During the testing period, the adaptive element is continuously learning.

Fig B-11 shows the velocity plot of CL-MRAC tracking the circle. In theory, the commanded trajectory is within the operating domain of MRAC and CL-MRAC, so the system should remain stable. In practice, on this experiment, fixed-center CL-MRAC consistently entered into a cycle of unstable oscillations leading to a crash. As seen in figure B-11,

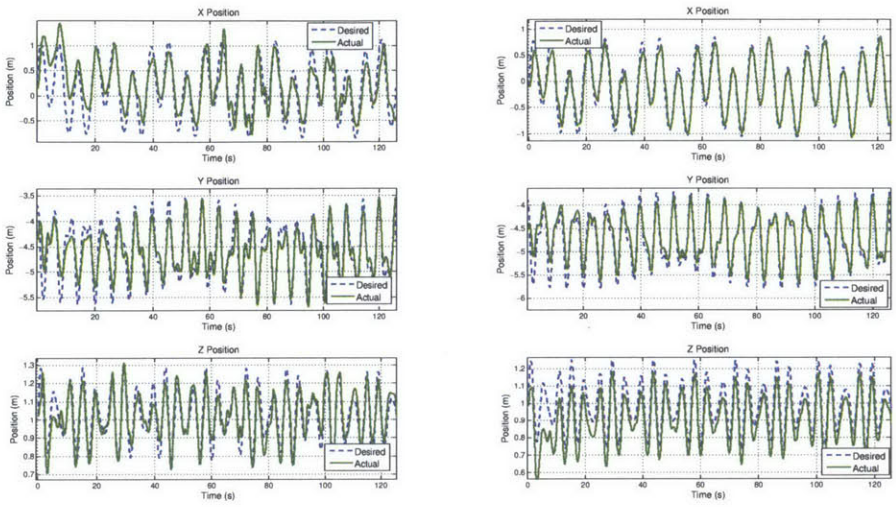


Figure B-6: Tracking performance in each dimension using CL-MRAC on the left and GP-MRAC on the right. While CL-MRAC improves performance through model learning, the improved model learning of GP-MRAC leads to even better tracking performance and very small tracking error over time.

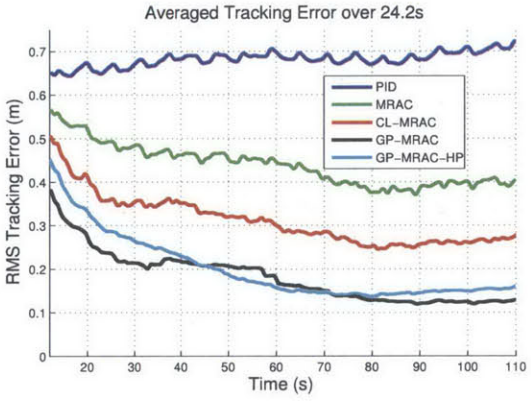


Figure B-7: GP-MRAC outperforms traditional MRAC in terms of RMSE tracking error for a random trajectory.

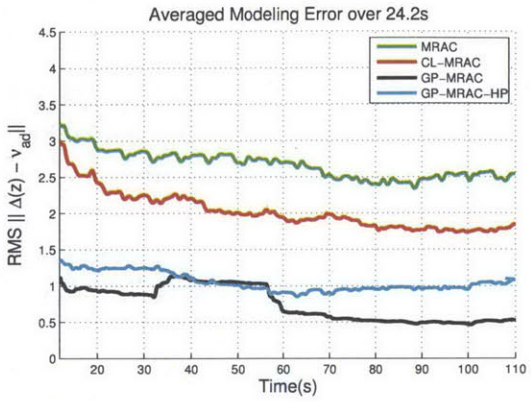


Figure B-8: GP-MRAC outperforms traditional MRAC in terms of RMSE modeling error for a random trajectory.

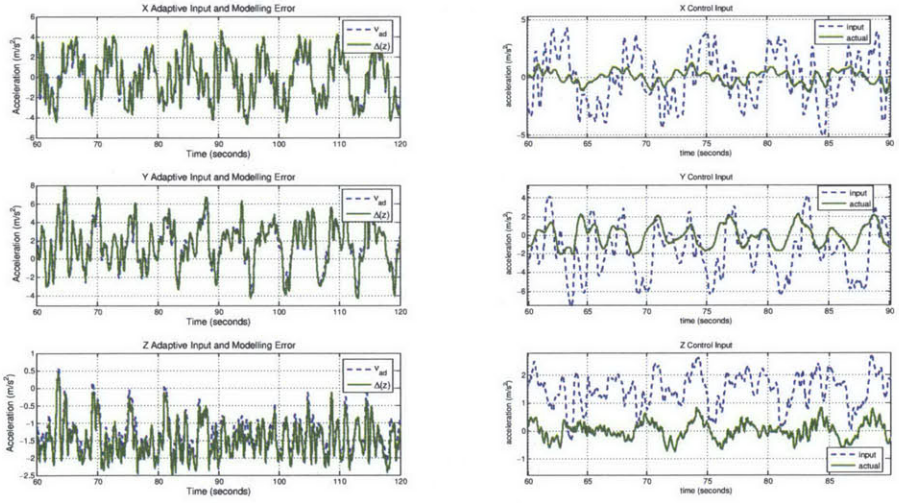


Figure B-9: GP-MRAC is able to reject sensor noise and produce a smooth control signal.

Figure B-10: GP-MRAC is able to characterize very fine variations in the model error which leads to superior tracking performance over standard RBFN-MRAC. However, GP-MRAC is still able to reject sensor noise and produce a smooth control signal.

fixed-center CL-MRAC begins to lag the trajectory in the y direction and lead in the x direction. Eventually the PID correction terms lead to overshooting at $t = 125$s which moves the system outside of the RBFN coverage in the z direction. This leads to oscillations resulting in the quadrotor crashing at $t = 126$s. The quadrotor crashes into a net which is why the velocity in the x, y directions are zero but the z velocity is non-zero immediately after the crash. Note that the instability is due to the inability of CL-MRAC to dynamically allocate centers, and not the parameter update law of CL-MRAC. A dynamic center allocation version of CL-MRAC, known as BKR-CL-MRAC[68] may perform better in this scenario. However, it was not tested since previous results [30] show that CL-MRAC outperforms BKR-CL-MRAC in terms of tracking performance when operating over the expected operating domain.

On the contrary, GP-MRAC is able to accommodate the overshooting by leveraging a combination of dynamic center allocation as well as the Bayesian mixing factor. Figure B-12 shows an example velocity plot from GP-MRAC. GP-MRAC overshoots at times $t = 75$ as well as $t = 85$ in the y direction as well as at $t = 71$ in the z direction. After the system overshoots, GP-MRAC is able to place new centers at the new locations. Additionally, since GP-MRAC begins operating in a new part of the input domain, the associated covariance of the adaptive element in that part of the operating region is large. This results in a lower ρ_{MAP} and conservative control which relies on the baseline PID controller more. After a few cycles, the system settles down and reaches steady state around $t = 91$s.

B.7 Conclusion

Gaussian Process Model Reference Adaptive Control (GP-MRAC) architecture builds upon the idea of using Bayesian nonparametric probabilistic framework, such as GPs, as an alternative to deterministic adaptive elements to enable adaptation with little or no prior domain knowledge [20]. GPs allow for dynamic allocation of kernel centers as well as automatic tuning of the hyperparameters. This work introduced the notion of a Bayesian mixing factor which scales the adaptive input by weighing the relative certainty in the prior model as well as the model of the adaptive element in a Bayesian manner. The main contribu-

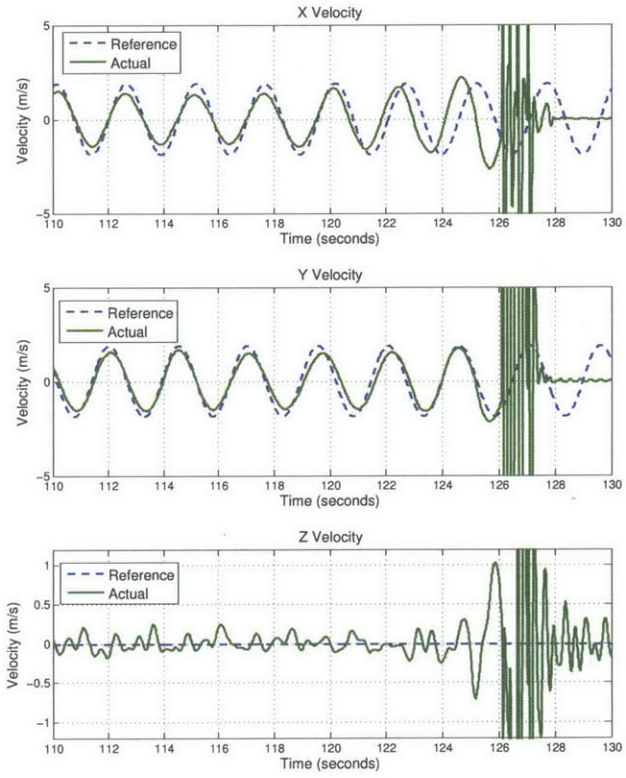


Figure B-11: CL-MRAC with fixed RBFN bases initially overshoots the trajectory at $t = 125$s. After overshooting, the system leaves the coverage of the RBF centers leading to unstable oscillations.

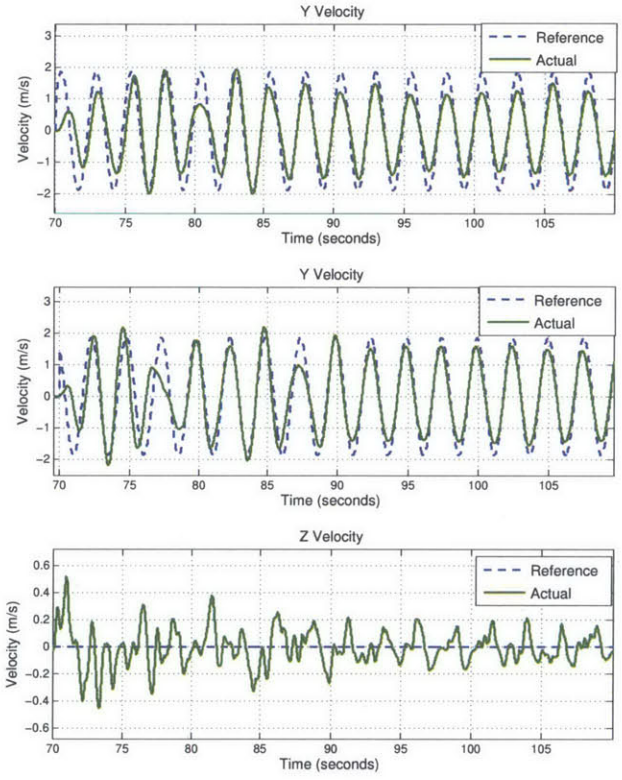


Figure B-12: GP-MRAC overshoots the trajectory at several points in the beginning, but is able to reallocate centers to maintain domain coverage. Additionally, as new centers are created, the model covariance increases, leading to a smaller ρ_{MAP} which results in a more conservative controller.

tion of this work was the first extensive flight results comparing the tracking performance of GP-MRAC to traditional neural network MRAC as well as neural network MRAC using concurrent learning (CL-MRAC). This work presented the first experimental results of online hyperparameter optimization in a closed loop control setting. Results show that GP-MRAC outperforms traditional MRAC by approximately a factor of three, and CL-MRAC by a factor of two, in terms of tracking error as well as characterization of the model error in this domain. Lastly, it was demonstrated that the ability of GP-MRAC to dynamically allocate RBF centers allowed it to maintain domain coverage, thus avoiding crashing.

Appendix C

Sample Efficient Reinforcement Learning with Gaussian Processes

This appendix presents sample complexity results for Gaussian Processes in RL. This work presents the first provably efficient model-free RL algorithm for continuous spaces, DGPQ. This work was initially published in [52] and represents work that I have done during my Masters which did not fit in thematically with the main text of the thesis.

C.1 Abstract

This work derives sample complexity results for using Gaussian Processes (GPs) in both model-based and model-free reinforcement learning (RL). It is shown that GPs are KWIK learnable, proving for the first time that a model-based RL approach using GPs, GP-Rmax, is sample efficient (PAC-MDP). However, it is then shown that previous approaches to model-free RL using GPs take an exponential number of steps to find an optimal policy, and are therefore not sample efficient. The third and main contribution is the introduction of a model-free RL algorithm using GPs, DGPQ, which is sample efficient and, in contrast to model-based algorithms, capable of acting in real time, as demonstrated on a five-dimensional aircraft simulator.

C.2 Introduction

In Reinforcement Learning (RL) [118], several new algorithms for efficient exploration in continuous state spaces have been proposed, including GP-Rmax [61] and C-PACE [95]. In particular, C-PACE was shown to be PAC-MDP, an important class of RL algorithms that obtain an optimal policy in a polynomial number of exploration steps. However, these approaches require a costly fixed-point computation on *each* experience, making them ill-suited for real-time control of physical systems, such as aircraft. This work presents a series of sample complexity (PAC-MDP) results for algorithms that use Gaussian Processes (GPs) [103] in RL, culminating with the introduction of a PAC-MDP model-free algorithm which does not require this fixed-point computation and is better suited for real-time learning and control.

First, using the KWIK learning framework [77], this work provides the first-ever sample complexity analysis of GP learning under the conditions necessary for RL. The result of this analysis actually proves that the previously described model-based GP-Rmax is indeed PAC-MDP. However, it still cannot be used in real time, as mentioned above. The second contribution is in model-free RL, in which a GP is used to directly model the Q-function. It is shown that existing model-free algorithms [32, 43] that use a single GP may require an exponential (in the discount factor) number of samples to reach a near-optimal policy.

The third, and primary, contribution of this work is the introduction of the first model-free continuous state space PAC-MDP algorithm using GPs: Delayed-GPQ (DGPQ). DGPQ represents the current value function as a GP, and updates a separately stored value function only when sufficient outlier data has been detected. This operation "overwrites" a portion of the stored value function and resets the GP confidence bounds, avoiding the slowed convergence rate of the naïve model-free approach.

The underlying analogy is that while GP-Rmax and C-PACE are generalizations of model-based Rmax [11], DGPQ is a generalization of model-free Delayed Q-learning (DQL) [114, 115] to general continuous spaces. That is, while early PAC-MDP RL algorithms were model-based and ran a planner after each experience [77], DQL is a PAC-MDP algorithm that performs *at most* a single Bellman backup on each step. In DGPQ, the

delayed updates of DQL are adapted to continuous state spaces using a GP.

It is proven that DGPQ is PAC-MDP and shown that using a sparse online GP implementation [33] DGPQ can perform in real-time. The empirical results, including those on an F-16 simulator, show DGPQ is both sample efficient and orders of magnitude faster in per-sample computation than other PAC-MDP continuous-state learners.

C.3 Background

Background material on Reinforcement Learning (RL) and Gaussian Processes (GPs) are now presented.

C.3.1 Reinforcement Learning

The RL environment is modeled as a Markov Decision Process [99] $M = \langle S, A, R, T, \gamma \rangle$ with a potentially infinite set of states S, finite actions A, and $0 \leq \gamma < 1$. Each step elicits a reward $R(s,a) \mapsto [0, R_{\max}]$ and a stochastic transition to state $s' \sim T(s,a)$. Every MDP has an optimal value function $Q^*(s,a) = R(s,a) + \gamma \int_{s'} T(s,a,s') V^*(s')$ where $V^*(s) = \max_a Q^*(s,a)$ and corresponding optimal policy $\pi^* : S \mapsto A$. Note the bounded reward function means $V^* \in [0, V_{\max}]$.

In RL, an agent is given S,A, and γ and then acts in M with the goal of enacting π^*. For value-based methods (as opposed to policy search [63]), there are roughly two classes of RL algorithms: model-based and model-free. Model-based algorithms, such as KWIK-Rmax [77], build models of T and R and then use a planner to find Q^*. Many model-based approaches have *sample efficiency* guarantees, that is bounds on the amount of *exploration* they perform. This work uses the PAC-MDP definition of sample complexity [114]. The definition uses definitions of the covering number of a set (adapted from [95]).

Definition C.1 The **Covering Number** $\mathcal{N}_U(r)$ of a compact domain $U \subset \mathbb{R}^n$ is the cardinality of the minimal set $C = \{c_i, \ldots, c_{N_c}\}$ s.t. $\forall x \in U$, $\exists c_j \in C$ s.t. $d(x, c_j) \leq r$, where $d(\cdot, \cdot)$ is some distance metric.

Definition C.2 Algorithm $\mathcal{A}$ (non-stationary policy $\mathcal{A}_t$) with accuracy parameters ϵ and δ in an MDP of size $\mathcal{N}_{SA}(r)$ is said to be Probably Approximately Correct for MDPs (PAC-MDP) if, with probability $1-\delta$, it takes no more than a polynomial (in $\langle \mathcal{N}_{SA}(r), \frac{1}{1-\gamma}, \frac{1}{\epsilon}, \frac{1}{\delta} \rangle$) number of steps where $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$.

By contrast, *model-free* methods such as Q-Learning [128] build Q^* directly from experience without explicitly representing T and R. Generally model-based methods are more sample efficient but require more computation time for the planner. Model-free methods are generally computationally light and can be applied without a planner, but need (sometimes exponentially) more samples, and are usually not PAC-MDP. There are also methods that are not easily classified in these categories, such as C-PACE [95], which does not explicitly model T and R but performs a fixed-point operation for planning.

C.3.2 Gaussian Processes

This work uses GPs as function approximators both in model-based RL (where GPs represent T and R) and model-free RL (where GPs represent Q^*), while showing how to maintain PAC-MDP sample efficiency in both cases. For information on GPs, the reader is directed to Section 2.1.

C.3.3 Related Work

GPs have been used for both model-free and model-based RL. In model-free RL, GP-Sarsa [43] has been used to model a value function and extended to off-policy learning [29], and for (heuristically) better exploration in iGP-Sarsa [32]. However, it is proven in Section C.5.1 that these algorithms may require an exponential (in $\frac{1}{1-\gamma}$) number of samples to reach optimal behavior since they only use a single GP to model the value function.

In model-based RL, the PILCO algorithm trained GPs to represent T, and then derived policies using policy search [39]. However, PILCO does not include a provably efficient (PAC-MDP) exploration strategy. GP-Rmax [61] does include an exploration strategy, specifically replacing areas of low confidence in the (T and R) GPs with high valued states, but no theoretical results are given in that paper. In Section C.4, it is shown that GP-Rmax

actually is PAC-MDP, but the algorithm's planning phase (comparable to the C-PACE planning in the experiments) after each update makes it computationally infeasible for real-time control.

REKWIRE [76] uses KWIK linear regression for a model-free PAC-MDP algorithm. However, REKWIRE needs H separate approximators for finite horizon H, and assumes Q can be modeled as a linear function, in contrast to the general continuous functions considered in this work. The C-PACE algorithm [95] has already been shown to be PAC-MDP in the continuous setting, though it does not use a GP representation. C-PACE stores data points that do not have close-enough neighbors to be considered "known". When it adds a new data point, the Q-values of each point are calculated by a value-iteration like operation. The computation for this operation grows with the number of datapoints and so (as shown in the experiments) the algorithm may not be able to act in real time.

C.4 GPs for Model-Based RL

In model-based RL (MBRL), models of T and R are created from data and a planner derives π^*. Here the KWIK-Rmax MBRL architecture, which is PAC-MDP, is reviewed. It is then shown that GPs are a KWIK-learnable representation of T and R, thereby proving that GP-Rmax [61] is PAC-MDP given an exact planner.

C.4.1 KWIK Learning and Exploration

The "Knows What it Knows" (KWIK) framework [77] is a supervised learning framework for measuring the number of times a learner will admit uncertainty. A KWIK learning agent is given a hypothesis class $H : X \mapsto Y$ for inputs X and outputs Y and parameters ϵ and δ. With probability $1 - \delta$ over a run, when given an adversarially chosen input x_t, the agent must, either (1) predict $\hat{y}_t$ if $||y_t - \hat{y}_t|| \leq \epsilon$ where y_t is the true expected output ($E[h^*(x)] = y_t$) or (2) admit "I don't know" (denoted $\perp$). A representation is *KWIK learnable* if an agent can KWIK learn any $h^* \in H$ with only a polynomial (in $\langle |H|, \frac{1}{\epsilon}, \frac{1}{\delta} \rangle$) number of $\perp$ predictions. In RL, the KWIK-Rmax algorithm [77], uses KWIK learners to model T and R and replaces the value of any $Q^*(s,a)$ where $T(s,a)$ or $R(s,a)$ is $\perp$ with

a value of $V_{\max} = \frac{R_{\max}}{1-\gamma}$. If T and R are KWIK-learnable, then the resulting KWIK-Rmax RL algorithm will be PAC-MDP under Definition C.2. It is now shown that a GP is KWIK learnable.

C.4.2 KWIK Learning a GP

First, a metric for determining if a GP's mean prediction at input x is ϵ-accurate with high probability is required. Lemma 3.1 gives such a sufficient condition to ensure that, with high probability, the mean of the GP is within ϵ_1 of $E[h^*(x)]$. A KWIK agent can now be constructed that predicts $\perp$ if the variance is greater than σ^2_{tol} and otherwise predicts $\hat{y}_t = \mu(x)$, the mean prediction of the GP. Theorem C.1 bounds the number of $\perp$ predictions by such an agent, and when $\delta_1 = \frac{\delta}{\mathcal{N}_U(r(\frac{1}{2}\sigma^2_{tol}))}$, establishes the KWIK learnability of GPs. For ease of exposition, the definition of the equivalent distance is used from Section 2.2.

Theorem C.1 Consider a GP model trained over a compact domain $U \subset \mathbb{R}^n$ with covering number $\mathcal{N}_U(r(\frac{1}{2}\sigma^2_{tol}))$. Let the observations $\vec{y} = [y_1, \ldots, y_n]$ be drawn from $p(y \mid x)$, with $\mathbb{E}[y \mid x] = f(x)$, and x drawn adversarially. Then, the worst case bound on the number of samples for which

$$Pr\left\{|\hat{\mu}(x) - f(x)| \geq \epsilon_1\right\} \leq \delta_1, \forall x \in U \tag{C.1}$$

and the GP is forced to return $\perp$ is at most

$$m = \left(\frac{4V_m^2}{\epsilon_1^2}\log\left(\frac{2}{\delta_1}\right)\right)\mathcal{N}_U\left(r\left(\frac{1}{2}\sigma^2_{tol}\right)\right). \tag{C.2}$$

Furthermore, $\mathcal{N}_U\left(r\left(\frac{1}{2}\sigma^2_{tol}\right)\right)$ grows polynomially with $\frac{1}{\epsilon_1}$ and V_m for the RBF kernel.

Proof sketch The proof is identical to that of Theorem C.1.

Intuitively, the KWIK learnability of GPs can be explained as follows. By knowing the value at a certain point within some tolerance $\frac{\epsilon}{2}$, the Lipschitz smoothness assumption means there is a nonempty region around this point where values are known within a larger tolerance ϵ. Therefore, given sufficient observations in a neighborhood, a GP is able to

generalize its learned values to other nearby points. Lemma 3.1 relates the error at any of these points to the predictive variance of the GP, so a KWIK agent using a GP can use the variance prediction to choose whether to predict $\perp$ or not. As a function becomes less smooth, the size of these neighborhoods shrinks, increasing the covering number, and the number of points required to learn over the entire input space increases.

Theorem C.1, combined with the KWIK-Rmax Theorem (Theorem 3 of [77]) establishes that the previously proposed GP-Rmax algorithm [61], which learns GP representations of T and R and replaces uncertainty with $V_{\max}$ values, is indeed PAC-MDP. GP-Rmax was empirically validated in many domains in this previous work but the PAC-MDP property was not formally proven. However, GP-Rmax relies on a planner that can derive Q^* from the learned T and R, which may be infeasible in continuous domains, especially for real-time control.

C.5 GPs for Model-Free RL

Model-free RL algorithms, such as Q-learning [128], are often used when planning is infeasible due to time constraints or the size of the state space. These algorithms do not store T and R but instead try to model Q^* directly through incremental updates of the form: $Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha\left(r_t + \gamma V_t^*(s_{t+1})\right)$. Unfortunately, most Q-learning variants have provably exponential sample complexity, including optimistic/greedy Q-learning with a linearly decaying learning rate [44], due to the incremental updates to Q combined with the decaying α term.

However, the more recent Delayed Q-learning (DQL) [115] *is* PAC-MDP. This algorithm works by initializing $Q(s,a)$ to $V_{\max}$ and then *overwriting* $Q(s,a) = \frac{\sum_{i=1}^{m} r_i + \gamma V(s_i')}{m}$ after m samples of that $\langle s,a \rangle$ pair have been seen. In section C.5.1, it is shown that using a single GP results in exponential sample complexity, like Q-learning; in section C.5.2, it is shown that using a similar overwriting approach to DQL achieves sample efficiency.

C.5.1 Naïve Model-free Learning using GPs

It is now shown that a naïve use of a single GP to model Q^* will result in exponentially slow convergence similar to greedy Q-learning with a linearly decaying α. Consider the following model-free algorithm using a GP to store values of $\hat{Q} = Q_t$. A GP for each action (GP_a) is initialized optimistically using the prior. At each step, an action is chosen greedily and GP_a is updated with an input/output sample: $\langle x_t, r_t + \gamma \max_b GP_b(s_{t+1})\rangle$. The worst-case performance is analyzed through a toy example and show that this approach requires an exponential number of samples to learn Q^*. This slow convergence is intrinsic to GPs due to the variance reduction rate and the non-stationarity of the $\hat{Q}$ estimate (as opposed to T and R, whose sampled values do not depend on the learner's current estimates). So, any model-free algorithm using a GP that does not reinitialize the variance will have the same worst-case complexity as in this toy example.

Consider an MDP with one state s and one action a that transitions deterministically back to s with reward $r = 0$, and discount factor γ. The Q function is initialized optimistically to $\hat{Q}_0(s) = 1$ using the GP's prior mean. Consider the naïve GP learning algorithm described above, kernel $k(s, s) = 1$, and measurement noise ω^2 to predict the Q-function using the GP regression equations. The behavior of the algorithm can be analyzed using induction.

Consider the first iteration: $\hat{Q}_0 = 1$, and the first measurement is γ. In this case, the GP prediction is equivalent to the MAP estimate of a random variable with Gaussian prior and linear measurements subject to Gaussian noise.

$$\hat{Q}_1 = \frac{\omega^2}{\sigma_0^2 + \omega^2} 1 + \frac{\sigma_0^2}{\sigma_0^2 + \omega^2}\gamma \tag{C.3}$$

Recursively, $\hat{Q}_{i+1} = (\frac{\omega^2}{\sigma_i^2+\omega^2} + \frac{\sigma_i^2}{\sigma_i^2+\omega^2}\gamma)\hat{Q}_i$ where $\sigma_i^2 = \frac{\omega^2\sigma_0^2}{\omega^2+i\sigma_0^2}$ is the GP variance at iteration i. Substituting for σ_i^2 and rearranging yields a recursion for the prediction of $\hat{Q}_i$ at each iteration,

$$\hat{Q}_{i+1} = \frac{\omega^2 + \sigma_0^2(i+\gamma)}{\omega^2 + \sigma_0^2(i+1)}\hat{Q}_i \tag{C.4}$$

From [44], a series of the form $\hat{Q}_{i+1} = \frac{i+\gamma}{i+1}\hat{Q}_i$ will converge to ϵ exponentially slowly in

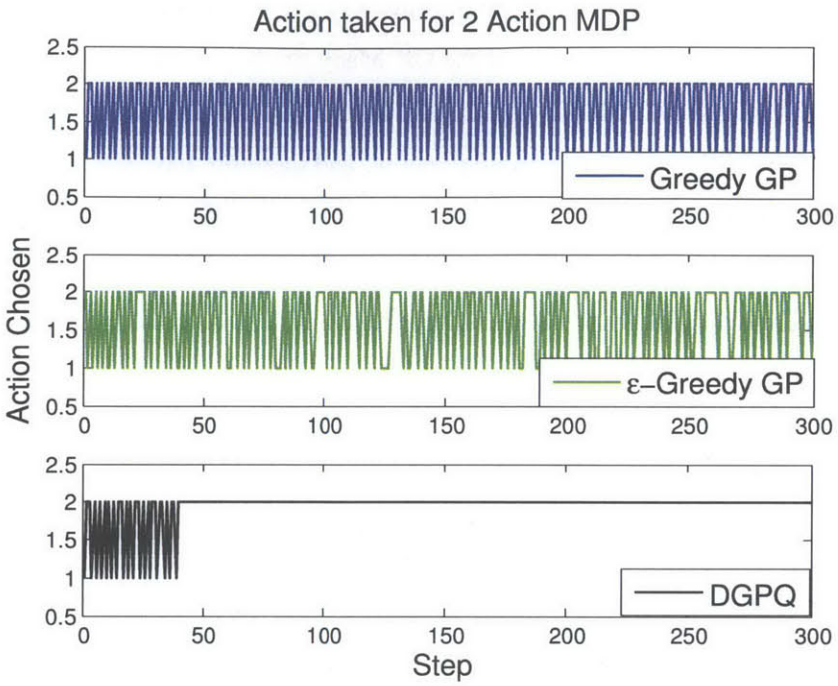


Figure C-1: Single-state MDP results: DGPQ converges quickly to the optimal policy while the naïve GP implementations oscillate.

terms of $\frac{1}{\epsilon}$ and $\frac{1}{1-\gamma}$. However, for each term in the series,

$$\frac{\frac{\omega^2}{\sigma_0^2} + i + \gamma}{\frac{\omega^2}{\sigma_0^2} + i + 1} \geq \frac{i+\gamma}{i+1} \tag{C.5}$$

The modulus of contraction is always at least as large as the RHS, so the series convergence to ϵ (since the true value is 0) is at least as slow as the example in [44]. Therefore, the naïve GP implementation's convergence to ϵ is also exponentially slow, and, in fact, has the same learning speed as Q-learning with a linear learning rate. This is because the variance of a GP decays linearly with number of observed data points, and the magnitude of GP updates is proportional to this variance.

Additionally, by adding a second action with reward $r = 1 - \gamma$, a greedy naïve agent would oscillate between the two actions for an exponential (in $\frac{1}{1-\gamma}$) number of steps, as shown in Figure C-1 (blue line), meaning the algorithm is not PAC-MDP, since the other action is not near optimal. Randomness in the policy with ϵ-greedy exploration (green

line) *will not* improve the slowness, which is due to the non-stationarity of the Q-function and the decaying update magnitudes of the GP. Many existing model-free GP algorithms, including GP-Sarsa [43], iGP-Sarsa [32], and (non-delayed) GPQ [29] perform exactly such GP updates without variance reinitialization, and therefore have the same worst-case exponential sample complexity.

C.5.2 Delayed GPQ for model-free RL

This section proposes a new algorithm, inspired by Delayed Q-learning, that guarantees polynomial sample efficiency by consistently reinitializing the variance of the GP when many observations differ significantly from the current $\hat{Q}$ estimate.

Algorithm 12 Delayed GPQ (DGPQ)

Input: GP kernel $k(\cdot,\cdot)$, Lipschitz Constant L_Q, Environment Env, Actions A, initial state s_0, discount γ, threshold σ^2_{tol}, ϵ_1
for $a \in A$ **do**
  $\hat{Q}_a = \emptyset$
  $GP_a = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot))$
end for
for each timestep t **do**
  $a_t = \arg\max_a \hat{Q}_a(s_t)$ by Eq C.6
  $\langle r_t, s_{t+1}\rangle = Env.\text{takeAct}(a_t)$
  $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
  $\sigma^2_1 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma^2_1 > \sigma^2_{tol}$ **then**
    $GP_{a_t}.\text{update}(s_t, q_t)$
  end if
  $\sigma^2_2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma^2_1 > \sigma^2_{tol} \geq \sigma^2_2$ **and**
  $\hat{Q}_{a_t}(s_t) - GP_{a_t}.\text{mean}(s_t) > 2\epsilon_1$ **then**
    $\hat{Q}_a.\text{update}(s_t, GP_{a_t}.\text{mean}(s_t) + \epsilon_1)$
    $\forall a \in A, GP_a = GP.\text{init}(\mu = \hat{Q}_a, k(\cdot,\cdot))$
  end if
end for

Delayed GPQ-Learning (DGPQ: Algorithm 12) maintains *two* representations of the value function. The first is a set of GPs, GP_a for each action, that is updated after each step but *not* used for action selection. The second representation of the value function, which stores values from previously converged GPs is denoted $\hat{Q}(s,a)$. Intuitively, the algorithm

uses $\hat{Q}$ as a "safe" version of the value function for choosing actions and retrieving backup values for the GP updates, and only updates $\hat{Q}(s,a)$ when GP_a converges to a significantly different value at s.

The algorithm chooses actions greedily based on $\hat{Q}$ (line 7) and updates the corresponding GP_a based on the observed rewards and $\hat{Q}$ at next the state (line 9). Note, in practice one creates a prior of $\hat{Q}_a(s_t)$ (line 17) by updating the GP with points $z_t = q_t - \hat{Q}_a(s_t)$ (line 12), and adding back $\hat{Q}_a(s_t)$ in the update on line 16. If the GP has just crossed the convergence threshold at point s and learned a value significantly lower ($2\epsilon_1$) than the current value of $\hat{Q}$ (line 15), the representation of $\hat{Q}$ is partially overwritten with this new value plus a bonus term using an operation described below. Crucially, the GPs are then *reset*, with all data erased. This reinitializes the variance of the GPs so that updates will have a large magnitude, avoiding the slowed convergence seen in the previous section. Guidelines for setting the sensitivity of the two tests (σ^2_{tol} and ϵ_1) are given in Section C.6.

There are significant parallels between DGPQ and the discrete-state DQL algorithm [114], but the advancements are non-trivial. First, both algorithms maintain two Q functions, one for choosing actions ($\hat{Q}$), and a temporary function that is updated on each step (GP_a), but in DGPQ specific representations have been chosen to handle continuous states and still maintain optimism and sample complexity guarantees. DQL's counting of (up to m) discrete state visits for each state cannot be used in continuous spaces, so DGPQ instead checks the immediate convergence of GP_a at s_t to determine if it should compare $\hat{Q}(s_t,a)$ and $GP_a(s_t)$. Lastly, DQL's discrete learning flags are not applicable in continuous spaces, so DGPQ only compares the two functions as the GP variance crosses a threshold, thereby partitioning the continuous MDP into areas that are known (w.r.t. $\hat{Q}$) and unknown, a property that will be vital for proving the algorithm's convergence.

One might be tempted to use yet another GP to model $\hat{Q}(s,a)$. However, it is difficult to guarantee the optimism of $\hat{Q}$ ($\hat{Q} \geq Q^* - \epsilon$) using a GP, which is a crucial property of most sample-efficient algorithms. In particular, if one attempts to update/overwrite the local prediction values of a GP, unless the kernel has finite support (a point's value only influences a finite region), the overwrite will affect the values of all points in the GP and possibly cause a point that was previously correct to fall below $Q^*(s,a) - \epsilon$.

In order to address this issue, an alternative function approximator for $\hat{Q}$ is used which includes an optimism bonus as used in C-PACE [95]. Specifically, $\hat{Q}(s,a)$ is stored using a set of values that have been updated from the set of GPs, $(\langle s_i, a_i \rangle, \hat{\mu}_i)$ and a Lipschitz constant L_Q that is used to find an optimistic upper bound for the Q function for $\langle s, a \rangle$:

$$\hat{Q}(s,a) = \min \left\{ \min_{\langle s_i, a \rangle \in BV} \hat{\mu}_i + L_Q d((s,a),(s_i,a)), V_{\max} \right\} \tag{C.6}$$

The set (s_i, a) is referred to as as the set of *basis vectors* (BV). Intuitively, the basis vectors store values from the previously learned GPs. Around these points, Q^* cannot be greater than $\hat{\mu}_i + L_Q d((s,a),(s_i,a))$ by continuity. To predict optimistically at points not in BV, the algorithm searches over BV for the point with the lowest prediction including the weighted distance bonus. If no point in BV is sufficiently close, then V_{max} is used instead. This representation is also used in C-PACE [95] but here it simply stores the optimistic Q-function; DGPQ does not perform a fixed point operation. Note the GP still plays a crucial role in Algorithm 12 because its confidence bounds, which are not captured by $\hat{Q}$, partition the space into known/unknown areas that are critical for controlling updates to $\hat{Q}$.

In order to perform an update (partial overwrite) of $\hat{Q}$, the algorithm adds an element $\langle (s_i, a_i), \hat{\mu}_i \rangle$ to the basis vector set. Redundant constraints are eliminated by checking if the new constraint results in a lower prediction value at other basis vector locations. Thus, the pseudocode for updating $\hat{Q}$ is as follows: add point $\langle (s_i, a_i), \hat{\mu}_i \rangle$ to the basis vector set; if for any j, $\mu_i + L_Q d((s_i,a),(s_j,a)) \leq \mu_j$, delete $\langle (s_j, a_j), \hat{\mu}_j \rangle$ from the set.

Figure C-1 shows the advantage of this technique over the naïve GP training discussed earlier. DGPQ learns the optimal action within 50 steps, while the naïve implementation as well as an ϵ-greedy variant both oscillate between the two actions. This example illustrates that the targeted exploration of DGPQ is of significant benefit compared to untargeted approaches, including the ϵ-greedy approach of GP-SARSA.

C.6 The Sample Complexity of DGPQ

In order to prove that DGPQ is PAC-MDP a proof structure similar to that of DQL [114] is adopted and referred to throughout. First the DQL definition of a "known state" MDP M_{K_t}

is extended, i.e. states containing state/actions from $\hat{Q}$ that have low Bellman residuals.

Definition C.3 During timestep t of DGPQ's execution with $\hat{Q}$ as specified and $\hat{V}(s) = \max_a \hat{Q}(s,a)$, the set of **known states** is given by $K_t = \{\langle s,a\rangle | \hat{Q}(s,a) - (R(s,a) + \gamma \int_{s'} T(s'|s,a)\hat{V}(s')ds') \leq 3\epsilon_1\}$. M_K contains the same MDP parameters as M for $\langle s,a\rangle \in K$ and values of $V_{\max}$ for $\langle s,a\rangle \notin K$.

Three sufficient conditions for proving an algorithm with greedy policy values V_t is PAC-MDP are repeated here (from Theorem 10 of [114]) (1) Optimism: $\hat{V}_t(s) \geq V^*(s) - \epsilon$ for all timesteps t. (2) Accuracy with respect to M_{K_t}'s values: $\hat{V}_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$ for all t. (3) The number of updates to $\hat{Q}$ and the number of times a state outside M_K is reached is bounded by a polynomial function of $\langle \mathcal{N}_S(r), \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma}\rangle$.

The proof structure is to develop lemmas that prove these three properties. After defining relevant structures and concepts, Lemmas C.1 and C.2 bound the number of possible changes and possible *attempts* to change $\hat{Q}$. The "typical" behavior of the algorithm is defined in Definition C.5 and show this behavior occurs with high probability in Lemma C.3. These conditions help ensure property 2 above. After that, Lemma C.4 shows that the function stored in $\hat{Q}$ is always optimistic, fulfilling property 1. Finally, property 3 is shown by combining Theorem C.1 (number of steps before the GP converges) with the number of updates to $\hat{Q}$ from Lemma C.1, as formalized in Lemma C.6.

An "update" (as well as "successful update") to $\hat{Q}$ is now defined, which extends definitions from the original DQL analysis.

Definition C.4 An **update** (or **successful update**) of state-action pair (s,a) is a timestep t for which a change to $\hat{Q}$ (an overwrite) occurs such that $\hat{Q}_t(s,a) - \hat{Q}_{t+1}(s,a) > \epsilon_1$. An **attempted update** of state-action pair (s,a) is a timestep t for which $\langle s,a\rangle$ is experienced and $GP_{a,t}.\text{Var}(s) > \sigma^2_{tol} \geq GP_{a,t+1}.\text{Var}(s)$. An attempted update that is not successful (does not change $\hat{Q}$) is an **unsuccessful update**.

The total number of updates to $\hat{Q}$ is now bounded by a polynomial term κ based on the concepts defined above.

Lemma C.1 The total number of successful updates (overwrites) during any execution of DGPQ with $\epsilon_1 = \frac{1}{3}\epsilon(1-\gamma)$ is

$$\kappa = |A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)\left(\frac{3R_{max}}{(1-\gamma)^2\epsilon}+1\right) \tag{C.7}$$

Proof The proof proceeds by showing that the bound on the number of updates in the single-state case from Section C.5.1 is, $\left(\frac{3R_{max}}{(1-\gamma)^2\epsilon}+1\right)$ based on driving the full function from $V_{\max}$ to $V_{\min}$. This quantity is then multiplied by the covering number and the number of actions. See the Supplementary Material of [52].

The next lemma bounds the number of attempted updates. The proof is similar to the DQL analysis and shown in the Supplementary Material of [52].

Lemma C.2 The total number of **attempted** updates (overwrites) during any execution of GPQ is $|A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)$.

Define the following event, called A2, to link back to the DQL proof. A2 describes the situation where a state/action that currently has an inaccurate value (high Bellman residual) with respect to $\hat{Q}$ is observed m times and this causes a successful update.

Definition C.5 Define **Event A2** to be the event that for all timesteps t, if $\langle s,a\rangle \notin K_{k_1}$ and an attempted update of $\langle s,a\rangle$ occurs during timestep t, the update will be successful, where $k_1 < k_2 < ... < k_m = t$ are the timesteps where $GP_{a_k}.\text{Var}(s_k) > \sigma^2_{tol}$ since the last update to $\hat{Q}$ that affected $\langle s,a\rangle$.

One can set an upper bound on m, the number of experiences required to make $GP_{a_t}.\text{Var}(s_t) < \sigma^2_{tol}$, based on m in Theorem C.1 from earlier. In practice m will be much smaller but unlike discrete DQL, DGPQ does not need to be given m, since it uses the GP variance estimate to decide if sufficient data has been collected. The following lemma shows that with high probability, a failed update will not occur under the conditions of event A2.

Lemma C.3 By setting

$$\sigma^2_{tol} = \frac{2\omega_n^2\epsilon^2(1-\gamma)^4}{9R^2_{max}\log(\frac{6}{\delta}\mathcal{N}_c\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa))} \tag{C.8}$$

it is ensured that the probability of event A2 occurring is $\geq 1 - \delta/3$.

Proof The maximum number of attempted updates is $\mathcal{N}_c\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)$ from Lemma C.2. Therefore, by setting $\delta_1 = \frac{\delta}{3\mathcal{N}_c\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)}$, $\epsilon_1 = \frac{1}{3}\epsilon(1-\gamma)$, and $V_m = \frac{R_{max}}{1-\gamma}$, Lemma 3.1 states that during each overwrite, there is no greater than probability $\delta_1 = \frac{\delta}{3\mathcal{N}_c\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)}$ of an incorrect update. Applying the union bound over all of the possible attempted updates, the total probability of A2 not occuring is $\frac{\delta}{3}$.

The next lemma shows the optimism of $\hat{Q}$ for all timesteps with high probability $\frac{\delta}{3}$. The proof structure is similar to Lemma 3.10 of [95]. See supplemental material of [52].

Lemma C.4 During execution of DGPQ, $Q^*(s,a) \leq Q_t(s,a) + \frac{2\epsilon_1}{1-\gamma}$ holds for all $\langle s,a \rangle$ with probability $\frac{\delta}{3}$.

The next Lemma connects an unsuccessful update to a state/action's presence in the known MDP M_K.

Lemma C.5 If event A2 occurs, then if an unsuccessful update occurs at time t and $GP_a.\text{Var}(s) < \sigma_{tol}^2$ at time $t+1$ then $\langle s,a \rangle \in K_{t+1}$.

Proof The proof is by contradiction and shows that an unsuccessful update in this case implies a previously successful update that would have left $GP_a.\text{Var}(s) \geq \sigma_{tol}^2$. See the Supplemental material of [52].

The final lemma bounds the number of encounters with state/actions not in K_t, which is intuitively the number of points that make updates to the GP from Theorem C.1 times the number of changes to $\hat{Q}$ from Lemma C.1. The proof is in the Supplemental Material of [52].

Lemma C.6 Let $\eta = \mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)$. If event A2 occurs and $\hat{Q}_t(s,a) \geq Q^*(s,a) - \frac{2\epsilon_1}{1-\gamma}$ holds for all t and $\langle s,a \rangle$ then the number of timesteps ζ where $\langle s_t, a_t \rangle \notin K_t$ is at most

$$\zeta = m|A|\eta\left(\frac{3R_{max}}{(1-\gamma)^2\epsilon} + 1\right) \tag{C.9}$$

where

$$m = \left(\frac{36R_{max}^2}{(1-\gamma)^4\epsilon^2}\log\left(\frac{6}{\delta}|A|\eta(1+\kappa)\right)\right)|A|\eta \tag{C.10}$$

Finally, the PAC-MDP result is stated for DGPQ, which is an instantiation of the General PAC-MDP Theorem (Theorem 10) from [114].

Theorem C.2 Given real numbers $0 < \epsilon < \frac{1}{1-\gamma}$ and $0 < \delta < 1$ and a continuous MDP M there exist inputs σ_{tol}^2 (see (C.8)) and $\epsilon_1 = \frac{1}{3}\epsilon(1-\gamma)$ such that DGPQ executed on M will be PAC-MDP by Definition C.2 with only

$$\frac{R_{\max}\zeta}{\epsilon(1-\gamma)^2}\log\left(\frac{1}{\delta}\right)\log\left(\frac{1}{\epsilon(1-\gamma)}\right) \tag{C.11}$$

timesteps where $Q_t(s_t, a_t) < V^*(s_t) - \epsilon$, where ζ is defined in Equation (C.9).

The proof of the theorem is the same as Theorem 16 by [114], but with the updated lemmas from above. The three crucial properties hold when A2 occurs, which Lemma C.3 guarantees with probability $\frac{\delta}{3}$. Property 1 (optimism) holds from Lemma C.4. Property 2 (accuracy) holds from the Definition C.3 and analysis of the Bellman Equation as in Theorem 16 by [114]. Finally, property 3, the bounded number of updates and escape events, is proven by Lemmas C.1 and C.6.

C.7 Empirical Results

The first experiment is in a 2-dimensional square over $[0, 1]^2$ designed to show the computational disparity between C-PACE and DGPQ. The agent starts at $[0, 0]$ with a goal of reaching within a distance of 0.15 of $[1, 1]$. Movements are 0.1 in the four compass directions with additive uniform noise of ± 0.01. The $\mathcal{L}_1$ distance metric is used with $L_Q = 9$ and an RBF kernel with $\theta = 0.05$, $\omega_n^2 = 0.1$ for the GP. Figure C-2 shows the number of steps needed per episode (capped at 200) and computational time per step used by C-PACE and DGPQ. C-PACE reaches the optimal policy in fewer episodes but requires orders of magnitude more computation during steps with fixed point computations. Such planning times of over 10s are unacceptable in time sensitive domains, while DGPQ only takes 0.003s per step.

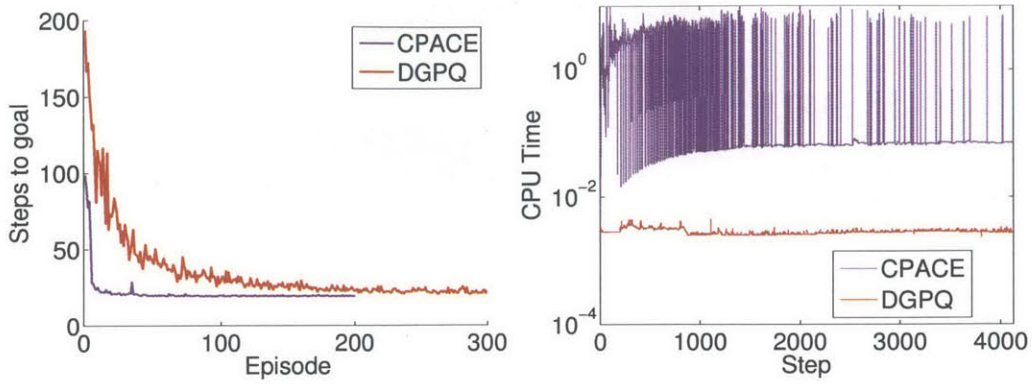


Figure C-2: Average (10 runs) steps to the goal and computation time for C-PACE and DGPQ on the square domain.

In the second domain, DGPQ stabilizes a simulator of the longitudinal dynamics of an unstable F16 with linearized dynamics, which motivates the need for a real-time computable policy. The five dimensional state space contains height, angle of attack, pitch angle, pitch rate, and airspeed. Details of the simulator can be found in [112]. The reward $r = -|h - h_d|/100\text{ft} - |\dot{h}|/100\text{ft/s} - |\delta_e|$ is used, with aircraft height h, desired height h_d and elevator angle (degrees) δ_e. The control input was discretized as $\delta_e \in \{-1, 0, 1\}$ and the elevator was used to control the aircraft. The thrust input to the engine was fixed as the reference thrust command at the (unstable) equilibrium. The simulation time step size was 0.05s and at each step, the air speed was perturbed with Gaussian noise $\mathcal{N}(0, 1)$ and the angle of attack was perturbed with Gaussian noise $\mathcal{N}(0, 0.01^2)$. A RBF kernel with $\theta = 0.05$, $\omega_n^2 = 0.1$ was used. The initial height was h_d, and if $|h - h_d| \geq 200$, then the vertical velocity and angle of attack were set to zero to act as boundaries.

DGPQ learns to stabilize the aircraft using less than 100 episodes for $L_Q = 5$ and about 1000 episodes for $L_Q = 10$. The disparity in learning speed is due to the curse of dimensionality. As the Lipschitz constant doubles, $\mathcal{N}_c$ increases in each dimension by 2, resulting in a 2^5 increase in $\mathcal{N}_c$. DGPQ requires on average 0.04s to compute its policy at each step, which is within the 20Hz command frequency required by the simulator. While the maximum computation time for DGPQ was 0.11s, the simulations were run in MATLAB so further optimization should be possible. Figure C-3 shows the average reward of DGPQ in this domain using $L_Q = \{5, 10\}$. C-PACE was also run in this domain but its

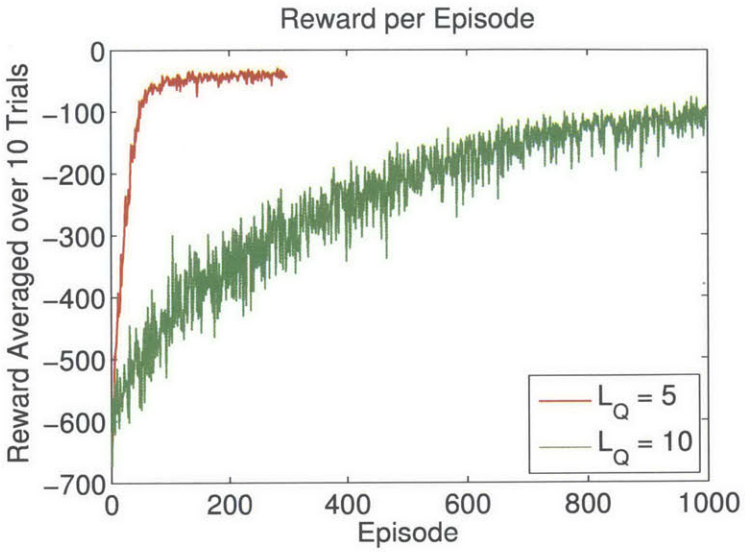


Figure C-3: Average (10 runs) reward on the F16 domain.

computation time reached over 60s per step in the first episode, well beyond the desired 20 Hz command rate.

C.8 Conclusions

This work provides sample efficiency results for using GPs in RL. In section C.4, GPs are proven to be usable in the KWIK-Rmax MBRL architecture, establishing the previously proposed algorithm GP-Rmax as PAC-MDP. In section C.5.1, it is proven that existing model-free algorithms using a single GP have exponential sample complexity, connecting to seemingly unrelated negative results on Q-learning learning speeds. Finally, the development of DGPQ provides the first provably sample efficient model-free (without a planner or fixed-point computation) RL algorithm for general continuous spaces.

Bibliography

[1] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[2] Rakshit Allamaraju, Hassan Kingravi, Allan Axelrod, Girish Chowdhary, Robert Grande, Jonathan P. How, Christopher Crick, and Weihua Sheng. Human aware uas path planning in urban environments using nonstationary mdps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2014.

[3] John Asmuth, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate. A bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 19–26. AUAI Press, 2009.

[4] Karl Johan Aström and Björn Wittenmark. *Adaptive Control*. Addison-Weseley, Readings, 2nd edition, 1995.

[5] Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *NIPS*, pages 89–96, 2008.

[6] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Algorithmic Foundations of Robotics X*, pages 475–491. Springer, 2013.

[7] Michèle Basseville and Igor V. Nikiforov. *Detection of abrupt changes: theory and application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[8] David M. Blei and Michael I. Jordan. Variational inference for dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–144, 2006.

[9] Justin Boyan and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Neural Information Processing Systems 7*, pages 369–376, 1995.

[10] Stephan Boyd and Sankar Sastry. Necessary and sufficient conditions for parameter convergence in adaptive control. *Automatica*, 22(6):629–639, 1986.

[11] Ronen Brafman and Moshe Tennenholtz. R-Max - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research (JMLR)*, 3:213–231, 2002.

[12] Ronen I. Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[13] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael Jordan. Streaming variational bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735, 2013.

[14] Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. *arXiv preprint arXiv:1309.6821*, 2013.

[15] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[16] A. Calise, N. Hovakimyan, and M. Idan. Adaptive output feedback control of nonlinear systems using neural networks. *Automatica*, 37(8):1201–1211, 2001. Special Issue on Neural Networks for Feedback Control.

[17] Chengyu Cao and N. Hovakimyan. Design and analysis of a novel adaptive control architecture with guaranteed transient performance. *Automatic Control, IEEE Transactions on*, 53(2):586 –591, march 2008.

[18] Iadine Chads, Guillaume Chapron, Marie-Jose Cros, Frdrick Garcia, and Rgis Sabbadin. Markov decision processes (MDP) toolbox. `http://www7.inra.fr/mia/T/MDPtoolbox/MDPtoolbox.html`, 2012.

[19] G. Chowdhary, H. Kingravi, R.C. Grande, J.P. How, and P. Vela. Nonparametric adaptive control using gaussian processes. In *AIAA Guidance, Navigation, and Control Conference (GNC)*. IEEE, 2013.

[20] G. Chowdhary, H. Kingravi, J.P. How, and P. Vela. Nonparametric adaptive control using gaussian processes. In *IEEE Conference on Decision and Control (CDC)*. IEEE, 2013.

[21] Girish Chowdhary. *Concurrent Learning for Convergence in Adaptive Control Without Persistency of Excitation*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, December 2010.

[22] Girish Chowdhary, Eric Johnson, Rajeev Chandramohan, Scott M. Kimbrell, and Anthony Calise. Autonomous guidance and control of airplanes under actuator failures and severe structural damage. *Journal of Guidane Control and Dynamics*, 2013. accepted.

[23] Girish Chowdhary and Eric N. Johnson. Concurrent learning for convergence in adaptive control without persistency of excitation. In *49th IEEE Conference on Decision and Control*, pages 3674–3679, 2010.

[24] Girish Chowdhary and Eric N. Johnson. Concurrent learning for improved parameter convergence in adaptive control. In *AIAA Guidance Navigation and Control Conference*, Toronto, Canada, 2010.

[25] Girish Chowdhary, Eric N. Johnson, Rajeev Chandramohan, Scott M. Kimbrell, and Anthony Calise. Autonomous guidance and control of airplanes under actuator failures and severe structural damage. *Journal of Guidance Control and Dynamics*, 2012. in-press.

[26] Girish Chowdhary, Hassan Kingravi, Jonathan P. How, and Patricio Vela. Bayesian nonparametric adaptive control of time varying systems using Gaussian processes. In *American Control Conference (ACC)*. IEEE, 2013.

[27] Girish Chowdhary, Hassan Kingravi, Jonathan P. How, and Patricio A. Vela. Nonparameteric adaptive control of time varying systems using gaussian processes. Aerospace control laboratory technical report, Massachusetts Institute of Technology, http://hdl.handle.net/1721.1/71875, March 2013.

[28] Girish Chowdhary, Hassan Kingravi, Jonathan P. How, and Patricio A. Vela. Bayesian nonparametric adaptive control using gaussian processes. *IEEE Transactions on Neural Networks*, 2013 (submitted).

[29] Girish Chowdhary, Miao Liu, Robert C. Grande, Thomas J. Walsh, Jonathan P. How, and Lawrence Carin. Off-policy reinforcement learning with gaussian processes. *Acta Automatica Sinica*, To appear, 2014.

[30] Girish Chowdhary, Tongbin Wu, Mark Cutler, and Jonathan P. How. Rapid transfer of controllers between UAVs using learning based adaptive control. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013.

[31] Girish Chowdhary, Tansel Yucelen, Maximillian Mühlegg, and Eric N Johnson. Concurrent learning adaptive control of linear systems with exponentially convergent bounds. *International Journal of Adaptive Control and Signal Processing*, 2012.

[32] Jen Jen Chung, Nicholas R. J. Lawrance, and Salah Sukkarieh. Gaussian processes for informative exploration in reinforcement learning. In *ICRA*, pages 2633–2639, 2013.

[33] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

[34] Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

[35] Mark Cutler. Design and Control of an Autonomous Variable-Pitch Quadrotor Helicopter. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, August 2012.

[36] Mark Cutler and Jonathan P. How. Actuator constrained trajectory generation and control for variable-pitch quadrotors. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Minneapolis, Minnesota, August 2012.

[37] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 465–472, 2011.

[38] Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes.* PhD thesis, Karlsruhe Institute of Technology, 2010.

[39] Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472, 2011.

[40] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputation*, 72(7-9):1508–1524, March 2009.

[41] Thomas Desautels, Andreas Krause, and Joel W. Burdick. Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization. In *ICML*, 2012.

[42] W. Durham. Constrained control allocation. *AIAA Journal of Guidance, Control, and Dynamics*, 16:717–772, 1993.

[43] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *International Conference on Machine Learning (ICML)*, 2005.

[44] Eyal Even-Dar and Yishay Mansour. Learning rates for q-learning. *Journal of Machine Learning Research*, 5:1–25, 2004.

[45] Sarah Ferguson, Brandon Luders, Robert Grande, and Jonathan How. Real-time predictive modeling and robust avoidance of pedestrians with uncertain, changing intentions. In *Workshop on Algorithmic Foundations of Robotics*. Springer, 2014 (submitted).

[46] Peter Finn. Domestic use of aerial drones by law enforcement likely to prompt privacy debate. *Washington Post*, 22, 2011.

[47] Roman Garnett, Michael A Osborne, Steven Reece, Alex Rogers, and Stephen J Roberts. Sequential bayesian prediction in the presence of changepoints and faults. *The Computer Journal*, 53(9):1430–1446, 2010.

[48] Agathe Girard, Carl Edward Rasmussen, Joaquin Quintero-Candela, and Roderick Murray-smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems*, pages 529–536. MIT Press, 2003.

[49] Robert Grande, Girish Chowdhary, and Jonathan How. Experimental validation of Bayesian nonparametric adaptive control using Gaussian processes. *Journal of Aerospace Information Systems*, 2014.

[50] Robert Grande, Girish Chowdhary, and Jonathan P. How. Nonparametric adaptive control using Gaussian processes with online hyperparameter estimation. In *IEEE Conference on Decision and Control (CDC)*. IEEE, 2013.

[51] Robert C. Grande, Thomas J. Walsh, Girish Chowdhary, Sarah Ferguson, and Jonathan P. How. Online regression for data with changepoints using Gaussian processes and reusable models. In *Conference on Uncertainty in Artificial Intelligence*, pages 19–26. AUAI Press, 2014 (Submitted).

[52] Robert C. Grande, Thomas J. Walsh, and Jonathan P. How. Sample efficient reinforcement learning with Gaussian processes. In *International Conference on Machine Learning (ICML)*, 2014.

[53] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[54] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, 28(2):51–64, April 2008.

[55] Jonathan P. How, Emilio Frazzoli, and Girish Chowdhary. *Handbook of Unmanned Aerial Vehicles*, chapter Linear Flight Contol Techniques for Unmanned Aerial Vehicles. Springer, 2012.

[56] J.P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-Time Indoor Autonomous Vehicle Test Environment. *IEEE control systems*, 28(2):51–64, 2008.

[57] Petros A. Ioannou and Jung Sun. *Robust Adaptive Control*. Prentice-Hall, Upper Saddle River, 1996.

[58] Eric Johnson and Suresh Kannan. Adaptive trajectory control for autonomous helicopters. *Journal of Guidance Control and Dynamics*, 28(3):524–538, May 2005.

[59] Eric N. Johnson and Anthony J. Calise. Limited authority adaptive flight control for reusable launch vehicles. Journal of Guidance, Control and Dynamics, 2001.

[60] Tobias Jung and Peter Stone. Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration. In *Machine Learning and Knowledge Discovery in Databases*, pages 601–616. Springer, 2010.

[61] Tobias Jung and Peter Stone. Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration. In *European Conference on Machine Learning (ECML)*, September 2010.

[62] Tobias Jung and Peter Stone. Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In *European COnference on Machine Learning (ECML)*, 2012.

[63] Shivaram Kalyanakrishnan and Peter Stone. An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 749–756, 2009.

[64] Suresh Kannan. *Adaptive Control of Systems in Cascade with Saturation.* PhD thesis, Georgia Institute of Technology, Atlanta Ga, 2005.

[65] H. K. Khalil. *Nonlinear Systems.* Macmillan, New York, 2002.

[66] Nakawan Kim. *Improved Methods in Neural Network Based Adaptive Output Feedback Control, with Applications to Flight Control.* PhD thesis, Georgia Institute of Technology, Atlanta Ga, 2003.

[67] Y. H. Kim and F.L. Lewis. *High-Level Feedback Control with Neural Networks*, volume 21 of *Robotics and Intelligent Systems*. World Scientific, Singapore, 1998.

[68] H. A. Kingravi, G. Chowdhary, P. A. Vela, and E. N. Johnson. Reproducing kernel hilbert space approach for the online update of radial bases in neuro-adaptive control. *Neural Networks and Learning Systems, IEEE Transactions on*, 23(7):1130 –1141, july 2012.

[69] Jens Kober and Jan Peters. Movement templates for learning of hitting and batting. In *Learning Motor Skills*, pages 69–82. Springer, 2014.

[70] George Konidaris, Scott Kuindersma, Andrew G. Barto, and Roderic A. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *NIPS*, volume 23, pages 1162–1170, 2010.

[71] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research (JMLR)*, 9:235–284, June 2008.

[72] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149, 2003.

[73] E. Lavretsky. Combined/composite model reference adaptive control. *Automatic Control, IEEE Transactions on*, 54(11):2692 –2697, nov. 2009.

[74] Alessandro Lazaric, Emma Brunskill, et al. Sequential transfer in multi-armed bandit with finite set of models. In *Advances in Neural Information Processing Systems*, pages 2220–2228, 2013.

[75] F. L. Lewis. Nonlinear network structures for feedback control. *Asian Journal of Control*, 1:205–228, 1999. Special Issue on Neural Networks for Feedback Control.

[76] Lihong Li and Miachael L. Littman. Reducing reinforcement learning to KWIK online regression. *Annals of Mathematics and Artificial Intelligence*, 58(3-4):217–237, 2010.

[77] Lihong Li, Michael L. Littman, Thomas J. Walsh, and Alexander L. Strehl. Knows what it knows: a framework for self-aware learning. *Machine Learning*, 82(3):399–443, 2011.

[78] Lihong Li, Michael L. Littman, Thomas J. Walsh, and Alexander L. Strehl. Knows what it knows: a framework for self-aware learning. *Machine learning*, 82(3):399–443, 2011.

[79] L. Ljung. Analysis of recursive stochastic algorithms. *Automatic Control, IEEE Transactions on*, 22(4):551 – 575, aug 1977.

[80] David A McAllester. Some PAC-Bayesian theorems. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 230–234. ACM, 1998.

[81] Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

[82] Bernard Michini, Mark Cutler, and Jonathan P. How. Scalable reward learning from demonstration. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 303–308. IEEE, 2013.

[83] Jeffrey W Miller and Matthew T Harrison. A simple example of dirichlet process mixture inconsistency for the number of components. In *Advances in Neural Information Processing Systems*, pages 199–206, 2013.

[84] Claire Monteleoni and Tommi Jaakkola. Online learning of non-stationary sequences. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[85] K.-R. Müller, S. Mika, G. Rätsch, S. Tsuda, and B Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–202, 2001.

[86] Flavio Nardi. *Neural Network based Adaptive Algorithms for Nonlinear Control.* PhD thesis, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA 30332, nov 2000.

[87] K. S. Narendra and A. M. Annaswamy. Robust adaptive control in the presence of bounded disturbances. *IEEE Transactions on Automatic Control*, AC-31(4):306–315, 1986.

[88] K.S. Narendra. Neural networks for control theory and practice. *Proceedings of the IEEE*, 84(10):1385 –1406, oct 1996.

[89] Kumpati S. Narendra and Anuradha M. Annaswamy. *Stable Adaptive Systems.* Prentice-Hall, Englewood Cliffs, 1989.

[90] Jerzy Neyman and Egon S Pearson. *On the problem of the most efficient tests of statistical hypotheses.* Springer, 1992.

[91] Nhan Nguyen. Asymptotic linearity of optimal control modification adaptive law with analytical stability margins. In *Infotech@AIAA conference*, Atlanta, GA, 2010.

[92] Ronald Ortner, Daniil Ryabko, et al. Online regret bounds for undiscounted continuous reinforcement learning. In *NIPS*, pages 1772–1780, 2012.

[93] J. Park and I.W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computatations*, 3:246–257, 1991.

[94] D. Patino and D. Liu. Neural network based model reference adaptive control system. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(1):198–204, 2000.

[95] Jason Pazis and Ronald Parr. PAC optimal exploration in continuous space Markov decision processes. In *AAAI*, 2013.

[96] WD Penny and SJ Roberts. Bayesian multivariate autoregressive models with structured priors. Technical report, Oxford University, 2000.

[97] Fernando Pérez-Cruz, Steven Van Vaerenbergh, Juan José Murillo-Fuentes, Miguel Lázaro-Gredilla, and Ignacio Santamaria. Gaussian processes for nonlinear signal processing. *arXiv preprint arXiv:1303.2823*, 2013.

[98] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 697–704. ACM, 2006.

[99] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[100] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons, 2009.

[101] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, 2009.

[102] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[103] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[104] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, December 2005.

[105] Carl Edward Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems*, 2:881–888, 2002.

[106] Yunus Saatçi, Ryan D Turner, and Carl E Rasmussen. Gaussian process change point models. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 927–934, 2010.

[107] R.M. Sanner and J.-J.E. Slotine. Gaussian networks for direct adaptive control. *Neural Networks, IEEE Transactions on*, 3(6):837 –863, nov 1992.

[108] M. A. Santillo and D. S. Bernstein. Adaptive control based on retrospective cost optimization. *AIAA Journal of Guidance Control and Dynamics*, 33(2), March-April 2010.

[109] Praveen Shankar. *Self-Organizing radial basis function networks for adaptive flight control and aircraft engine state estimation.* Ph.d., The Ohio State University, Ohio, 2007.

[110] Manu Sharma, Anthony J. Calise, and J. Eric Corban. An adaptive autopilot design for guided munitions. In *AIAA Guidance, Navigation, and Control Conference*, number 4490, Boston, MA, aug 1998.

[111] Manu Sharma, Anthony J. Calise, and Seungjae Lee. Development of a reconfigurable flight control law for the X-36 tailless fighter aircraft. In *AIAA Guidance, Navigation and Control Conference*, number 3940, Denver, CO, aug 2000.

[112] B.L. Stevens and F.L. Lewis. *Aircraft Control and Simulation.* Wiley-Interscience, 2003.

[113] Florian Stimberg, Andreas Ruttor, and Manfred Opper. Bayesian inference for change points in dynamical systems with reusable states - a chinese restaurant process approach. *Journal of Machine Learning Research - Proceedings Track*, 22:1117–1124, 2012.

[114] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research (JMLR)*, 10:2413–2444, December 2009.

[115] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *ICML*, pages 881–888, 2006.

[116] Alexander L Strehl and Michael L Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pages 856–863. ACM, 2005.

[117] N. Sundararajan, P. Saratchandran, and L. Yan. *Fully Tuned Radial Basis Function Neural Networks for Flight Control.* Springer, 2002.

[118] R. Sutton and A. Barto. *Reinforcement Learning, an Introduction.* MIT Press, Cambridge, MA, 1998.

[119] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[120] Johan A.K. Suykens, Joos P.L. Vandewalle, and Bart L.R. De Moor. *Artificial Neural Networks for Modelling and Control of Non-Linear Systems*. Kluwer, Norwell, 1996.

[121] Gang Tao. *Adaptive Control Design and Analysis*. Wiley, New York, 2003.

[122] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.

[123] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[124] Nina Vaits and Koby Crammer. Re-adapting the regularization of weights for non-stationary regression. In *Proceedings of the 22nd international conference on Algorithmic learning theory*, pages 114–128, 2011.

[125] Kostyantyn Y. Volyanskyy, Wassim M. Haddad, and Anthony J. Calise. A new neuroadaptive control architecture for nonlinear uncertain dynamical systems: Beyond σ and e-modifications. *IEEE Transactions on Neural Networks*, 20(11):1707–1723, Nov 2009.

[126] Zhikun Wang, Katharina Mülling, Marc Peter Deisenroth, Heni Ben Amor, David Vogt, Bernhard Schölkopf, and Jan Peters. Probabilistic movement modeling for intention inference in human-robot interaction. *The International Journal of Robotics Research*, 2013.

[127] C. J. Watkins. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[128] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[129] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022. ACM, 2007.

[130] Tansel Yucelen and Anthony Calise. Kalman filter modification in adaptive control. *Journal of Guidance, Control, and Dynamics*, 33(2):426–439, march-april 2010.