Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination

by

Brett M. Bethke
S.M., Aeronautical/Astronautical Engineering
Massachusetts Institute of Technology (2007)
S.B., Aeronautical/Astronautical Engineering
S.B., Physics
Massachusetts Institute of Technology (2005)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2010

Author .....
Department of Aeronautics and Astronautics
December 9, 2009

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

Certified by .....
Dimitri P. Bertsekas
McAfee Professor of Electrical Engineering

Certified by .....
Asuman Ozdaglar
Associate Professor of Electrical Engineering

Accepted by .....
Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination

by

Brett M. Bethke

Submitted to the Department of Aeronautics and Astronautics
on December 9, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Many sequential decision-making problems related to multi-agent robotic systems can be naturally posed as Markov Decision Processes (MDPs). An important advantage of the MDP framework is the ability to utilize stochastic system models, thereby allowing the system to make sound decisions even if there is randomness in the system evolution over time. Unfortunately, the curse of dimensionality prevents most MDPs of practical size from being solved exactly. One main focus of the thesis is on the development of a new family of algorithms for computing approximate solutions to large-scale MDPs. Our algorithms are similar in spirit to Bellman residual methods, which attempt to minimize the error incurred in solving Bellman's equation at a set of sample states. However, by exploiting kernel-based regression techniques (such as support vector regression and Gaussian process regression) with nondegenerate kernel functions as the underlying cost-to-go function approximation architecture, our algorithms are able to construct cost-to-go solutions for which the Bellman residuals are explicitly forced to zero at the sample states. For this reason, we have named our approach Bellman residual elimination (BRE). In addition to developing the basic ideas behind BRE, we present multi-stage and model-free extensions to the approach. The multi-stage extension allows for automatic selection of an appropriate kernel for the MDP at hand, while the model-free extension can use simulated or real state trajectory data to learn an approximate policy when a system model is unavailable. We present theoretical analysis of all BRE algorithms proving convergence to the optimal policy in the limit of sampling the entire state space, and show computational results on several benchmark problems.

Another challenge in implementing control policies based on MDPs is that there may be parameters of the system model that are poorly known and/or vary with time as the system operates. System performance can suffer if the model used to compute the policy differs from the true model. To address this challenge, we develop an adaptive architecture that allows for online MDP model learning and simultaneous re-computation of the policy. As a result, the adaptive architecture allows the system to continuously re-tune its control policy to account for better model information

obtained through observations of the actual system in operation, and react to changes in the model as they occur.

Planning in complex, large-scale multi-agent robotic systems is another focus of the thesis. In particular, we investigate the persistent surveillance problem, in which one or more unmanned aerial vehicles (UAVs) and/or unmanned ground vehicles (UGVs) must provide sensor coverage over a designated location on a continuous basis. This continuous coverage must be maintained even in the event that agents suffer failures over the course of the mission. The persistent surveillance problem is pertinent to a number of applications, including search and rescue, natural disaster relief operations, urban traffic monitoring, etc. Using both simulations and actual flight experiments conducted in the MIT RAVEN indoor flight facility, we demonstrate the successful application of the BRE algorithms and the adaptive MDP architecture in achieving high mission performance despite the random occurrence of failures. Furthermore, we demonstrate performance benefits of our approach over a deterministic planning approach that does not account for these failures.

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

Acknowledgments

The work presented in this thesis would not have been possible without the support of many people. First, I would like to thank my advisor, Prof. Jonathan How, for providing a challenging and intellectually stimulating research environment, and for his commitment to his students. My thesis committee members, Prof. Dimitri Bertsekas and Prof. Asu Ozdaglar, were a pleasure to work with and provided very insightful feedback on the work, especially in the area of approximate dynamic programming. I would also like to thank my thesis readers, Prof. Nick Roy and Dr. Mykel Kochenderfer, for their detailed comments and suggestions in preparing the thesis. I am truly grateful to have had the opportunity to learn and conduct research in the company of such talented, hard-working, and creative people.

I am indebted to the Hertz Foundation and the American Society for Engineering Education for their generous financial support during my graduate work. In addition, the Hertz Foundation has provided valuable advice on navigating the graduate school landscape and has introduced me to new and interesting people and ideas outside of the normal scope of my research; their commitment to and support of their graduate fellows is greatly appreciated.

Some of the work in this thesis has been done in conjunction with the Boeing Company under the contract "Vehicle And Vehicle System Health Management," and I would like to acknowledge Dr. John Vian as well as his colleagues Dr. Stefan Bieniawski, Greg Clark, Paul Pigg, Emad Saad, and Matt Vavrina. It was a great opportunity to work with our partners at Boeing, and their feedback was valuable in shaping the direction of the research. This research has also been supported in part by AFOSR grant FA9550-04-1-0458.

Throughout my graduate career, the Aerospace Controls Laboratory community provided a creative and fun environment in which to work. It was a great pleasure and privilege to work with so many talented and genuinely good people in the ACL, and I would like to thank Spencer Ahrens, Simone Airoldi, Georges Aoude, Luca Bertuccelli, Luc Brunet, Sergio Cafarelli, Han-Lim Choi, Emily Craparo, Dan Dale,

Vishnu Desaraju, Frank Fan, Adrian Frank, Cam Fraser, Alex Guzman, Ray He, Luke Johnson, Till Junge, Karl Kulling, Yoshi Kuwata, Dan Levine, Kenneth Lee, Brandon Luders, Jim McGrew, Buddy Michini, Sameera Ponda, Josh Redding, Mike Robbins, Frant Sobolic, Justin Teo, Tuna Toksoz, Glenn Tournier, Aditya Undurti, Mario Valenti, Andy Whitten, Albert Wu, and Rodrigo Zeledon for making my time at the ACL a memorable and incredibly fun one. I could not have asked for a better group of people to work with. I would also like to thank Kathryn Fischer for her tireless efforts in keeping the lab running smoothly, and for her good spirits and kindness.

Outside of the research lab, I have been fortunate to be involved in a number of other wonderful communities at MIT, and I would particularly like to thank the brothers of Zeta Beta Tau fraternity, the students of Simmons Hall, and the singers of the MIT Concert Choir for their support and friendship over the years.

Many people helped provide me with exceptional opportunities as I was growing up, many of which laid the foundation for my subsequent adventures at MIT. In particular, I would like to thank my high school physics teacher, Mr. Jameson, for taking so much time to show me the beauty of physics.

My family has been a constant source of love and support throughout my life. I would like to thank my parents, Mike and Marti, for always encouraging me to follow my dreams, for teaching me to remember the things that are truly important in life, and for making so many sacrifices to give me the opportunities I have been so fortunate to have. I would also like to thank my brother Alex, my sister Jessi, and my brother-in-law Ombeni for their wonderful smiles and sense of humor. Finally, I would like to thank my wife Anna, who has been by my side every step of the way through my graduate school experience, for her unwavering love and encouragement, and for constantly reminding me of all the beauty life has to offer. This thesis is dedicated to her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Problems in which decisions are made in stages are ubiquitous in the world: they arise in engineering, science, business, and more generally, in life. In these problems, the ability to make good decisions relies critically on balancing the needs of the present with those of the future. To present a few examples: the farmer, when choosing how many seeds to plant, must weigh the short-term profits of a larger harvest against the long-term risk of depleting the soil if too many plants are grown. The chess player, when considering a particular move, must weigh the short-term material gain of that move (tactical advantage) with its long-term consequences (strategic advantage). The student must weigh the short-term opportunity cost of obtaining an education against the long-term prospects of more desirable career opportunities.

Most such sequential decision-making problems are characterized by a high degree of complexity. This complexity arises from two main sources. First, the long-term nature of these problems means that the tree of all possible decision sequences and outcomes is typically extraordinarily large (for example, the number of possible games of chess has been estimated at $10^{10^{50}}$ [73]). Second, some degree of uncertainty is often present in these problems, which makes prediction of the future difficult.

In the face of these formidable challenges, humans are often remarkably good at solving many sequential decision-making problems. That humans are able to play chess, make sound business decisions, and navigate through life's sometimes daunting complexities is a testament to our critical-thinking and problem-solving skills. Moti-

vated by our own success at dealing with these challenges, it is natural to ask whether it is possible to build a machine that can perform as well or perhaps even better than a human in these complex situations. This difficult question, which is central to the field of Artificial Intelligence [133], offers the tantalizing possibilities of not only being able to create "intelligent" machines, but also of better understanding the ways that humans think and solve problems.

1.1 Motivation: Multi-agent Robotic Systems

Building machines that make good decisions in real-world, sequential decision-making problems is the topic of this thesis. The motivating application, which will be developed and used in much of the thesis work, is drawn from the rapidly evolving field of multi-agent robotics.

Robotic systems are becoming increasingly sophisticated in terms of hardware capabilities. Advances in sensor systems, onboard computational platforms, energy storage, and other enabling technologies have made it possible to build a huge variety of air-, ground-, and sea-based robotic vehicles for a range of different mission scenarios [6, 116]. Many of the mission scenarios of interest, such as persistent surveillance, search and rescue, weather pattern monitoring, etc., are inherently long-duration and require sustained coordination of multiple, cooperating robots in order to achieve the mission objectives. In these types of missions, a high level of autonomy is desired due to the logistical complexity and expense of direct human control of each individual vehicle. Currently, autonomous mission planning and control for multi-agent systems is an active area of research [69, 77, 92, 113, 119]. Some of the issues in this area are similar to questions arising in manufacturing systems [22, 74] and air transportation [7, 14, 25, 48, 72, 76, 131]. While these efforts have made significant progress in understanding how to handle some of the complexity inherent in multi-agent problems, there remain a number of open questions in this area.

One important question is referred to as the *health management problem for multi-agent systems* [98, 99]. Designs of current and future robotic platforms increasingly

incorporate a large number of sensors for monitoring the health of the robot's own subsystems. For example, sensors may be installed to measure the temperature and current of electric motors, the effectiveness of the robots' control actuators, or the fuel consumption rates in the engine. On a typical robot, sensors may provide a wealth of data about a large number of vehicle subsystems. By making appropriate use of this data, a health-aware autonomous system may be able to achieve a higher level of overall mission performance, as compared to a non-health-aware system, by making decisions that account for the current capabilities of each agent. For example, in a search and track mission, utilization of sensor health data may allow an autonomous system to assign the robots with the best-performing sensors to the search areas with the highest probability of finding the target.

Utilization of the current status of each robot is an important aspect of the health management problem. Another important aspect is the ability not only to react to the current status, but to consider the implications of future changes in health status or failures on the successful outcome of the mission. This predictive capability is of paramount importance, since it may allow an autonomous system to avoid an undesirable future outcome. For example, if a robot tracking a high value target is known to have a high probability of failure in the future, the autonomous system may be able to assign a backup robot to track the target, ensuring that the tracking can continue even if one of the robots fails.

Part of the work of this thesis addresses these health management issues and develops a general framework for thinking about the health management problem. It then specializes the discussion to the persistent surveillance problem, where uncertainties in fuel usage and sensor failures, coupled with communication constraints, create a complex and challenging planning problem. This work builds on previous health management techniques developed for the persistent surveillance problem [98]. While the previous work focused on embedding health-aware heuristics into an already-existing mission management algorithm, this thesis develops a new formulation of the problem in which health-aware behaviors emerge automatically.

1.2 Problem Formulation and Solution Approach

Dynamic programming is a natural mathematical framework for addressing the sequential decision-making problems that are the subject of this thesis [22]. First proposed by Bellman [18], the basic dynamic programming framework incorporates two main elements. The first of these elements is a discrete-time, dynamic model of the system in question. The dynamic model captures the evolution of the state of the system over time, under the influence of a given sequence of actions. Depending on the nature of the system, this model may incorporate either deterministic or stochastic state transitions. The second element is a cost function that depends on the system state and possibly the decision made in that state. In order to solve a dynamic program, we seek to find a state-dependent rule for choosing which action to take (a *policy*) that minimizes the expected, discounted sum of the costs incurred over time (see the discussion of Eq. (1.1) below).

If the state space associated with a particular dynamic programming problem is discrete, either because the state space is naturally described in a discrete setting or because it results from the discretization of an underlying continuous space, the resulting problem formulation is referred to as a Markov Decision Process (MDP). This thesis considers the general class of infinite horizon, discounted MDPs, where the goal is to minimize the so-called cost-to-go function $J_\mu : \mathcal{S} \to \mathbb{R}$ over the set of policies Π:

$$\min_{\mu \in \Pi} J_\mu(i_0) = \min_{\mu \in \Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(i_k, \mu(i_k))\right]. \tag{1.1}$$

Here, $i_0 \in \mathcal{S}$ is an initial state and the expectation $\mathbb{E}$ is taken over the possible future states $\{i_1, i_2, \dots\}$, given i_0 and the policy μ. In order to ensure that the sum in Eq. (1.1) is finite, the discount factor $0 < \alpha < 1$ is included, and we require in addition that the individual costs $g(i, u)$ are bounded by a constant for all values of i and u. Further terminology and notation for MDPs will be explained in detail in Chapter 2, which establishes necessary background material.

Approximate Methods

While MDPs are a powerful and general framework, there are two significant challenges, known as the "twin curses" of dynamic programming [19], that must be overcome in order to successfully apply them to real-world problems:

The curse of dimensionality refers to the fact that as the problem size, as measured by the dimensionality of the state space, increases, the number of possible states in the state space $|\mathcal{S}|$, and therefore the amount of computation necessary to solve (1.1) using standard techniques such as value iteration [22], grows exponentially rapidly. For typical problems, this exponential growth quickly overwhelms the available computational resources, making it difficult or impossible to solve the problem exactly in reasonable time.

The curse of modeling refers to the potential difficulty of accurately modeling the dynamics of the system in question. Many systems exhibit *parametric uncertainty*. In this scenario, the basic character of the governing dynamic equations is known, but the exact values of one or more parameters are not. For example, in a spacecraft control problem, the basic equations of motion (rigid-body dynamics) are well-understood, but the various moments of inertia may not be known exactly. In more extreme cases, even the basic form of the dynamic equations may not be fully understood. Inaccuracies in the model used to solve the MDP may lead to poor performance when the resulting control policy is implemented on the real system.

The question of how to address these challenges has been the subject of a great deal of research, leading to the development of fields known as *approximate dynamic programming*, *reinforcement learning*, and *neuro-dynamic programming* [24, 154]. Semantically, these fields are very closely related (having different names due mainly to the fact that they were originally developed by different communities), and for the sake of clarity, we shall generally refer to them all as approximate dynamic programming (abbreviated ADP).

A wide variety of methods for dealing with the twin curses of dynamic programming have been proposed in the ADP literature. Since many of these methods are significantly different from each other, it is helpful to understand a few general properties that allow us to categorize and compare different methods. Broadly speaking, there are four main properties of interest:

Choice of approximation architecture A central idea for addressing the curse of dimensionality is the use of a *function approximation architecture* to represent the (approximate) cost-to-go function $\widetilde{J}_\mu$. Functions that are representable within the chosen architecture are typically described by a set of parameters, where the number of parameters N_p is much smaller than the size of the state space $|\mathcal{S}|$. In this way, the problem of finding the cost-to-go function is reduced from a search in an $|\mathcal{S}|$-dimensional to one in an N_p-dimensional space, where $N_p \ll |\mathcal{S}|$. A large number of approximation architectures are possible and may involve, for example, neural networks, polynomials, radial basis functions, wavelets, kernel-based techniques, etc. The type of approximation architecture employed is a key property of any ADP method.

Model availability The amount of system model information assumed to be available is another key property of any ADP method. There are two main classes of methods with regard to model availability. *Model-free* methods attempt to address the curse of modeling by assuming no knowledge of the system model whatsoever. In order to learn the cost-to-go function, model-free methods rely on observed state transition data, obtained from either the real system or a simulation thereof. In contrast, *model-based* methods assume that some form of system model information is available and exploit this information in order to learn the cost-to-go function.

Training technique The question of how to select the best cost-to-go function from the chosen approximation architecture (for example, choosing the weights in a neural network approximation architecture) is called "training". Generally, training is formulated as an optimization problem. The type of optimization

problem (i.e. linear, quadratic, convex, non-convex, etc) and method of solution (i.e. gradient descent, analytic, linear programming, etc) are further important characteristics of an ADP method.

State observability The question of whether or not the full state of the system is known exactly leads to an important distinction. In particular, if the full state is not known exactly but must be inferred from noisy observations of the system in operation, then in general the problem must be modeled as a Partially Observable Markov Decision Process (POMDP). POMDPs are a generalization of the MDP framework to include an *observation model*, which defines a probability distribution over observations given the current state of the system. For further information about POMDPs, see [83, 87, 88, 108]. In this thesis, we shall assume that the state is fully observable, and will therefore use an MDP framework.

1.3 Literature Review

Some of the earliest investigations into ADP methods can be traced back to Shannon and Samuel in the 1950's. Shannon was perhaps the first to propose the approximation architecture now known as the *linear combination of basis functions* in his work with computer chess [144]. In this work, both important features of the chess game (such as material advantage, piece mobility, etc) and their corresponding weights were hand-selected and used to evaluate potential moves. Samuel, working with the game of checkers, went a step further and devised a automated training method for the computer to learn the weights [136]. Eventually, Samuel's checkers program was able to consistently beat its human creator.

The development of artificial neural networks [34, 75, 79] led to a great deal of interest in using them as approximation architectures in ADP problems. One of the most notable successes in this area was Tesauro's computer backgammon player, which was able to achieve world-class play [156, 157]. Tesauo's work utilized the method of temporal differences (TD) [24, 152–154] as the training mechanism. The

success of TD methods has since encouraged development of other methods such as Least Squares Policy Iteration (LSPI) [93], Least Squares Policy Evaluation (LSPE) [23, 114] and Least Squares Temporal Differences (LSTD) [39, 40, 93]; a summary of these methods is given in [22, Vol. 2, Chap. 6].

Another group of ADP methods arise from the observation that finding the optimal cost-to-go function can be cast as a linear programming problem [36, 58, 78, 104]. The resulting linear program suffers from the curse of dimensionality; it has as many decision variables as states in the MDP, and an even larger number of constraints. De Farias and Van Roy proposed an approach, known as approximate linear programming (ALP), that reduces the dimensionality of the linear program by utilizing a linear combination of basis functions architecture combined with a mechanism to sample only a small subset of the constraints [52–55]. An extension to the ALP approach that automatically generates the basis functions in the linear architecture was developed by Valenti [161].

A further class of ADP methods, known as *Bellman residual methods*, are motivated by attempting to find cost-to-go solutions $\widetilde{J}_\mu$ that minimize the error incurred in solving Bellman's equation. In particular, a natural criterion for evaluating the accuracy of a cost-to-go solution is the *Bellman error* BE:

$$BE \equiv ||\widetilde{J}_\mu - T_\mu \widetilde{J}_\mu|| = \left(\sum_{i \in \mathcal{S}} |\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)|^2 \right)^{1/2}. \tag{1.2}$$

The individual terms

$$\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)$$

which appear in the sum are referred to as the *Bellman residuals*, where T_μ is the fixed-policy Bellman operator (this operator will be defined precisely in Chapter 2). Designing an ADP algorithm that attempts to minimize the Bellman error over a set of candidate cost-to-go solutions is a sensible approach, since achieving an error of zero immediately implies that the exact solution has been found. However, it is difficult to carry out this minimization directly, since evaluation of Eq. (1.2) requires that the

Bellman residuals for every state in the state space be computed. To overcome this difficulty, a common approach is to generate a smaller set of representative sample states $\widetilde{\mathcal{S}}$ (using simulations of the system, prior knowledge about the important states in the system, or other means) and work with an *approximation* to the Bellman error $\widetilde{BE}$ obtained by summing Bellman residuals over only the sample states: [24, Ch. 6]:

$$\widetilde{BE} \equiv \left(\sum_{i \in \widetilde{\mathcal{S}}} |\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)|^2 \right)^{1/2}. \tag{1.3}$$

It is then practical to minimize $\widetilde{BE}$ over the set of candidate functions. This approach has been investigated by several authors, including [9, 13, 112, 143].

Recently, the use of kernel-based architectures for pattern classification and function approximation has generated excitement in the machine learning community. Building on results from statistical learning [50, 110, 162, 163] and the theory of reproducing kernels [10, 21, 71, 135, 139, 167], initial progress in this area focused on the development of kernel-based classifiers such as Support Vector Machines (SVMs) [20, 37, 41, 44, 140]. Much research has been done on efficient SVM training, even when the size of the problem is large [38, 89, 118, 121, 122, 171]. Similar kernel-based techniques can also be applied to the function approximation (regression) problem, yielding a family of regression algorithms such as Support Vector Regression [141, 142, 147], Gaussian Process Regression [43, 128, 148], and Kernel Ridge Regression [138]. Compared with function approximation architectures such as the neural networks used in much of the previous ADP work, kernel-based architectures enjoy a number of advantages, such as being trainable via convex (quadratic) optimization problems that do not suffer from local minima, and allowing the use of powerful, infinite-dimensional data representations.

Motivated by these recent advances, some research has been done to apply powerful kernel-based techniques to the ADP problem. Dietterich and Wang investigated a kernelized form of the linear programming approach to dynamic programming [59]. Ormoniet and Sen presented a model-free approach for doing approximate value it-

eration using kernel smoothers, under some restrictions on the structure of the state space [117]. Using Gaussian processes, Rasmussen and Kuss derived analytic expressions for the approximate cost-to-go of a fixed policy, in the case where the system state variables are restricted to evolve independently according to Gaussian probability transition functions [127]. Engel, Mannor, and Meir applied a Gaussian process approximation architecture to TD learning [65–67], and Reisinger, Stone, and Miikkulainen subsequently adapted this framework to allow the kernel parameters to be estimated online [130]. Tobias and Daniel proposed a LSTD approach based on SVMs [158]. Several researchers have investigated designing specialized kernels that exploit manifold structure in the state space [16, 17, 102, 103, 146, 150, 151].

1.4 Summary of Contributions

A main focus of this thesis is on the development of a new class of kernel-based ADP algorithms that are similar in spirit to traditional Bellman residual methods. Similar to traditional methods, the new algorithms are designed to minimize an approximate form of the Bellman error as given in Eq. (1.3). The motivation behind this work is the observation that, given the approximate Bellman error $\widetilde{BE}$ as the objective function to be minimized, we should seek to find a solution for which the objective is identically zero, the smallest possible value. The ability to find such a solution depends on the richness of the function approximation architecture employed, which in turn defines the set of candidate solutions. Traditional, parametric approximation architectures such as neural networks and linear combinations of basis functions are finite-dimensional, and therefore it may not always be possible to find a solution satisfying $\widetilde{BE} = 0$ (indeed, if a poor network topology or set of basis functions is chosen, the minimum achievable error may be large). In contrast, in this thesis we shall show that by exploiting the richness and flexibility of kernel-based approximation architectures, it is possible to construct algorithms that always produce a solution for which $\widetilde{BE} = 0$. As an immediate consequence, our algorithms have the desirable property of reducing to *exact* policy iteration in the limit of sampling the entire state

space, since in this limit, the Bellman residuals are zero everywhere, and therefore the obtained cost-to-go function is exact ($\widetilde{J}_\mu = J_\mu$). Furthermore, by exploiting knowledge of the system model (we assume this model is available), the algorithms eliminate the need to perform trajectory simulations and therefore do not suffer from simulation noise effects. We refer to our approach as *Bellman residual elimination* (BRE), rather than Bellman residual minimization, to emphasize the fact that the error is explicitly forced to zero.

With regard to the development of these new ADP algorithms, the thesis makes the following contributions:

- The basic, model-based BRE approach is developed, and its theoretical properties are studied. In particular, we show how the problem of finding a cost-to-go solution, for which the Bellman residuals are identically zero at the sample states, can be cast as a regression problem in an appropriate Reproducing Kernel Hilbert Space (RKHS). We then explain how any kernel-based regression technique can be used to solve this problem, leading to a family of BRE algorithms. We prove that these algorithms converge to the optimal policy in the limit of sampling the entire state space. Furthermore, we show that the BRE algorithm based on Gaussian process regression can provide error bounds on the cost-to-go solution and can automatically learn free parameters in the kernel.

- A multi-stage extension of the basic BRE approach is developed. In this extension, Bellman residuals of the form $|\widetilde{J}_\mu(i) - T_\mu^n \widetilde{J}_\mu(i)|$, where $n \geq 1$ is an integer, are eliminated at the sample states. We show how, as a result of this extension, a kernel function arises that automatically captures local structure in the state space. In effect, this allows the multi-stage BRE algorithms to automatically discover and use a kernel that is tailored to the problem at hand. Similar to the basic BRE approach, we prove convergence of multi-stage BRE to the optimal policy in the limit of sampling the entire state space.

- A model-free variant of BRE is developed. We show how the general, multi-stage BRE algorithms can be carried out when a system model is unavailable,

by using simulated or actual state trajectories to approximate the data computed by the BRE algorithms. Again, we prove convergence results for these model-free algorithms. Furthermore, we present results comparing the performance of model-based and model-free BRE against model-based and model-free LSPI [93], and show that both variants of BRE yield more consistent, higher performing policies than LSPI in a benchmark problem.

A second focus area of the thesis deals with online adaptation for MDPs. In many applications, the basic form of the system model is known beforehand, but there may be a number of parameters of the model whose values are unknown or only known poorly before the system begins operating. As a result, when the MDP is solved offline, this poor knowledge of the model may lead to suboptimal performance when the control policy is implemented on the real system. Furthermore, in some circumstances, these parameters may be time-varying, which presents further difficulties if the MDP can only be solved offline. To address these issues, an online adaptation architecture is developed that combines a generic MDP solver with a model parameter estimator. In particular, this adaptive architecture:

- Allows the unknown model parameters to be estimated online; using these parameter estimates, a policy for the corresponding MDP model is then recomputed in real-time. As a result, the adaptive architecture allows the system to continuously re-tune its control policy to account for better model information obtained through observations of the actual system in operation, and react to changes in the model as they occur.

- Permits any MDP solution technique to be utilized to recompute the policy online. In particular, the architecture allows BRE-based algorithms to be used to compute approximate policies for large MDPs quickly.

- Is validated through hardware flight tests carried out in the MIT RAVEN indoor flight facility [80]. These flight tests demonstrate the successful ability of the adaptive architecture to quickly adjust the policy as new model information arrives, resulting in improved overall mission performance.

A third focus area deals with autonomous planning in multi-agent robotic systems. In particular, we investigate the *persistent surveillance* problem, in which one or more unmanned aerial vehicles (UAVs) and/or unmanned ground vehicles (UGVs) must be maintained at a designated location on a continuous basis. This problem is pertinent to a number of applications, including search and rescue, natural disaster relief operations, urban traffic monitoring, etc. These missions are challenging in part because their long duration increases the likelihood that one or more agents will experience failures over the course of the mission. The planning system must provide a high level of mission performance even in the event of failures. In order to investigate these issues, this thesis:

- Formulates the basic persistent surveillance problem as an MDP, which captures the requirement of scheduling assets to periodically move back and forth between the surveillance location and the base location for refueling and maintenance. In addition, we incorporate a number of randomly-occurring failure scenarios (such as sensor failures, unexpected fuel usage due to adverse weather conditions, etc) and constraints (such as the requirement to maintain a communications link between the base and agents in the surveillance area) into the problem formulation. We show that the optimal policy for the persistent surveillance problem formulation not only properly manages asset scheduling, but also *anticipates* the adverse effects of failures on the mission and takes actions to mitigate their impact on mission performance.

- Highlights the difficulties encountered in solving large instances of the persistent surveillance problem using exact methods, and demonstrates that BRE can be used to quickly compute near-optimal approximate policies for these large problems.

- Presents a fully autonomous UxV mission architecture which incorporates the persistent surveillance planner into a larger framework that handles other necessary aspects including low-level vehicle control; path planning; task generation and assignment; and online policy adaptation using the adaptive MDP archi-

tecture.

- Presents flight experiments, conducted in the MIT RAVEN indoor flight facility [80], that demonstrate the successful application of our problem formulation, solution technique, and mission architecture to controlling a heterogeneous team of UxVs in a number of complex and realistic mission scenarios. Furthermore, we demonstrate performance benefits of our approach over a deterministic planning approach that does not account for randomness in the system dynamics model.

1.5 Thesis Outline

The organization of the thesis is as follows: Chapter 2 presents background material on Markov Decision Processes, kernel-based regression, and reproducing kernel Hilbert spaces, which will be used in later chapters. Chapter 3 presents the main ideas behind the Bellman residual elimination approach, while Chapter 4 develops the multi-stage and model-free extensions. Chapter 5 discusses the general problem of health management in multi-agent robotic systems, develops the persistent surveillance problem formulation and analyzes its properties. Chapter 6 presents the adaptive MDP architecture. Chapter 7 presents a set of simulation and flight experiments demonstrating the effectiveness of the BRE solution approach and the adaptive MDP architecture for controlling teams of autonomous UxVs. Finally, Chapter 8 offers concluding remarks and highlights areas for future research.

Chapter 2

Background

This chapter provides a background in the mathematical concepts that this thesis builds upon.

2.1 Markov Decision Processes

An infinite horizon, discounted, finite state MDP is specified by $(\mathcal{S}, \mathcal{A}, P, g, \alpha)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space (assumed to be finite), P is the system dynamics model where $P_{ij}(u)$ gives the transition probability from state i to state j under action u, and $g(i, u)$ gives the immediate cost of taking action u in state i. Future costs are discounted by a factor $0 < \alpha < 1$. A policy of the MDP is denoted by $\mu : \mathcal{S} \to \mathcal{A}$. Given the MDP specification, the problem is to minimize the so-called cost-to-go function $J_\mu : \mathcal{S} \to \mathbb{R}$ over the set of admissible policies Π:

$$\min_{\mu \in \Pi} J_\mu(i_0) = \min_{\mu \in \Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(i_k, \mu(i_k))\right]. \tag{2.1}$$

Here, $i_0 \in \mathcal{S}$ is an initial state and the expectation $\mathbb{E}$ is taken over the possible future states $\{i_1, i_2, \dots\}$, given i_0 and the policy μ.

In solving the MDP, the primary goal is to find a policy $\mu^\star$ which achieves the minimum in (2.1) (note that this policy need not necessarily be unique). The optimal

cost associated with $\mu^\star$, denoted by $J^\star \equiv J_{\mu^\star}$, satisfies the Bellman equation [22]

$$J^\star(i) = \min_{u \in \mathcal{A}} \left(g(i,u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J^\star(j) \right) \quad \forall i \in \mathcal{S}. \tag{2.2}$$

It is customary to define the *dynamic programming operator* T as

$$(TJ)(i) \equiv \min_{u \in \mathcal{A}} \left(g(i,u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J(j) \right),$$

so that Eq. (2.2) can be written compactly as

$$J^\star = TJ^\star.$$

If $J^\star$ can be found by solving Eq. (2.2), then the optimal policy $\mu^\star$ is given by

$$\mu^\star(i) = \arg\min_{u \in \mathcal{A}} \left(g(i,u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J^\star(j) \right). \tag{2.3}$$

Eq. (2.3) establishes a relationship between the policy and its associated cost function. We assume that the minimization in Eq. (2.3) can be performed exactly since $\mathcal{A}$ is a finite set. Therefore, the bulk of the work in solving an MDP involves computing the optimal cost function by solving the nonlinear system of equations given in (2.2).

One approach to computing the optimal cost is to solve Eq. (2.2) in an iterative fashion using value iteration. An alternative approach arises from the observation that if a policy μ is fixed, then the nonlinear system Eq. (2.2) reduces to a linear system which is easier to solve:

$$J_\mu(i) = g(i, \mu(i)) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(\mu(i)) J_\mu(j) \quad \forall i \in \mathcal{S}. \tag{2.4}$$

For notational convenience, the fixed-policy cost and state transition functions are

defined as

$$g_i^\mu \equiv g(i, \mu(i)) \tag{2.5}$$

$$P_{ij}^\mu \equiv P_{ij}(\mu(i)). \tag{2.6}$$

In vector-matrix notation, Eq. (2.4) can also be expressed as

$$\underline{J}_\mu = (I - \alpha P^\mu)^{-1} \underline{g}^\mu, \tag{2.7}$$

where $\underline{g}^\mu$ is the vector of immediate costs $g(i, \mu(i))$ over the state space $\mathcal{S}$, and similarly, $\underline{J}_\mu$ is the vector of cost-to-go values over $\mathcal{S}$.

We define the *fixed-policy dynamic programming operator* T_μ as

$$(T_\mu J)(i) \equiv g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu J(j), \tag{2.8}$$

so that Eq. (2.4) can be written compactly as

$$J_\mu = T_\mu J_\mu.$$

Solving Eq. (2.4) is known as *policy evaluation*. The solution J_μ is the cost-to-go of the fixed policy μ. Once the policy's cost-to-go function J_μ is known, a new, better policy μ' can be constructed by performing a *policy improvement* step:

$$\mu'(i) = \arg\min_{u \in \mathcal{A}} \left(g(i, u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J_\mu(j) \right).$$

By iteratively performing policy evaluation followed by policy improvement, a sequence of policies that are guaranteed to converge to the optimal policy $\mu^\star$ is obtained [22]. This procedure is known as *policy iteration*.

2.2 Support Vector Regression

This section provides a brief overview of support vector regression (SVR); for more details, see [147]. The objective of the SVR problem is to learn a function $f(x)$ of the form

$$f(x) = \sum_{l=1}^{r} \theta_l \phi_l(x) = \langle \underline{\Theta}, \underline{\Phi}(x) \rangle \tag{2.9}$$

that gives a good approximation to a given set of training data

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

where $x_i \in \mathbb{R}^m$ is the input data and $y_i \in \mathbb{R}$ is the observed output. The vector

$$\underline{\Phi}(x) = (\phi_1(x) \ \ldots \ \phi_r(x))^T$$

is referred to as the *feature vector* of the point x, where each *feature* (also called a *basis function*) $\phi_i(x)$ is a scalar-valued function of x. The vector

$$\underline{\Theta} = (\theta_1 \ \ldots \ \theta_r)^T$$

is referred to as the *weight vector*. The notation $\langle \cdot, \cdot \rangle$ is used to denote the standard inner product.

The training problem is posed as the following quadratic optimization problem:

$$\min_{\underline{\Theta}, \underline{\xi}, \underline{\xi}^\star} \quad \frac{1}{2}||\underline{\Theta}||^2 + c\sum_{i=1}^{n}(\xi_i + \xi_i^\star) \tag{2.10}$$

$$\text{subj. to} \quad y_i - \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle \leq \epsilon + \xi_i \tag{2.11}$$

$$-y_i + \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle \leq \epsilon + \xi_i^\star \tag{2.12}$$

$$\xi_i, \xi_i^\star \geq 0 \qquad \forall i \in \{1, \ldots, n\}. \tag{2.13}$$

Here, the regularization term $\frac{1}{2}||\underline{\Theta}||^2$ penalizes model complexity, and the $\xi_i, \xi_i^\star$ are slack variables which are active whenever a training point y_i lies farther than a dis-

tance ϵ from the approximating function $f(x_i)$, giving rise to the so-called ϵ-*insensitive loss function*. The parameter c trades off model complexity with accuracy of fitting the observed training data. As c increases, any data points for which the slack variables are active incur higher cost, so the optimization problem tends to fit the data more closely (note that fitting too closely may not be desired if the training data is noisy).

The minimization problem [Eqs. (2.10)-(2.13)] is difficult to solve when the number of features r is large, for two reasons. First, it is computationally demanding to compute the values of all r features for each of the data points. Second, the number of decision variables in the problem is $r + 2n$ (since there is one weight element θ_i for each basis function $\phi_i(\cdot)$ and two slack variables $\xi_i, \xi_i^\star$ for each training point), so the minimization must be carried out in an $(r + 2n)$-dimensional space. To address these issues, one can solve the primal problem through its dual, which can be formulated by computing the Lagrangian and minimizing with respect to the primal variables $\underline{\Theta}$ and $\underline{\xi}, \underline{\xi}_i^\star$ (again, for more details, see [147]). The dual problem is

$$\begin{aligned} \max_{\underline{\lambda}, \underline{\lambda}^\star} \quad & -\tfrac{1}{2} \textstyle\sum_{i,i'=1}^{n} (\lambda_i^\star - \lambda_i)(\lambda_{i'}^\star - \lambda_{i'}) \langle \underline{\Phi}(x_i), \underline{\Phi}(x_{i'}) \rangle \\ & -\epsilon \textstyle\sum_{i=1}^{n} (\lambda_i^\star + \lambda_i) + \sum_{i=1}^{n} y_i (\lambda_i^\star - \lambda_i) \qquad (2.14) \\ \text{subj. to} \quad & 0 \le \lambda_i, \lambda_i^\star \le c \quad \forall i \in \{1, \ldots, n\}. \qquad (2.15) \end{aligned}$$

Note that the feature vectors $\underline{\Phi}(x_i)$ now enter into the optimization problem only as inner products. This is important, because it allows a *kernel function*,

$$k(x_i, x_{i'}) \equiv \langle \underline{\Phi}(x_i), \underline{\Phi}(x_{i'}) \rangle,$$

to be defined whose evaluation may avoid the need to explicitly calculate the vectors $\underline{\Phi}(x_i)$, resulting in significant computational savings. Also, the dimensionality of the dual problem is reduced to only $2n$ decision variables, since there is one λ_i and one $\lambda_i^\star$ for each of the training points. When the number of features is large, this again results in significant computational savings. Furthermore, it is well known

that the dual problem can be solved efficiently using special-purpose techniques such as Sequential Minimal Optimization (SMO) [89, 122]. Once the dual variables are known, the weight vector is given by

$$\underline{\Theta} = \sum_{i=1}^{n} (\lambda_i - \lambda_i^\star) \underline{\Phi}(x_i), \tag{2.16}$$

and the function $f(x)$ can be computed using the so-called *support vector expansion*:

$$\begin{aligned} f(x) &= \langle \underline{\Theta}, \underline{\Phi}(x) \rangle \\ &= \sum_{i=1}^{n} (\lambda_i - \lambda_i^\star) \langle \underline{\Phi}(x_i), \underline{\Phi}(x) \rangle \\ &= \sum_{i=1}^{n} (\lambda_i - \lambda_i^\star) k(x_i, x). \end{aligned} \tag{2.17}$$

2.3 Gaussian Process Regression

In this section, another kernel-based regression technique, Gaussian process regression [128], is reviewed. Gaussian process regression attempts to solve the same problem as the support vector regression technique discussed in the previous section: given a set of training data $\mathcal{D}$, find a function $f(x)$ that provides a good approximation to the data. Gaussian process regression approaches this problem by defining a probability distribution over a set of admissible functions and performing Bayesian inference over this set. A Gaussian process is defined as a (possible infinite) collection of random variables, any finite set of which is described by a joint Gaussian distribution. The process is therefore completely specified by a mean function

$$m(x) = \mathbb{E}[f(x)]$$

and positive semidefinite covariance (kernel) function

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))].$$

The Gaussian process is denoted by

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

For the purposes of regression, the random variables of a Gaussian process given by $\mathcal{GP}(m(x), k(x, x'))$ are interpreted as the function values $f(x)$ at particular values of the input x. Note the important fact that given any finite set of input points $\mathcal{X} = \{x_1, \ldots, x_n\}$, the distribution over the corresponding output variables $\{y_1, \ldots, y_n\}$ is given by a standard Gaussian distribution

$$(y_1, \ldots, y_n)^T \sim \mathcal{N}(\underline{\mu}, \Sigma),$$

where the mean $\underline{\mu}$ and covariance Σ of the distribution are obtained by "sampling" the mean $m(x)$ and covariance $k(x, x')$ functions of the Gaussian process at the points $\mathcal{X}$:

$$\begin{aligned} \underline{\mu} &= (m(x_1), \ldots, m(x_n))^T \\ \Sigma &= \mathbb{K}(\mathcal{X}, \mathcal{X}) = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \end{aligned}$$

Here, $\mathbb{K}(\mathcal{X}, \mathcal{X})$ denotes the $n \times n$ Gram matrix of the kernel $k(x, x')$ evaluated for the points $\mathcal{X}$.

The Gaussian process $\mathcal{GP}(m(x), k(x, x'))$ represents a prior distribution over functions. To perform regression, the training data $\mathcal{D}$ must be incorporated into the Gaussian process to form a posterior distribution, such that every function in the support of the posterior agrees with the observed data. From a probabilistic standpoint, this amounts to conditioning the prior on the data. Fortunately, since the prior is a Gaussian distribution, the conditioning operation can be computed analytically. To be precise, assume that we wish to know the value of the function f at a set of points $\mathcal{X}_\star = \{x_1^\star, \ldots, x_l^\star\}$, conditioned on the training data. Denote the vector $(y_1, \ldots, y_n)^T$

by $\underline{y}$, $(f(x_1), \ldots, f(x_n))^T$ by $\underline{f}(\mathcal{X})$, and $(f(x_1^\star), \ldots, f(x_l^\star))^T$ by $\underline{f}(\mathcal{X}_\star)$. Now, using the definition of the Gaussian process, the joint prior over the outputs is

$$\begin{pmatrix} \underline{f}(\mathcal{X}) \\ \underline{f}(\mathcal{X}_\star) \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} \mathbb{K}(\mathcal{X}, \mathcal{X}) & \mathbb{K}(\mathcal{X}, \mathcal{X}_\star) \\ \mathbb{K}(\mathcal{X}_\star, \mathcal{X}) & \mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star) \end{pmatrix}\right),$$

where $\mathbb{K}(\mathcal{X}, \mathcal{X}_\star)$ denotes the $n \times l$ Gram matrix of covariances between all pairs of training and test points (the other matrices $\mathbb{K}(\mathcal{X}, \mathcal{X})$, $\mathbb{K}(\mathcal{X}_\star, \mathcal{X})$, and $\mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star)$ are defined similarly).

Conditioning this joint prior on the data yields [128, Sec. A.2]

$$\underline{f}(\mathcal{X}_\star) \mid \left(\mathcal{X}_\star, \mathcal{X}, \underline{f}(\mathcal{X}) = \underline{y}\right) \;\sim\; \mathcal{N}(\underline{\mu}_{posterior}, \Sigma_{posterior}), \tag{2.18}$$

where

$$\begin{aligned} \underline{\mu}_{posterior} &= \mathbb{K}(\mathcal{X}_\star, \mathcal{X})\mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\underline{y}, \\ \Sigma_{posterior} &= \mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star) - \mathbb{K}(\mathcal{X}_\star, \mathcal{X})\mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\mathbb{K}(\mathcal{X}, \mathcal{X}_\star). \end{aligned}$$

Eq. (2.18) is a general result that predicts the mean and covariance of the function values at all of the points $\mathcal{X}_\star$. If we wish to query the function at only a single point $x_\star$, Eq. (2.18) can be simplified. Note that if $|\mathcal{X}_\star| = 1$, then the matrices $\mathbb{K}(\mathcal{X}, \mathcal{X}_\star)$ and $\mathbb{K}(\mathcal{X}_\star, \mathcal{X})$ reduce to a column vector and a row vector, which are denoted by $\underline{k}_\star$ and $\underline{k}_\star^T$, respectively. Similarly, the matrix $\mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star)$ reduces to the scalar $k(x_\star, x_\star)$. With this notation, the mean $\bar{f}(x_\star)$ and variance $\mathbb{V}[f(x_\star)]$ of the function value at $x_\star$ can be expressed as

$$\bar{f}(x_\star) = \underline{k}_\star^T \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\underline{y} \tag{2.19}$$

$$\mathbb{V}[f(x_\star)] = k(x_\star, x_\star) - \underline{k}_\star^T \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\underline{k}_\star. \tag{2.20}$$

Defining the vector $\underline{\lambda}$ as

$$\underline{\lambda} = \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\underline{y},$$

Eq. (2.19) also can be written as

$$\bar{f}(x_\star) = \sum_{i=1}^{n} \lambda_i k(x_i, x_\star). \tag{2.21}$$

Marginal Likelihood and Kernel Parameter Selection

In many cases, the kernel function $k(i, i')$ depends on a set of parameters $\underline{\Omega}$. The notation $k(i, i'; \underline{\Omega})$ and $\mathbb{K}(\mathcal{X}, \mathcal{S}; \underline{\Omega})$ shall be used when we wish to explicitly emphasize dependence of the kernel and its associated Gram matrix on the parameters. Each choice of $\underline{\Omega}$ defines a different model of the data, and not all models perform equally well at explaining the observed data. In Gaussian process regression, there is a simple way to quantify the performance of a given model. This quantity is called the *log marginal likelihood* and is interpreted as the probability of observing the data, given the model. The log marginal likelihood is given by [128, Sec. 5.4]

$$\log p(\underline{y}|\mathcal{X}, \underline{\Omega}) = -\frac{1}{2}\underline{y}^T \mathbb{K}(\mathcal{X}, \mathcal{X}; \underline{\Omega})^{-1}\underline{y} - \frac{1}{2}\log|\mathbb{K}(\mathcal{X}, \mathcal{X}; \underline{\Omega})| - \frac{n}{2}\log 2\pi. \tag{2.22}$$

The best choice of the kernel parameters $\underline{\Omega}$ are those which give the highest probability of the data; or equivalently, those which maximize the log marginal likelihood [Eq. (2.22)]. Note that maximizing the log marginal likelihood is equivalent to performing Bayesian inference given a uniform prior over $\underline{\Omega}$. The derivative of the likelihood with respect to the individual parameters Ω_j can be calculated analytically [128, 5.4]:

$$\frac{\partial \log p(\underline{y}|\mathcal{X}, \underline{\Omega})}{\partial \Omega_j} = \frac{1}{2}\mathrm{tr}\left((\underline{\lambda}\underline{\lambda}^T - \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1})\frac{\partial \mathbb{K}(\mathcal{X}, \mathcal{X})}{\partial \Omega_j}\right). \tag{2.23}$$

Eq. (2.23) allows the use of any gradient-based optimization method to select the optimal values for the parameters $\underline{\Omega}$.

2.4 Reproducing Kernel Hilbert Spaces

This section provides a brief overview of kernel functions and the associated theory of Reproducing Kernel Hilbert Spaces. The presentation of some of this material follows the more detailed discussion in [141, Sec. 2.2].

The kernel function $k(\cdot, \cdot)$ plays an important role in any kernel-based learning method. The kernel maps any two elements from a space of input patterns $\mathcal{X}$ to the real numbers, $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and can be thought of as a similarity measure on the input space. We shall take $\mathcal{X}$ to be the state space $\mathcal{S}$ of the MDP. In the derivation of kernel methods such as support vector regression, the kernel function arises naturally as an inner (dot) product in a high-dimensional feature space. As such, the kernel must satisfy several important properties of an inner product: it must be *symmetric* ($k(x, y) = k(y, x)$) and *positive semi-definite*. The latter property is defined as positive semi-definiteness of the associated Gram matrix $\mathbb{K}$, where

$$\mathbb{K}_{ij} \equiv k(x_i, x_j),$$

for all subsets $\{x_1, \ldots, x_m\} \subset \mathcal{X}$. A kernel that satisfies these properties is said to be *admissible*. In the thesis, we shall deal only with admissible kernels. Furthermore, if the associated Gram matrix $\mathbb{K}$ is strictly positive definite for all subsets $\{x_1, \ldots, x_m\} \subset \mathcal{X}$, the kernel is called *nondegenerate*. The use of nondegenerate kernels will play an important role in establishing many of the theoretical results in the thesis.

Given a mapping from inputs to the feature space, the corresponding kernel function can be constructed. However, in many cases, it is desirable to avoid explicitly defining the feature mapping and instead specify the kernel function directly. Therefore, it is useful to consider a construction that proceeds in the opposite direction. That is: *given a kernel, we seek to construct a feature mapping such that the kernel can be expressed as an inner product in the corresponding feature space.*

To begin, assume that $\mathcal{X}$ is an arbitrary set of input data. (In later sections, the set $\mathcal{X}$ will be taken as $\mathcal{S}$, the set of states in the MDP). Furthermore, assume

that $k(\cdot,\cdot)$ is a symmetric, positive definite kernel which maps two elements in $\mathcal{X}$ to the reals: $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Now, define a mapping $\underline{\Phi}$ from $\mathcal{X}$ to the space of real-valued functions over $\mathcal{X}$ as follows:

$$\begin{aligned} \underline{\Phi} : \qquad & \mathcal{X} \to \mathbb{R}^{\mathcal{X}} \\ & \underline{\Phi}(x)(\cdot) = k(\cdot, x). \end{aligned} \tag{2.24}$$

Using the set of functions $\{\underline{\Phi}(x)(\cdot) \mid x \in \mathcal{X}\}$ as a basis, a real vector space $\mathcal{H}$ can be constructed by taking linear combinations of the form

$$f(\cdot) = \sum_{i=1}^{m} \lambda_i \underline{\Phi}(x_i)(\cdot) = \sum_{i=1}^{m} \lambda_i k(\cdot, x_i), \tag{2.25}$$

where $m \in \mathbb{N}$, $\lambda_1 \ldots \lambda_m \in \mathbb{R}$, and $x_1, \ldots, x_m \in \mathcal{X}$ are arbitrary. The vector space $\mathcal{H}$ is then given by the set of all such functions $f(\cdot)$:

$$\mathcal{H} = \{f(\cdot) \mid m \in \mathbb{N}, \lambda_1 \ldots \lambda_m \in \mathbb{R}, x_1, \ldots, x_m \in \mathcal{X}\}$$

Furthermore, if

$$g(\cdot) = \sum_{j=1}^{m'} \beta_j k(\cdot, x'_j)$$

is another function in the vector space, an inner product $\langle f(\cdot), g(\cdot) \rangle$ can be defined as

$$\langle f(\cdot), g(\cdot) \rangle = \sum_{i=1}^{m} \sum_{j=1}^{m'} \lambda_i \beta_j k(x_i, x'_j). \tag{2.26}$$

It is straightforward to show that $\langle \cdot, \cdot \rangle$ satisfies the necessary properties of an inner product [141, 2.2.2].

The inner product as defined in Eq. (2.26) has the following important property, which follows immediately from the definition:

$$\langle f(\cdot), k(\cdot, x) \rangle = \sum_{i=1}^{m} \lambda_i k(x, x_i) = f(x). \tag{2.27}$$

In particular, letting $f(\cdot) = k(\cdot, x')$:

$$\langle k(\cdot, x), k(\cdot, x') \rangle = k(x, x').$$

Substituting Eq. (2.24),

$$\langle \underline{\Phi}(x)(\cdot), \underline{\Phi}(x')(\cdot) \rangle = k(x, x'). \tag{2.28}$$

$k(\cdot, \cdot)$ is therefore said to have the *reproducing property* in $\mathcal{H}$, and the mapping $\underline{\Phi}$ is called the *reproducing kernel map*. Notice that the main objective has now been accomplished: starting from the kernel function $k(\cdot, \cdot)$, a feature mapping $\underline{\Phi}(x)$ [Eq. (2.24)] has been constructed such that the kernel is expressible as an inner product in the feature space $\mathcal{H}$ [Eq. (2.28)].

An important equivalent characterization of nondegenerate kernels states that *a kernel is nondegenerate if and only if its associated features* $\{\underline{\Phi}(x_1)(\cdot), \ldots, \underline{\Phi}(x_m)(\cdot)\}$ *are linearly independent for all subsets* $\{x_1, \ldots, x_m\} \subset \mathcal{X}$. This property will be important in a number of proofs later in the thesis.

RKHS Definition

The feature space $\mathcal{H}$ constructed in the previous section is an inner product (or *pre-Hilbert*) space. $\mathcal{H}$ can be turned into a real Hilbert space by endowing it with the norm $||.||$ associated with the inner product $\langle \cdot, \cdot \rangle$: $||f|| = \sqrt{\langle f, f \rangle}$. In light of the reproducing property of the kernel in $\mathcal{H}$, the resulting space is called a *Reproducing Kernel Hilbert Space*. A formal definition of an RKHS is as follows:

Definition: Reproducing Kernel Hilbert Space. Let $\mathcal{X}$ be an input set, and $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a Reproducing Kernel Hilbert Space endowed with an inner product $\langle \cdot, \cdot \rangle$ and norm $|| \cdot ||_{\mathcal{H}}$ if there exists a kernel function $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with the following properties:

1. k has the *reproducing property* [Eq. (2.27)].

2. The set of functions $\{k(\cdot, x) \mid x \in \mathcal{X}\}$ spans $\mathcal{H}$.

It can be shown [141, 2.2.3] that the RKHS uniquely defines k. The preceding section shows that k also uniquely determines the RKHS (by construction), so there is a one-to-one relationship between the kernel and its corresponding RKHS. As such, we shall sometimes write $\mathcal{H}_k$ to explicitly denote the RKHS corresponding to the kernel k. Notice that every element $f(\cdot) \in \mathcal{H}$, being a linear combination of functions $k(\cdot, x_i)$ [Eq. (2.25)], can be represented by its expansion coefficients $\{\lambda_i \mid i = 1, \ldots, m\}$ and input elements $\{x_i \mid i = 1, \ldots, m\}$. This representation will be important in our development of the BRE algorithms.

Chapter 3

Model-Based Bellman Residual Elimination

Markov Decision Processes (MDPs) are a powerful framework for addressing problems involving sequential decision making under uncertainty [22, 124]. Such problems arise frequently in a number of fields, including engineering, finance, and operations research. It is well-known that MDPs suffer from the curse of dimensionality, which states that the size of the state space (and therefore the amount of time necessary to compute the optimal policy using standard techniques such as value iteration [22]), increases exponentially rapidly with the dimensionality of the problem. The curse of dimensionality renders most MDPs of practical interest very difficult to solve exactly using standard methods such as value iteration or policy iteration. To overcome this challenge, a wide variety of methods for generating approximate solutions to large MDPs have been developed, giving rise to the field of *approximate dynamic programming* [24, 154].

Approximate policy iteration is a central idea in many approximate dynamic programming methods. In this approach, an approximation to the cost-to-go vector of a fixed policy is computed; this step is known as *policy evaluation*. Once this approximate cost-to-go is known, a *policy improvement* step computes a new, potentially improved policy, and the process is repeated. In many problems, the policy improvement step involves a straightforward minimization over a finite set of possible

actions, and therefore can be performed exactly. However, the policy evaluation step is generally more difficult, since it involves solving the fixed-policy Bellman equation:

$$T_\mu J_\mu = J_\mu. \tag{3.1}$$

Here, J_μ represents the cost-to-go vector of the policy μ, and T_μ is the fixed-policy dynamic programming operator (these objects will be fully explained in the next section). Eq. (3.1) is a linear system of dimension $|\mathcal{S}|$, where $|\mathcal{S}|$ denotes the size of the state space $\mathcal{S}$. Because $|\mathcal{S}|$ is typically very large, solving Eq. (3.1) exactly is impractical, and an alternative approach must be taken to generate an approximate solution. Much of the research done in approximate dynamic programming focuses on how to generate these approximate solutions, which will be denoted in this chapter by $\widetilde{J}_\mu$.

The accuracy of an approximate solution $\widetilde{J}_\mu$ generated by an ADP algorithm is important to the ultimate performance achieved by the algorithm. A natural criterion for evaluating solution accuracy in this context is the *Bellman error* BE:

$$BE \equiv ||\widetilde{J}_\mu - T_\mu \widetilde{J}_\mu|| = \left(\sum_{i \in \mathcal{S}} |\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)|^2 \right)^{1/2}. \tag{3.2}$$

The individual terms

$$\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)$$

which appear in the sum are referred to as the *Bellman residuals*. Designing an ADP algorithm that attempts to minimize the Bellman error over a set of candidate cost-to-go solutions is a sensible approach, since achieving an error of zero immediately implies that the exact solution has been found. However, it is difficult to carry out this minimization directly, since evaluation of Eq. (3.2) requires that the Bellman residuals for every state in the state space be computed. To overcome this difficulty, a common approach is to generate a smaller set of representative sample states $\widetilde{\mathcal{S}}$ (using simulations of the system, prior knowledge about the important states in the system, or other means) and work with an approximation to the Bellman error $\widetilde{BE}$

obtained by summing Bellman residuals over only the sample states: [24, Ch. 6]:

$$\widetilde{BE} \equiv \left(\sum_{i \in \widetilde{\mathcal{S}}} |\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)|^2 \right)^{1/2}. \tag{3.3}$$

It is then practical to minimize $\widetilde{BE}$ over the set of candidate functions. This approach has been investigated by several authors, including [9, 13, 112, 143], resulting in a class of approximate dynamic programming algorithms known as *Bellman residual methods*.

The set of candidate functions is usually referred to as a *function approximation architecture*, and the choice of architecture is an important issue in the design of any approximate dynamic programming algorithm. Numerous approximation architectures, such as neural networks [34, 75, 79, 156, 157], linear architectures [22–24, 54, 93, 145, 161], splines [160], and wavelets [101, 103] have been investigated for use in approximate dynamic programming. Recently, motivated by the success of kernel-based methods such as support vector machines [41, 47, 147] and Gaussian processes [43, 128] for pattern classification and regression, researchers have begun applying these powerful techniques in the approximate dynamic programming domain. Dietterich and Wang investigated a kernelized form of the linear programming approach to dynamic programming [59]. Ormoniet and Sen presented a model-free approach for doing approximate value iteration using kernel smoothers, under some restrictions on the structure of the state space [117]. Using Gaussian processes, Rasmussen and Kuss derived analytic expressions for the approximate cost-to-go of a fixed policy, in the case where the system state variables are restricted to evolve independently according to Gaussian probability transition functions [127]. Engel, Mannor, and Meir applied a Gaussian process approximation architecture to TD learning [65–67], and Reisinger, Stone, and Miikkulainen subsequently adapted this framework to allow the kernel parameters to be estimated online [130]. Tobias and Daniel proposed a LSTD approach based on support vector machines [158]. Several researchers have investigated designing specialized kernels that exploit manifold

structure in the state space [16, 17, 102, 103, 146, 150, 151]. Deisenroth, Jan, and Rasmussen used Gaussian processes in an approximate value iteration algorithm for computing the cost-to-go function [57]. Similar to the well-studied class of linear architectures, kernel-based architectures map an input pattern into a set of features; however, unlike linear architectures, the effective feature vector of a kernel-based architecture may be infinite-dimensional. This property gives kernel methods a great deal of flexibility and makes them particularly appropriate in approximate dynamic programming, where the structure of the cost-to-go function may not be well understood.

The focus of this chapter is on the development of a new class of kernel-based approximate dynamic programming algorithms that are similar in spirit to traditional Bellman residual methods. Similar to traditional methods, the new algorithms are designed to minimize an approximate form of the Bellman error as given in Eq. (3.3). The motivation behind our work is the observation that, given the approximate Bellman error $\widetilde{BE}$ as the objective function to be minimized, we should seek to find a solution for which the objective is identically zero, the smallest possible value. The ability to find such a solution depends on the richness of the function approximation architecture employed, which in turn defines the set of candidate solutions. Traditional, parametric approximation architectures such as neural networks and linear combinations of basis functions are finite-dimensional, and therefore it may not always be possible to find a solution satisfying $\widetilde{BE} = 0$ (indeed, if a poor network topology or set of basis functions is chosen, the minimum achievable error may be large). In contrast, in this chapter we shall show that by exploiting the richness and flexibility of kernel-based approximation architectures, it is possible to construct algorithms that always produce a solution for which $\widetilde{BE} = 0$. As an immediate consequence, our algorithms have the desirable property of reducing to *exact* policy iteration in the limit of sampling the entire state space, since in this limit, the Bellman residuals are zero everywhere, and therefore the obtained cost-to-go function is exact ($\widetilde{J}_\mu = J_\mu$). We refer to our approach as *Bellman residual elimination* (BRE), rather than Bellman residual minimization, to emphasize the fact that the error is explicitly forced to zero.

In this chapter, we shall assume that the system model (i.e. the state transition function $P_{ij}(u)$) is known and show that by exploiting this knowledge, algorithms can be constructed that eliminate the need to perform trajectory simulations and therefore do not suffer from simulation noise effects. In Chapter 4, we will extend the BRE approach to the case when knowledge of the model is not available.

The theoretical basis of our approach relies on the idea of Reproducing Kernel Hilbert Spaces (RKHSs) [10, 21, 71, 135, 139, 141, 167], which are a specialized type of function space especially useful in the analysis of kernel methods. Specifically, the chapter presents two related RKHSs, one corresponding to the space of candidate cost-to-go functions, and one corresponding to the space of Bellman residual functions. We show how an invertible linear mapping between these spaces can be constructed, using properties of the Bellman equation and the assumption that the kernel is non-degenerate. This mapping is useful because the desired property $\widetilde{BE} = 0$ is naturally encoded as a simple regression problem in the Bellman residual space, allowing the construction of algorithms that find a solution with this property in the Bellman residual space. Any kernel-based regression technique, such as support vector regression or Gaussian process regression, can be used to solve the regression problem, resulting in a class of related BRE algorithms. Once the solution is known, the linear mapping is used to find the corresponding cost-to-go function. The use of a nondegenerate kernel function (that is, one with an infinite-dimensional feature vector) is key to the success of this approach, since the linear mapping is not always invertible when using a finite-dimensional architecture such as those used in [23, 54, 93].

3.1 BRE Using Support Vector Regression

As a starting point for the development of BRE, this section demonstrates how the basic support vector regression problem can be used to construct a BRE algorithm. We will refer to the resulting algorithm as BRE(SV) and show that it can efficiently solve practical approximate dynamic programming problems. Subsequent sections

will show how the key ideas used in BRE(SV) can be generalized to use any kernel-based regression technique.

We begin with the following problem statement: assume that an MDP specification $(\mathcal{S}, \mathcal{A}, P, g)$ is given, along with a policy μ of the MDP. Furthermore, assume that a representative set of sample states $\widetilde{\mathcal{S}}$ is known. The goal is to construct an approximate cost-to-go $\widetilde{J}_\mu$ of the policy μ, such that the Bellman residuals at the sample states are identically zero.

Following the standard functional form assumed in the support vector regression problem [Eq. (2.9)], we express the approximate cost-to-go (at a specified state $i \in \mathcal{S}$) as the inner product between a feature mapping $\underline{\Phi}(i)$ and a set of weights $\underline{\Theta}$:

$$\widetilde{J}_\mu(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \qquad i \in \mathcal{S}. \tag{3.4}$$

The kernel $k(i, i')$ corresponding to the feature mapping $\underline{\Phi}(\cdot)$ is given by

$$k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle, \quad i, i' \in \mathcal{S}. \tag{3.5}$$

Recall that the Bellman residual at i, $BR(i)$, is defined as

$$\begin{aligned} BR(i) &\equiv \widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i) \\ &= \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \right). \end{aligned} \tag{3.6}$$

Substituting the functional form of the cost-to-go function, Eq. (3.4), into the expression for $BR(i)$ yields

$$BR(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle - \left(g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \langle \underline{\Theta}, \underline{\Phi}(j) \rangle \right).$$

Finally, by exploiting linearity of the inner product $\langle \cdot, \cdot \rangle$, we can express $BR(i)$ as

$$\begin{aligned} BR(i) &= -g_i^\mu + \langle \underline{\Theta}, \left(\underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \underline{\Phi}(j) \right) \rangle \\ &= -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle, \end{aligned} \tag{3.7}$$

where $\underline{\Psi}(i)$ is defined as

$$\underline{\Psi}(i) \equiv \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \underline{\Phi}(j). \tag{3.8}$$

$\underline{\Psi}(i)$ represents a new feature mapping that accounts for the structure of the MDP dynamics (in particular, it represents a combination of the features at i and all states j that can be reached in one step from i). The following lemma states an important property of the new feature mapping that will be important in establishing the theoretical properties of our BRE algorithms.

Lemma 1. *Assume the vectors* $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$ *are linearly independent. Then the vectors* $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$*, where* $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \underline{\Phi}(j)$*, are also linearly independent.*

This lemma, as well as all of the following lemmas, theorems and corollaries in the chapter, are proved in the appendix at the end of this chapter. The definition of $\underline{\Psi}(i)$ and the corresponding expression for the Bellman residual [Eq. (3.7)] now allow the basic support vector regression problem to be modified to find a cost-to-go function, of the form Eq. (3.4), which has small Bellman residuals at the sample states.

3.1.1 Error Function Substitution

Recall that the support vector regression problem seeks to minimize the absolute value of an ϵ-insensitive loss function, encoded by the constraints Eqs. (2.11) and (2.12). In the nominal problem, the error is

$$y_i - \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle,$$

which is just the difference between the observed function value y_i and the predicted value $f(x_i) = \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle$. In order to perform BRE over the set of sample states $\widetilde{\mathcal{S}}$, the Bellman residual is first substituted for the error function in the nominal problem:

$$\begin{aligned} \min_{\underline{\Theta}, \underline{\xi}, \underline{\xi}^\star} \quad & \frac{1}{2}||\underline{\Theta}||^2 \quad + \quad c \sum_{i \in \widetilde{\mathcal{S}}} (\xi_i + \xi_i^\star) \\ \text{subj. to} \quad & BR(i) \quad \leq \quad \epsilon + \xi_i & (3.9) \\ & -BR(i) \quad \leq \quad \epsilon + \xi_i^\star & (3.10) \\ & \xi_i, \xi_i^\star \quad \geq \quad 0 \qquad \forall i \in \widetilde{\mathcal{S}}. \end{aligned}$$

By introducing the Bellman residual as the error function to be minimized, the optimization process will seek to find a solution such that the residuals are small at the sample states, as desired. If the optimization problem can be solved, the solution yields the values of the weights $\underline{\Theta}$, which in turn uniquely determine the cost-to-go function $\widetilde{J}_\mu$ through Eq. (3.4). However, it is not yet clear if this optimization problem still fits the general form of the support vector regression problem [Eqs. (2.10)-(2.13)]. To see that it does, the expression for the Bellman residual [Eq. (3.7)] is substituted in the constraints Eqs. (3.9)-(3.10), and the following optimization problem is obtained:

$$\begin{aligned} \min_{\underline{\Theta}, \underline{\xi}, \underline{\xi}^\star} \quad & \frac{1}{2}||\underline{\Theta}||^2 \quad + \quad c \sum_{i \in \widetilde{\mathcal{S}}} (\xi_i + \xi_i^\star) & (3.11) \\ \text{subj. to} \quad & -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle \quad \leq \quad \epsilon + \xi_i & (3.12) \\ & g_i^\mu - \langle \underline{\Theta}, \underline{\Psi}(i) \rangle \quad \leq \quad \epsilon + \xi_i^\star & (3.13) \\ & \xi_i, \xi_i^\star \quad \geq \quad 0 \qquad \forall i \in \widetilde{\mathcal{S}}. & (3.14) \end{aligned}$$

Eqs. (3.11)-(3.14) are referred to as the *Bellman residual minimization problem.* This problem is identical to the basic support vector regression problem [Eqs. (2.10)-(2.13)]. The original feature mapping $\underline{\Phi}(i)$ has been replaced by the new feature mapping $\underline{\Psi}(i)$; this is only a notational change and does not alter the structure of the basic problem in any way. Furthermore, the one-stage costs g_i^μ have replaced the generic observations y_i, but again, this is only a notational change. Therefore, the dual problem is given by

the normal support vector regression dual [Eqs. (2.14)-(2.15)], with the substitutions $\underline{\Phi}(i) \rightarrow \underline{\Psi}(i)$ and $y_i \rightarrow g_i^\mu$:

$$\begin{aligned} \max_{\underline{\lambda},\underline{\lambda}^\star} \quad & -\tfrac{1}{2}\textstyle\sum_{i,i'\in\widetilde{\mathcal{S}}}(\lambda_i^\star - \lambda_i)(\lambda_{i'}^\star - \lambda_{i'})\langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle \\ & -\epsilon\textstyle\sum_{i\in\widetilde{\mathcal{S}}}(\lambda_i^\star + \lambda_i) + \sum_{i\in\widetilde{\mathcal{S}}} g_i^\mu(\lambda_i^\star - \lambda_i) \qquad (3.15) \\ \text{subj. to} \quad & 0 \leq \lambda_i, \lambda_i^\star \leq c \quad \forall i \in \widetilde{\mathcal{S}}. \qquad (3.16) \end{aligned}$$

Notice that the dual of the Bellman residual minimization problem contains a new kernel function, denoted by $\mathcal{K}$:

$$\mathcal{K}(i,i') = \langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle.$$

This kernel is related to the base kernel k [Eq. (3.5)] through the feature mapping $\underline{\Psi}$ [Eq. (3.8)]:

$$\begin{aligned} \mathcal{K}(i,i') &= \langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle \\ &= \langle\ \underline{\Phi}(i) - \alpha\sum_{j\in\mathcal{S}} P_{ij}^\mu\underline{\Phi}(j)\ ,\ \underline{\Phi}(i') - \alpha\sum_{j\in\mathcal{S}} P_{i'j}^\mu\underline{\Phi}(j)\ \rangle \\ &= \langle\underline{\Phi}(i),\underline{\Phi}(i')\rangle - \alpha\sum_{j\in\mathcal{S}}\left(P_{i'j}^\mu\langle\underline{\Phi}(i),\underline{\Phi}(j)\rangle + P_{ij}^\mu\langle\underline{\Phi}(i'),\underline{\Phi}(j)\rangle\right) \\ &\qquad +\alpha^2\sum_{j,j'\in\mathcal{S}} P_{ij}^\mu P_{i'j'}^\mu\langle\underline{\Phi}(j),\underline{\Phi}(j')\rangle \\ &= k(i,i') - \alpha\sum_{j\in\mathcal{S}}\left(P_{i'j}^\mu k(i,j) + P_{ij}^\mu k(i',j)\right) \\ &\qquad +\alpha^2\sum_{j,j'\in\mathcal{S}} P_{ij}^\mu P_{i'j'}^\mu k(j,j'). \qquad (3.17) \end{aligned}$$

We shall refer to $\mathcal{K}$ as the *Bellman kernel associated with* k. The following theorem establishes an important property of this kernel.

Theorem 2. *Assume that the kernel* $k(i,i') = \langle\underline{\Phi}(i),\underline{\Phi}(i')\rangle$ *is nondegenerate. Then the associated Bellman kernel defined by* $\mathcal{K}(i,i') = \langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle$, *where* $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha\sum_{j\in\mathcal{S}} P_{ij}^\mu\underline{\Phi}(j)$, *is also nondegenerate.*

In order to specify and solve the dual problem [Eqs. (3.15)-(3.16)], the one-stage

cost values g_i^μ and the kernel values $\mathcal{K}(i,i')$ must be computed. Since by assumption the MDP model is known, the state transition probabilities P_{ij}^μ [Eq. (2.6)] are available, and thus Eq. (3.17) allows direct computation of the kernel values $\mathcal{K}(i,i')$. Furthermore, the g_i^μ values are also available [Eq. (2.5)]. Thus, all information necessary to formulate and solve the dual problem is known.

Once the kernel values $\mathcal{K}(i,i')$ and the cost values g_i^μ are computed, the dual problem is completely specified and can be solved using, for example, a standard SVM solving package such as libSVM [44]. The solution yields the dual variables $\underline{\lambda}, \underline{\lambda}^\star$. Again, in direct analogy with the normal SV regression problem, the primal variables $\underline{\Theta}$ are given by the support vector expansion [Eq. (2.16)], with the substitution $\underline{\Phi} \to \underline{\Psi}$:

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} (\lambda_i - \lambda_i^\star) \underline{\Psi}(i).$$

Finally, in order to find the cost-to-go $\widetilde{J}_\mu(i)$, the expressions for $\underline{\Theta}$ and $\underline{\Psi}$ are substituted into Eq. (3.4):

$$\begin{aligned}
\widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \langle \underline{\Psi}(i'), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \langle \left(\underline{\Phi}(i') - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu \underline{\Phi}(j) \right), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \left(\langle \underline{\Phi}(i'), \underline{\Phi}(i) \rangle - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu \langle \underline{\Phi}(j), \underline{\Phi}(i) \rangle \right) \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \left(k(i',i) - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu k(j,i) \right). \qquad (3.18)
\end{aligned}$$

Thus, once the dual variables are solved for, Eq. (3.18) can be used to calculate the primary object of interest, the cost-to-go function $\widetilde{J}_\mu(i)$.

This section has shown how the basic support vector regression problem can be modified to find a cost-to-go function whose Bellman residuals are small at the sample states $\widetilde{\mathcal{S}}$. A summary of the procedure is given in Algorithm 1. While this algorithm is

Algorithm 1 Support vector policy evaluation, Bellman residuals not explicitly forced to zero

1: **Input:** (μ, $\widetilde{\mathcal{S}}$, k, ϵ, c)
2: μ: policy to be evaluated
3: $\widetilde{\mathcal{S}}$: set of sample states
4: k: kernel function defined on $\mathcal{S} \times \mathcal{S}$
5: ϵ: width of ϵ-insensitive loss function
6: c: model complexity parameter
7: **Begin**
8: $\forall i, i' \in \widetilde{\mathcal{S}}$, compute $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$ using Eq. (3.17)
9: $\forall i \in \widetilde{\mathcal{S}}$, compute g_i^μ using Eq. (2.5)
10: Using $\mathcal{K}(i, i')$ and g_i^μ values, solve the dual optimization problem Eqs. (3.15)-(3.16) for the dual variables $\underline{\lambda}, \underline{\lambda}^\star$
11: Using the dual solution variables $\underline{\lambda}, \underline{\lambda}^\star$, compute the cost-to-go $\widetilde{J}_\mu(i)$ using Eq. (3.18)
12: **End**

directly related to the basic support vector regression problem, it does not yet achieve the goal of explicitly forcing all of the Bellman residuals to zero. Intuitively, this is because the model complexity penalty (controlled by the parameter c) may prevent the optimal solution from achieving the lowest possible value of the Bellman residuals. The next section will explain how to modify Algorithm 1 in order to explicitly force the Bellman residuals to zero.

3.1.2 Forcing the Bellman Residuals to Zero

The basic support vector regression problem is designed to deal with noisy observations of the true function values. With noisy observations, the problem of overfitting becomes important, and choosing a regression function that matches the observations exactly may not be desirable. The use of the ϵ-insensitive loss function helps to alleviate the overfitting problem, allowing observation points to lie within a distance ϵ of the regression function without causing the objective function to increase. This loss function leads to good performance with noisy data sets.

However, in Algorithm 1, the "observations" g_i^μ are *exact* values dictated by the structure of the MDP. Therefore, overfitting is not a concern, since we are working directly with the Bellman residuals, which are exact mathematical relationships be-

tween the cost-to-go values $J_\mu(i)$ for all states $i \in \mathcal{S}$. Indeed, in order to force the Bellman residuals to zero as desired, the regression function $\langle \underline{\Theta}, \underline{\Psi}(i) \rangle$ must match the "observation" g_i^μ exactly:

$$BR(i) = -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = 0 \quad \Longleftrightarrow \quad \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = g_i^\mu.$$

Fortunately, the support vector regression problem can be modified to perform exact regression, where the learned regression function passes exactly through each of the observation points. This is accomplished by setting $\epsilon = 0$ and $c = \infty$; intuitively, this causes the objective function to be unbounded whenever the regression function does not match an observation. Having fixed c and ϵ, the Bellman residual minimization problem [Eqs. (3.11)-(3.14)] can be recast in a simpler form. With $c = \infty$, any feasible solution must have $\xi_i, \xi_i^\star = 0$ for all i, since otherwise the objective function will be unbounded. Furthermore, if ϵ is also zero, then the constraints [Eqs. (3.12) and (3.13)] become equalities. With these modifications, the primal problem reduces to

$$\begin{aligned} & \min_{\underline{\Theta}} \quad \frac{1}{2}||\underline{\Theta}||^2 \\ \text{subj. to} \quad & g_i^\mu - \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = 0 \qquad \forall i \in \widetilde{\mathcal{S}}, \end{aligned} \tag{3.19}$$

where $\underline{\Psi}(i)$ is given by Eq. (3.8). The Lagrangian dual of this optimization problem can be calculated as follows. The Lagrangian is

$$\mathcal{L}(\underline{\Theta}, \underline{\lambda}) = \frac{1}{2}||\underline{\Theta}||^2 + \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \left(g_i^\mu - \langle \underline{\Theta}, \underline{\Psi}(i) \rangle \right). \tag{3.20}$$

Maximizing $\mathcal{L}(\underline{\Theta}, \underline{\lambda})$ with respect to $\underline{\Theta}$ is accomplished by setting the corresponding partial derivative to zero:

$$\frac{\partial \mathcal{L}}{\partial \underline{\Theta}} = \underline{\Theta} - \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Psi}(i) = 0,$$

and therefore

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Psi}(i). \tag{3.21}$$

As a result, the cost-to-go function $\widetilde{J}_\mu(i)$ [Eq. (3.4)] is given by

$$\begin{aligned} \widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \langle \underline{\Psi}(i'), \underline{\Phi}(i) \ \rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i', i) - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} k(j, i) \right). \end{aligned} \tag{3.22}$$

Finally, substituting Eq. (3.21) into Eq. (3.20) and maximizing with respect to $\underline{\lambda}$ gives the dual problem:

$$\max_{\underline{\lambda}} \ -\frac{1}{2} \sum_{i, i' \in \widetilde{\mathcal{S}}} \lambda_i \lambda_{i'} \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle + \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i g^\mu_i.$$

This can also be written in vector form as

$$\max_{\underline{\lambda}} \ -\frac{1}{2} \underline{\lambda}^T \mathbb{K} \underline{\lambda} + \underline{\lambda}^T \underline{g}^\mu, \tag{3.23}$$

where

$$\mathbb{K}_{ii'} = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle = \mathcal{K}(i, i')$$

is the Gram matrix of the kernel $\mathcal{K}$ (this matrix is calculated using Eq. (3.17)). Note that the problem is just an unconstrained maximization of a quadratic form. Since by assumption the base kernel k is nondegenerate, Theorem 2 implies that the associated Bellman kernel $\mathcal{K}$ is nondegenerate also. Therefore, the Gram matrix $\mathbb{K}$ is full-rank and strictly positive definite, so the quadratic form has a unique maximum. The solution is found analytically by setting the derivative of the objective with respect to $\underline{\lambda}$ to zero, which yields

$$\mathbb{K} \underline{\lambda} = \underline{g}^\mu. \tag{3.24}$$

Algorithm 2 Support vector policy evaluation, Bellman residuals explicitly forced to zero

Input: $(\mu, \widetilde{\mathcal{S}}, k)$
μ: policy to be evaluated
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel function defined on $\mathcal{S} \times \mathcal{S}$
Begin
Construct the Gram matrix $\mathbb{K}$, where $\mathbb{K}_{ii'} = \mathcal{K}(i, i') \;\; \forall i, i' \in \widetilde{\mathcal{S}}$, using Eq. (3.17)
$\forall i \in \widetilde{\mathcal{S}}$, compute g_i^μ using Eq. (2.5)
Using $\mathcal{K}(i, i')$ and g_i^μ values, solve the linear system Eq. (3.24) for $\underline{\lambda}$
Using solution $\underline{\lambda}$, compute the cost-to-go $\widetilde{J}_\mu(i)$ using Eq. (3.22)
End

Note the important fact that the dimension of this linear system is $n_s = |\widetilde{\mathcal{S}}|$, the number of sampled states.

The development of the policy evaluation procedure is now complete and is summarized in Algorithm 2. The key computational step in the algorithm is solving the linear system Eq. (3.24). Recall that the original problem of solving the full, fixed-policy Bellman equation [Eq. (2.4)] involves solving an N-dimensional linear system, where $N = |\mathcal{S}|$ is the size of the entire state space. After developing a support vector regression-based method for approximating the cost-to-go and reformulating it to force Bellman residuals to zero, the approximate policy evaluation problem has been reduced to the problem of solving another linear system [Eq. (3.24)]. However, the dimensionality of this system is only $n_s = |\widetilde{\mathcal{S}}|$, where $n_s \ll N$. Furthermore, since the designer is in control of $\widetilde{\mathcal{S}}$, he can select an appropriate number of sample states based on the computational resources available.

The following theorem establishes the important claim that the Bellman residuals are exactly zero at the sampled states.

Theorem 3. *Assume that the kernel $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \qquad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ are identically equal to zero at every state $i \in \widetilde{\mathcal{S}}$.

An immediate corollary of Theorem 3 follows:

Corollary 4. *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = J_\mu(i) \qquad \forall i \in \mathcal{S}.$$

That is, the cost-to-go function $\widetilde{J}_\mu(i)$ is exact.

Recovering the Bellman equation

Corollary 4 claims that Algorithm 2 yields the exact cost-to-go when the entire state space is sampled. Recall that the exact cost-to-go can be computed by solving the Bellman equation

$$\underline{J}_\mu = (I - \alpha P^\mu)^{-1} \underline{g}^\mu.$$

Intuitively, therefore, the Bellman equation must be somehow "embedded" into the algorithm. The following calculation shows that this is indeed the case: the Bellman equation can be recovered from Algorithm 2 under a suitable choice of kernel. We begin by taking the kernel function $k(i,i')$ to be the Kronecker delta function:

$$k(i,i') = \delta_{ii'}.$$

This kernel is nondegenerate, since its Gram matrix is always the identity matrix, which is clearly positive definite. Furthermore, we take $\widetilde{\mathcal{S}} = \mathcal{S}$, so that both conditions of Corollary 4 are met. Now, notice that with this choice of kernel, the associated Bellman kernel [Eq. (3.17)] is

$$\begin{aligned} \mathcal{K}(i,i') &= \delta_{ii'} - \alpha \sum_{j \in \mathcal{S}} \left(P^\mu_{i'j} \delta_{ij} + P^\mu_{ij} \delta_{i'j} \right) + \alpha^2 \sum_{j,j' \in \mathcal{S}} P^\mu_{ij} P^\mu_{i'j'} \delta_{jj'} \\ &= \delta_{ii'} - \alpha \left(P^\mu_{i'i} + P^\mu_{ii'} \right) + \alpha^2 \sum_{j \in \mathcal{S}} P^\mu_{ij} P^\mu_{i'j}. \end{aligned} \tag{3.25}$$

Line 6 in Algorithm 2 computes the Gram matrix $\mathbb{K}$ of the associated Bellman kernel. A straightforward calculation shows that, with the associated Bellman kernel given by Eq. (3.25), the Gram matrix is

$$\mathbb{K} = (I - \alpha P^{\mu})^2.$$

Line 7 simply constructs the vector $\underline{g}^{\mu}$, and Line 8 computes $\underline{\lambda}$, which is given by

$$\underline{\lambda} = \mathbb{K}^{-1}\underline{g}^{\mu} = (I - \alpha P^{\mu})^{-2}\underline{g}^{\mu}.$$

Once $\underline{\lambda}$ is known, the cost-to-go function $\widetilde{J}_{\mu}(i)$ is computed in Line 9:

$$\begin{aligned}
\widetilde{J}_{\mu}(i) &= \sum_{i' \in \mathcal{S}} \lambda_{i'} \left(\delta_{i'i} - \alpha \sum_{j \in \mathcal{S}} P^{\mu}_{i'j} \delta_{ji} \right) \\
&= \lambda_i - \alpha \sum_{i' \in \mathcal{S}} \lambda_{i'} P^{\mu}_{i'i}.
\end{aligned}$$

Expressed as a vector, this cost-to-go function can now be written as

$$\begin{aligned}
\underline{\widetilde{J}}_{\mu} &= \underline{\lambda} - \alpha P^{\mu}\underline{\lambda} \\
&= (I - \alpha P^{\mu})\underline{\lambda} \\
&= (I - \alpha P^{\mu})(I - \alpha P^{\mu})^{-2}\underline{g}^{\mu} \\
&= (I - \alpha P^{\mu})^{-1}\underline{g}^{\mu} \\
&= \underline{J}_{\mu}.
\end{aligned}$$

Thus, we see that for the choice of kernel $k(i, i') = \delta_{ii'}$ and sample states $\widetilde{\mathcal{S}} = \mathcal{S}$, Algorithm 2 effectively reduces to solving the Bellman equation directly. Of course, if the Bellman equation could be solved directly for a particular problem, it would make little sense to use an approximation algorithm of any kind. Furthermore, the Kronecker delta is a poor practical choice for the kernel, since the kernel is zero everywhere except at $i = i'$; we use it here only to illustrate the properties of the algorithm. The real value of the algorithm arises because the main results—namely,

that the Bellman residuals are identically zero at the sample states—are valid for *any* nondegenerate kernel function and *any* set of sample states $\widetilde{\mathcal{S}} \subseteq \mathcal{S}$. This permits the use of "smoothing" kernels that enable the computed cost-to-go function to generalize to states that were not sampled, which is a major goal of any approximate dynamic programming algorithm. Thus, in the more realistic case where solving the Bellman equation directly is not practical, Algorithm 2 (with a smaller set of sample states and a suitable kernel function) becomes a way of effectively reducing the dimensionality of the Bellman equation to a tractable size while preserving the character of the original equation.

The associated Bellman kernel [Eq. (3.17)] plays a central role in the work developed thus far; it is this kernel and its associated feature mapping $\underline{\Psi}$ [Eq. (3.8)] that effectively allow the Bellman equation to be embedded into the algorithms, and consequently to ensure that the Bellman residuals at the sampled states are zero. This idea will be developed further in Section 3.2.

3.1.3 BRE(SV), A Full Policy Iteration Algorithm

Algorithm 2 shows how to construct a cost-to-go approximation $\widetilde{J}_\mu(\cdot)$ of a fixed policy μ such that the Bellman residuals are exactly zero at the sample states $\widetilde{\mathcal{S}}$. This section now presents BRE(SV) (Algorithm 3), a full policy iteration algorithm that uses Algorithm 2 in the policy evaluation step.

The following corollary to Theorem 3 establishes that BRE(SV) reduces to exact policy iteration when the entire state space is sampled:

Corollary 5. *Assume that the kernel* $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ *is nondegenerate, and that* $\widetilde{\mathcal{S}} = \mathcal{S}$. *Then BRE(SV) is equivalent to exact policy iteration.*

This result is encouraging, since it is well known that exact policy iteration converges to the optimal policy in a finite number of steps [22].

Algorithm 3 BRE(SV)

Input: $(\mu_0, \widetilde{\mathcal{S}}, k(\cdot,\cdot))$
μ_0: initial policy
$\widetilde{\mathcal{S}}$: set of sample states
$k(\cdot,\cdot)$: kernel function defined on $\mathcal{S} \times \mathcal{S}$
Begin
$\mu \leftarrow \mu_0$
loop
  Construct the Gram matrix $\mathbb{K}$, where $\mathbb{K}_{ii'} = \mathcal{K}(i,i') \ \ \forall i, i' \in \widetilde{\mathcal{S}}$, using Eq. (3.17)
  $\forall i \in \widetilde{\mathcal{S}}$, compute g_i^μ
  Using $\mathbb{K}_{ii'}$ and g_i^μ values, solve the linear system $\mathbb{K}\underline{\lambda} = \underline{g}^\mu$ for $\underline{\lambda}$
  Using solution $\underline{\lambda}$, compute the cost-to-go $\widetilde{J}_\mu(i)$ using Eq. (3.18) {Policy evaluation complete}
  $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u) \Big(g(i,u) + \alpha \widetilde{J}_\mu(j) \Big)$ {Policy improvement}
end loop
End

3.1.4 Computational Results - Mountain Car

BRE(SV) was implemented on the well-known "mountain car problem" [127, 154] to evaluate its performance. In this problem, a unit mass, frictionless car moves along a hilly landscape whose height $H(x)$ is described by

$$H(x) = \begin{cases} x^2 + x & \text{if } x < 0 \\ \frac{x}{\sqrt{1+5x^2}} & \text{if } x \geq 0 \end{cases}$$

The system state is given by $(x, \dot{x})$ (the horizontal position and velocity of the car). A horizontal control force $-4 \leq u \leq 4$ can be applied to the car, and the goal is to drive the car from its starting location $x = -0.5$ to the "parking area" $0.5 \leq x \leq 0.7$ as quickly as possible. Since the car is also acted upon by gravitational acceleration $g = 9.8$, the total horizontal acceleration of the car, $\ddot{x}$, is given by

$$\ddot{x} = \frac{u - gH'(x)}{1 + H'(x)^2},$$

where $H'(x)$ denotes the derivative $\frac{dH(x)}{dx}$. The problem is challenging because the car is underpowered: it cannot simply drive up the steep slope. Rather, it must use the features of the landscape to build momentum and eventually escape the steep valley centered at $x = -0.5$. The state space of the problem is naturally continuous; to convert it into a discrete MDP, Kuhn triangulation [111] is used. The system response under the optimal policy (computed using value iteration) is shown as the dashed line in Figure 3-2; notice that the car initially moves *away* from the parking area before reaching it at time $t = 14$.

In order to apply the support vector policy iteration algorithm, an evenly spaced 9x9 grid of sample states,

$$\widetilde{\mathcal{S}} = \{(x, \dot{x}) \mid x = -1.0, -0.75, \ldots, 0.75, 1.0 \\ \dot{x} = -2.0, -1.5, \ldots, 1.5, 2.0\}$$

was chosen. Furthermore, a radial basis function kernel, with differing length-scales for the x and $\dot{x}$ axes in the state space, was used:

$$k((x_1, \dot{x}_1), (x_2, \dot{x}_2)) = \exp\left(-(x_1 - x_2)^2/(0.25)^2 - (\dot{x}_1 - \dot{x}_2)^2/(0.40)^2\right).$$

The length-scales were chosen by hand through experimentation. BRE(SV) was executed, resulting in a sequence of policies (and associated cost-to-go functions) that converged after three iterations. The sequence of cost-to-go functions is shown in Figure 3-1 along with the optimal cost-to-go function for comparison. Of course, the main objective is to learn a policy that is similar to the optimal one. The solid line in Figure 3-2 shows the system response under the approximate policy generated by the algorithm after 3 iterations. Notice that the qualitative behavior is the same as the optimal policy; that is, the car first accelerates away from the parking area to gain momentum. The approximate policy arrives at the parking area at $t = 17$, only 3 time steps slower than the optimal policy.

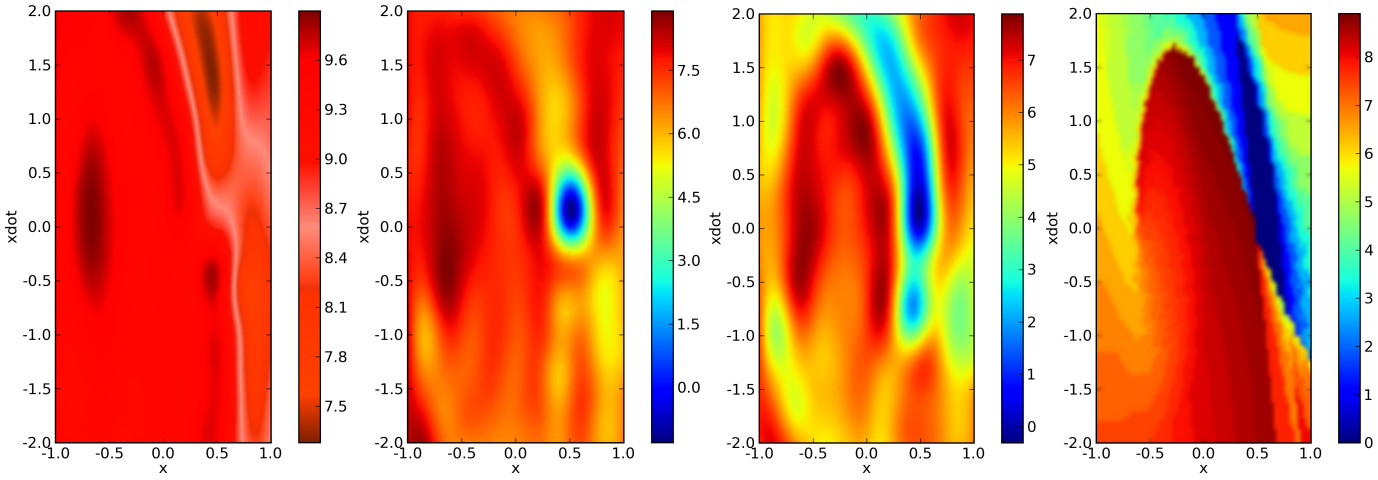


Figure 3-1: From left to right: Approximate cost-to-go computed by BRE(SV) for iterations 1, 2 , and 3; exact cost-to-go computed using value iteration.

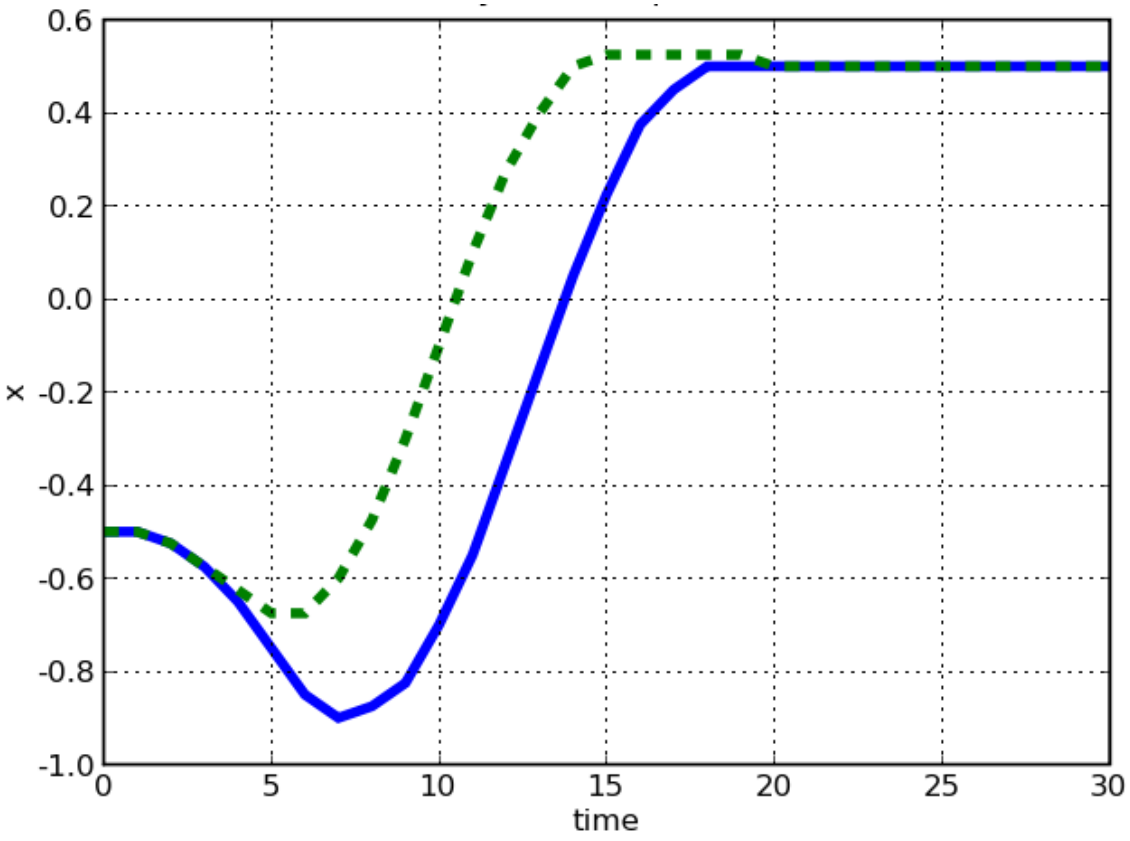


Figure 3-2: System response under the optimal policy (dashed line) and the policy computed by BRE(SV) after 3 iterations (solid line).

BRE(SV) was derived starting from the viewpoint of the standard support vector regression framework. The theorems of this section have established several advantageous theoretical properties of the approach, and furthermore, the results from the mountain car example problem demonstrate that BRE(SV) produces a high-quality cost-to-go approximation and policy. The example problem highlights two interesting questions:

1. The scaling parameters (0.25 and 0.40) used in the squared exponential kernel were found, in this case, by carrying out numerical experiments to see which values fit the data best. Is it possible to design a BRE algorithm that can learn these values automatically as it runs?

2. Is it possible to provide error bounds on the learned cost-to-go function for the non-sampled states?

Fortunately, the answer to both of these questions is yes. In the next section, we show how the key ideas in BRE(SV) can be extended to a more general framework, which allows BRE to be performed using any kernel-based regression technique. This analysis will result in a deeper understanding of the approach and also enable the development of a BRE algorithm based on Gaussian process regression, BRE(GP), that preserves all the advantages of BRE(SV) while addressing these two questions.

3.2 A General BRE Framework

Building on the ideas and intuition presented in the development of BRE(SV), we now seek to understand how that algorithm can be generalized. To begin, it is useful to view kernel-based regression techniques, such as support vector regression and Gaussian process regression, as algorithms that search over a reproducing kernel Hilbert space for a regression function solution. In particular, given a kernel k and a set of training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, these kernel-based regression techniques

find a solution of the standard form

$$f(x) = \langle \underline{\Theta}, \underline{\Phi}(x) \rangle,$$

where $\underline{\Phi}$ is the feature mapping of the kernel k. The solution f belongs to the RKHS $\mathcal{H}_k$ of the kernel k:

$$f(\cdot) \in \mathcal{H}_k.$$

Broadly speaking, the various kernel-based regression techniques are differentiated by how they choose the weighting element $\underline{\Theta}$, which in turn uniquely specifies the solution f from the space of functions $\mathcal{H}_k$.

In the BRE(SV) algorithm developed in the previous section, the kernel k and its associated Bellman kernel $\mathcal{K}$ play a central role. Recall the relationship between these two kernels:

$$\begin{aligned} k(i,i') &= \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle & (3.26) \\ \underline{\Psi}(i) &= \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j) & (3.27) \\ \mathcal{K}(i,i') &= \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle. & (3.28) \end{aligned}$$

By the uniqueness property of Section 2.4, each of the kernels is associated with its own, unique RKHS, denoted by $\mathcal{H}_k$ and $\mathcal{H}_{\mathcal{K}}$, respectively. In this section, we explore some properties of these RKHSs, and use the properties to develop a general BRE framework. In particular, it is shown that $\mathcal{H}_k$ corresponds to the space of possible cost-to-go functions, while $\mathcal{H}_{\mathcal{K}}$ corresponds to the space of Bellman residual functions. We explain how the problem of performing BRE is equivalent to solving a simple regression problem in $\mathcal{H}_{\mathcal{K}}$ in order to find a Bellman residual function with the desired property of being zero at the sampled states $\widetilde{\mathcal{S}}$. This regression problem can be solved using any kernel-based regression technique. Furthermore, an invertible linear mapping between elements of $\mathcal{H}_k$ and $\mathcal{H}_{\mathcal{K}}$ is constructed. This mapping can be used to find the corresponding cost-to-go function once the Bellman residual function is found.

3.2.1 The Cost-To-Go Space $\mathcal{H}_k$

The goal of any approximate policy iteration algorithm, including our BRE methods, is to learn an approximation $\widetilde{J}_\mu(\cdot)$ to the true cost-to-go function $J_\mu(\cdot)$. As discussed, if a kernel-based regression algorithm, with kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$, is used to find the approximate cost-to-go function $\widetilde{J}_\mu(\cdot)$, then this function is an element of $\mathcal{H}_k$ and takes the standard form

$$\widetilde{J}_\mu(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle.$$

This form was the starting point for the development of the BRE(SV) algorithm [Eq. (3.4)]. It is important to note that the weighting element $\underline{\Theta}$ is itself an element of the RKHS $\mathcal{H}_k$, and therefore can be expressed in terms of the basis functions $\{\underline{\Phi}(i) \mid i \in \widetilde{\mathcal{S}}\}$, which span $\mathcal{H}_k$:

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Phi}(i).$$

The structure of the kernel k and cost-to-go functions $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ are summarized in Figure 3-3. Note that there are two equivalent forms for representing $\widetilde{J}_\mu(\cdot)$ [Eqs. (3.30) and (3.31)]. Eq. (3.30) is important from a computational standpoint because the output of a general kernel-based regression method is explicitly of this compact form (that is, the input to the regression method is a set of training data $\mathcal{D}$, and the output are the coefficients λ_i). Eq. (3.31) is important because it helps establish the mapping from $\mathcal{H}_k$ to $\mathcal{H}_\mathcal{K}$, as will be shown in the next section.

3.2.2 The Bellman Residual Space $\mathcal{H}_\mathcal{K}$

The previous section showed that the RKHS $\mathcal{H}_k$ contains the possible set of cost-to-go functions $\widetilde{J}_\mu(\cdot)$. This section will show that the RKHS $\mathcal{H}_\mathcal{K}$ of the associated Bellman kernel contains functions which are closely related to the set of Bellman residual functions $BR(\cdot)$ associated with the cost-to-go functions $\widetilde{J}_\mu(\cdot)$. The structure of $\mathcal{H}_\mathcal{K}$ is summarized in Figure 3-4; in particular, note that every element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ can

1. Using the reproducing kernel map,

$$\underline{\Phi}(i)(\cdot) = k(\cdot, i),$$

the kernel $k(\cdot, \cdot)$ is represented as

$$k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle \tag{3.29}$$

2. An arbitrary element $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ is represented in one of two equivalent forms:

$$\widetilde{J}_\mu(\cdot) = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i k(\cdot, i) \tag{3.30}$$

$$= \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle, \tag{3.31}$$

where

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Phi}(i).$$

In either form, $\widetilde{J}_\mu(\cdot)$ is uniquely specified by the expansion coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ and sample states $\{i \mid i \in \widetilde{\mathcal{S}}\}$.

Figure 3-3: Structure of the cost-to-go space $\mathcal{H}_k$

be expressed as

$$\widetilde{W}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle. \tag{3.32}$$

Recall that, given any cost-to-go function $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$, the associated Bellman residual function $BR(\cdot)$ is defined by

$$BR(i) = \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \right).$$

The derivation leading to Eq. (3.7) showed that this Bellman residual function can be expressed as

$$BR(i) = -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle.$$

Finally, identifying $\langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle$ as an element $\widetilde{W}_\mu(\cdot)$ of $\mathcal{H}_\mathcal{K}$ allows the Bellman residual to be written as

$$BR(i) = -g_i^\mu + \widetilde{W}_\mu(i). \tag{3.33}$$

1. Using the reproducing kernel map,

$$\underline{\Psi}(i)(\cdot) = \mathcal{K}(\cdot, i),$$

the kernel $\mathcal{K}(\cdot,\cdot)$ is represented as

$$\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle \tag{3.34}$$

2. An arbitrary element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ is represented in one of two equivalent forms:

$$\widetilde{W}_\mu(\cdot) = \sum_{i \in \widetilde{\mathcal{S}}} \beta_i \mathcal{K}(\cdot, i) \tag{3.35}$$

$$= \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle, \tag{3.36}$$

where

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \beta_i \underline{\Psi}(i). \tag{3.37}$$

In either form, $\widetilde{W}_\mu(\cdot)$ is uniquely specified by the expansion coefficients $\{\beta_i \mid i \in \widetilde{\mathcal{S}}\}$ and sample states $\{i \mid i \in \widetilde{\mathcal{S}}\}$.

Figure 3-4: Structure of the Bellman residual space $\mathcal{H}_\mathcal{K}$

By the preceding construction, every cost-to-go function $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ has a corresponding function $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$, and this function $\widetilde{W}_\mu(\cdot)$ is equal to the Bellman residual function up to the known factor $-g_i^\mu$. In other words, a mapping $\mathcal{H}_k \to \mathcal{H}_\mathcal{K}$ has been constructed which, given a cost-to-go function $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$, allows us to find its corresponding residual function $\widetilde{W}_\mu$ in $\mathcal{H}_\mathcal{K}$. However, we are interested in algorithms which first compute the residual function and then find the corresponding cost-to-go function. Therefore, the mapping must be shown to be invertible. The following theorem establishes this important property:

Theorem 6. *Assume that the kernel $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then there exists a linear, invertible mapping $\hat{A}$ from $\mathcal{H}_k$, the RKHS of k, to $\mathcal{H}_\mathcal{K}$, the RKHS of the associated Bellman kernel $\mathcal{K}$:*

$$\hat{A}: \qquad \mathcal{H}_k \to \mathcal{H}_\mathcal{K}$$

$$\hat{A}\widetilde{J}_\mu(\cdot) = \widetilde{W}_\mu(\cdot).$$

Furthermore, if an element $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ *is given by*

$$\widetilde{J}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle,$$

then the corresponding element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ *is given by*

$$\widetilde{W}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle,$$

and vice versa:

$$\begin{array}{rcl} \widetilde{J}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ & \Updownarrow & \\ \widetilde{W}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle. \end{array} \tag{3.38}$$

The mapping $\hat{A}$ of Theorem 6 allows us to design BRE algorithms that find the weight element $\underline{\Theta}$ of the desired residual function $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ and then map the result back to $\mathcal{H}_k$ in order to find the associated cost-to-go function $\widetilde{J}_\mu(\cdot)$ (using Eq. (3.38)).

The objective of the BRE algorithms is to force the Bellman residuals $BR(i)$ to zero at the sample states $\widetilde{\mathcal{S}}$:

$$BR(i) = 0 \quad \forall i \in \widetilde{\mathcal{S}}. \tag{3.39}$$

Now, using Eq. (3.33), we see that Eq. (3.39) is equivalent to

$$\widetilde{W}_\mu(i) = g_i^\mu \quad \forall i \in \widetilde{\mathcal{S}}. \tag{3.40}$$

We have now succeeded in reducing the BRE problem to a straightforward regression problem in $\mathcal{H}_\mathcal{K}$: the task is to find a function $\widetilde{W}_\mu(i) \in \mathcal{H}_\mathcal{K}$ that satisfies Eq. (3.40). The training data of the regression problem is given by

$$\mathcal{D} = \{(i, g_i^\mu) \mid i \in \widetilde{\mathcal{S}}\}.$$

Algorithm 4 Generalized approximate policy iteration using kernel-based BRE

Input: $(\mu_0, \widetilde{\mathcal{S}}, k, \mathcal{R})$
μ_0: initial policy
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel function defined on $\mathcal{S} \times \mathcal{S}$, $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$
$\mathcal{R}$: generic kernel-based regression method
Begin
Define $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$ using Eq. (3.17) {Define the associated Bellman kernel}
$\mu \leftarrow \mu_0$
loop
  Set $\mathcal{D} = \{(i, g_i^\mu) \mid i \in \widetilde{\mathcal{S}}\}$. {Construct the training data set}
  Compute $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\} = \mathcal{R}(\mathcal{D}, \mathcal{K})$. {Solve the regression problem, using the regression method $\mathcal{R}$ and the associated Bellman kernel $\mathcal{K}$}
  Using the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$, compute the cost-to-go function $\widetilde{J}_\mu(i)$ using Eq. (3.42). {Policy evaluation step complete}
  $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u) \left(g(i, u) + \alpha \widetilde{J}_\mu(j) \right)$ {Policy improvement}
end loop
End

Notice that the training data is readily available, since by assumption the sample states $\widetilde{\mathcal{S}}$ are given, and the stage costs g_i^μ are obtained directly from the MDP specification.

3.2.3 A Family of Kernel-Based BRE Algorithms

With the invertible mapping between the RKHSs $\mathcal{H}_k$ and $\mathcal{H}_\mathcal{K}$ established, a generalized family of algorithms for kernel-based BRE can now be presented. The family is summarized in Algorithm 4. An important input to the algorithms is a generic kernel-based regression method (denoted by $\mathcal{R}$), which takes the set of training data $\mathcal{D} = \{(i, g_i^\mu) \mid i \in \widetilde{\mathcal{S}}\}$ and the associated Bellman kernel $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$, and outputs the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ such that the regression function solution $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ is given by

$$\widetilde{W}_\mu(i) = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \mathcal{K}(i, i').$$

The corresponding weight element $\underline{\Theta}$ is given by

$$\underline{\Theta} = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \underline{\Psi}(i'). \tag{3.41}$$

For notational convenience, the process of solving the regression problem using the regression method $\mathcal{R}$ to obtain the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ is denoted by

$$\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\} = \mathcal{R}(\mathcal{D}, \mathcal{K}).$$

By choosing different kernel-based regression techniques as the input $\mathcal{R}$ to Algorithm 4, a family of BRE algorithms is obtained.

After the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ are determined, the cost-to-go function $\widetilde{J}_\mu(i)$ can be computed using Equations (3.38), (3.41), and (3.27):

$$\begin{aligned}
\widetilde{W}_\mu(i) &= \langle \underline{\Theta}, \underline{\Psi}(i) \rangle \\
&\Updownarrow \\
\widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\
&= \langle \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \underline{\Psi}(i') \ , \ \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \langle \underline{\Psi}(i'), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(\langle \underline{\Phi}(i'), \underline{\Phi}(i) \rangle - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} \langle \underline{\Phi}(j), \underline{\Phi}(i) \rangle \right) \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i', i) - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} k(j, i) \right)
\end{aligned} \tag{3.42}$$

Once $\widetilde{J}_\mu(\cdot)$ is found, policy evaluation is complete.

Note that Eq. (3.42) is identical to the functional form for $\widetilde{J}_\mu(\cdot)$ that was found in deriving BRE(SV) [Eq. (3.22)]. Although that derivation was carried out in a different way (by computing a dual optimization problem instead of explicitly constructing a map between $\mathcal{H}_k$ and $\mathcal{H}_\mathcal{K}$), the resulting cost-to-go functions have the same functional

form. The more general family of BRE algorithms developed in this section provides a deeper insight into how the BRE(SV) algorithm works.

To summarize, the generalized family of BRE algorithms allows any kernel-based regression method to be utilized to perform BRE and generate an approximate cost-to-go function. If the learned regression function $\widetilde{W}_\mu(\cdot)$ satisfies $\widetilde{W}_\mu(i) = g_i$ at some state i, then the Bellman residual of the corresponding cost-to-go function $\widetilde{J}_\mu(i)$, calculated using Eq. (3.42), is identically zero (as long as a nondegenerate kernel is used, as shown by Theorem 6).

3.3 BRE Using Gaussian Process Regression

The previous section showed how the problem of eliminating the Bellman residuals at the set of sample states $\widetilde{\mathcal{S}}$ is equivalent to a simple regression problem in $\mathcal{H}_\mathcal{K}$, and how to compute the corresponding cost-to-go function once the regression problem is solved. We now present BRE(GP), a BRE algorithm that uses Gaussian process regression to compute the solution to the regression problem in $\mathcal{H}_\mathcal{K}$. Like BRE(SV), BRE(GP) produces a cost-to-go solution whose Bellman residuals are zero at the sample states, and therefore reduces to exact policy iteration in the limit of sampling the entire state space (this will be proved later in the section). In addition, BRE(GP) can automatically select any free kernel parameters and provide natural error bounds on the cost-to-go solution, and therefore directly addresses the two important questions posed at the end of Section 3.1. As we will see, these desirable features of BRE(GP) arise because Gaussian process regression provides a tractable way to compute the posterior covariance and log marginal likelihood.

Pseudocode for BRE(GP) is shown in Algorithm 5. Similar to BRE(SV), BRE(GP) takes an initial policy μ_0, a set of sample states $\widetilde{\mathcal{S}}$, and a kernel k as input. However, it also takes a set of initial kernel parameters $\underline{\Omega} \in \mathbf{\Omega}$ (in BRE(SV), these parameters were assumed to be fixed). The kernel k, as well as its associated Bellman kernel $\mathcal{K}$ and the Gram matrix $\mathbb{K}$ all depend on these parameters, and to emphasize this dependence they are written as $k(i, i'; \underline{\Omega})$, $\mathcal{K}(i, i'; \underline{\Omega})$, and $\mathbb{K}(\underline{\Omega})$, respectively.

Algorithm 5 BRE(GP)

Input: $(\mu_0, \widetilde{\mathcal{S}}, k, \underline{\Omega})$
μ_0: initial policy
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel (covariance) function defined on $\mathcal{S}\times\mathcal{S}\times\mathbf{\Omega}$, $k(i,i';\underline{\Omega}) = \langle\underline{\Phi}(i;\underline{\Omega}), \underline{\Phi}(i';\underline{\Omega})\rangle$
$\underline{\Omega}$: initial set of kernel parameters
Begin
Define $\mathcal{K}(i,i';\underline{\Omega}) = \langle\underline{\Psi}(i;\underline{\Omega}), \underline{\Psi}(i';\underline{\Omega})\rangle$ {Define the associated Bellman kernel}
$\mu \leftarrow \mu_0$
loop
  Construct $\underline{g}^\mu$, the vector of stage costs $g_i^\mu \quad \forall i \in \widetilde{\mathcal{S}}$
  repeat
    Construct the Gram matrix $\mathbb{K}(\underline{\Omega})$, where $\mathbb{K}(\underline{\Omega})_{ii'} = \mathcal{K}(i,i';\underline{\Omega}) \quad \forall i,i' \in \widetilde{\mathcal{S}}$, using Eq. (3.17)
    Solve $\underline{\lambda} = \mathbb{K}(\underline{\Omega})^{-1}\underline{g}^\mu$
    Calculate the gradient of the log marginal likelihood, $\nabla_{\underline{\Omega}} \log p(\underline{g}^\mu|\widetilde{\mathcal{S}}, \underline{\Omega})$, where

$$\frac{\partial \log p(\underline{g}^\mu|\widetilde{\mathcal{S}}, \underline{\Omega})}{\partial \Omega_j} = \frac{1}{2}\mathrm{tr}\left((\underline{\lambda}\underline{\lambda}^T - \mathbb{K}(\underline{\Omega})^{-1})\frac{\partial \mathbb{K}(\underline{\Omega})}{\partial \Omega_j}\right).$$

    Update the kernel parameters using any gradient-based optimization rule:

$$\underline{\Omega} \leftarrow \underline{\Omega} + \gamma\nabla_{\underline{\Omega}} \log p(\underline{g}^\mu|\widetilde{\mathcal{S}}, \underline{\Omega}),$$

    where γ is an appropriately selected step size
  until stopping condition for gradient-based optimization rule is met
  Using the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ and kernel parameters $\underline{\Omega}$, compute the cost-to-go function

$$\widetilde{J}_\mu(i) = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i',i;\underline{\Omega}) - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu k(j,i;\underline{\Omega})\right)$$

  {Policy evaluation step complete}
  Compute $E(i)$, the 1-σ error bound on the Bellman residual function

$$E(i) = \sqrt{\mathcal{K}(i,i;\underline{\Omega}) - \underline{h}^T\mathbb{K}(\underline{\Omega})^{-1}\underline{h}}$$

  where $h_j \equiv \mathcal{K}(i,j;\underline{\Omega}), \quad j \in \widetilde{\mathcal{S}}$
  $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u)\left(g(i,u) + \alpha\widetilde{J}_\mu(j)\right)$ {Policy improvement}
end loop
End

The algorithm consists of two nested loops. The inner loop (lines 11–16) is responsible for repeatedly solving the regression problem in $\mathcal{H}_\mathcal{K}$ (notice that the target values of the regression problem, the one-stage costs $\underline{g}^\mu$, are computed in line 10) and adjusting the kernel parameters using gradient-based optimization. This process is carried out with the policy fixed, so the kernel parameters are tuned to each policy prior to the policy improvement stage being carried out. Line 13 computes the $\underline{\lambda}$ values necessary to compute the mean function of the posterior process [Eq. (2.21)]. Lines 14 and 15 then compute the gradient of the log likelihood function, using Eq. (2.23), and use this gradient information to update the kernel parameters. This process continues until a maximum of the log likelihood function has been found.

Once the kernel parameters are optimally adjusted for the current policy μ, the main body of the outer loop (lines 17-19) performs three important tasks. First, it computes the cost-to-go solution $\widetilde{J}_\mu(i)$ using Eq. (3.42) (rewritten on line 17 to emphasize dependence on $\underline{\Omega}$). Second, on line 18, it computes the posterior standard deviation $E(i)$ of the Bellman residual function. This quantity is computed using the standard result from Gaussian process regression for computing the posterior variance [Eq. (2.20)], and it gives a Bayesian error bound on the magnitude of the Bellman residual $BR(i)$. This error bound is useful, of course, because the goal is to achieve small Bellman residuals at as many states as possible. Finally, the algorithm carries out a policy improvement step in Line 19.

Note that the log likelihood function is not necessarily guaranteed to be convex, so it may be possible for the gradient optimization carried out in the inner loop (lines 11-16) to converge to a local (but not global) maximum. However, by initializing the optimization process at several starting points, it may be possible to decrease the probability of ending up in a local maximum [128, 5.4].

The following theorems establish the same important properties of BRE(GP) that were proved earlier for BRE(SV):

Theorem 7. *Assume that the kernel* $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ *is nondegenerate.*

Then the cost-to-go functions $\widetilde{J}_\mu(i)$ computed by BRE(GP) (on line 17) satisfy

$$\widetilde{J}_\mu(i) = g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \qquad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ are identically zero at every state $i \in \widetilde{\mathcal{S}}$.

An immediate corollary of Theorem 7 follows:

Corollary 8. *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by BRE(GP) satisfies*

$$\widetilde{J}_\mu(i) = J_\mu(i) \qquad \forall i \in \mathcal{S}.$$

That is, the cost-to-go function $\widetilde{J}_\mu(i)$ is exact.

Corollary 9. *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then BRE(GP) is equivalent to exact policy iteration.*

3.3.1 Computational Complexity

The computational complexity of running BRE(GP) is dominated by steps 12 (constructing the Gram matrix) and 13 (inverting the Gram matrix). To obtain an expression for the computational complexity of the algorithm, first define $n_s \equiv |\widetilde{\mathcal{S}}|$ as the number of sample states. Furthermore, define the average branching factor of the MDP, β, as the average number of possible successor states for any state $i \in \mathcal{S}$, or equivalently, as the average number of terms P_{ij}^μ that are nonzero for fixed i and μ. Finally, recall that since $\mathcal{K}(i, i')$ is nondegenerate (as proved earlier), the Gram matrix is positive definite and symmetric.

Now, each entry of the Gram matrix $\mathbb{K}(\underline{\Omega})_{ii'}$ is computed using Eq. (3.17). Using the average branching factor defined above, computing a single element of the Gram matrix using Eq. (3.17) therefore requires $\mathcal{O}(\beta^2)$ operations. Since the Gram matrix is symmetric and of dimension $n_s \times n_s$, there are $n_s(n_s + 1)/2$ unique elements that

must be computed. Therefore, the total number of operations to compute the full Gram matrix is $\mathcal{O}(\beta^2 n_s(n_s+1)/2)$.

Once the Gram matrix is constructed, its inverse must be computed in line 13. Since the Gram matrix is positive definite and symmetric, Cholesky decomposition can be used, resulting in a total complexity of $\mathcal{O}(n_s^3)$ for line 13. (The results of the Cholesky decomposition computed in line 13 can and should be saved to speed up the calculation of the log marginal likelihood in line 14, which also requires the inverse of the Gram matrix). As a result, the total complexity of the BRE(GP) algorithm is

$$\mathcal{O}(n_s^3 + \beta^2 n_s(n_s+1)/2). \tag{3.43}$$

This complexity can be compared with the complexity of exact policy iteration. Recall that exact policy iteration requires inverting the matrix $(I - \alpha P^\mu)$ in order to evaluate the policy μ:

$$\underline{J}_\mu = (I - \alpha P^\mu)^{-1} \underline{g}^\mu.$$

$(I - \alpha P^\mu)$ is of dimension $N \times N$, where $N = |\mathcal{S}|$ is the size of the state space. Therefore, the complexity of exact policy iteration is

$$\mathcal{O}(N^3). \tag{3.44}$$

Comparing Eqs. (3.43) and (3.44), notice that both expressions involve cubic terms in n_s and N, respectively. By assumption, the number of sample states is taken to be much smaller than the total size of the state space ($n_s \ll N$). Furthermore, the average branching factor β is typically also much smaller than N (and can certainly be no larger than N). Therefore, the computational complexity of BRE(GP) is significantly less than the computational complexity of exact policy iteration.

3.3.2 Computational Results - Mountain Car

BRE(GP) was implemented on the same mountain car problem examined earlier for BRE(SV). The kernel function employed for these experiments was

$$k((x_1, \dot{x}_1), (x_2, \dot{x}_2); \underline{\Omega}) = \exp\left(-(x_1 - x_2)^2/\Omega_1^2 - (\dot{x}_1 - \dot{x}_2)^2/\Omega_2^2\right).$$

This is the same functional form of the kernel that was used for BRE(SV). However, the length-scales Ω_1 and Ω_2 were left as free kernel parameters to be automatically learned by the BRE(GP) algorithm, in contrast with the BRE(SV) tests, where these values had to be specified by hand. The parameters were initially set at $\Omega_1 = \Omega_2 = 10$. These values were deliberately chosen to be far from the empirically chosen values used in the BRE(SV) tests, which were $\Omega_1 = 0.25$ and $\Omega_2 = 0.40$. Therefore, the test reflected a realistic situation in which the parameter values were not initially well-known.

Initially, the same 9×9 grid of sample states as used in the BRE(SV) tests was employed to compare the performance of the two algorithms:

$$\widetilde{\mathcal{S}} = \{(x, \dot{x}) \mid x = -1.0, -0.75, \ldots, 0.75, 1.0$$
$$\dot{x} = -2.0, -1.5, \ldots, 1.5, 2.0\}.$$

BRE(GP) was then executed. Like the BRE(SV) test, the algorithm converged after 3 policy updates. The associated cost-to-go functions (after each round of kernel parameter optimization) are shown in Figure 3-5 along with the true optimal cost-to-go for comparison. The final parameter values found by BRE(GP) were $\Omega_1 = 0.253$ and $\Omega_2 = 0.572$. These values are similar to those found empirically in the BRE(SV) tests but are actually better values, as evidenced by the system response under the policy found by BRE(GP), which is shown in Figure 3-6. The BRE(GP) policy arrives in the parking area at time $t = 15$, 2 time steps faster than the BRE(SV) policy and only 1 step step slower than the optimal policy. This result demonstrates that BRE(GP) is not only able to learn appropriate values for initially unknown

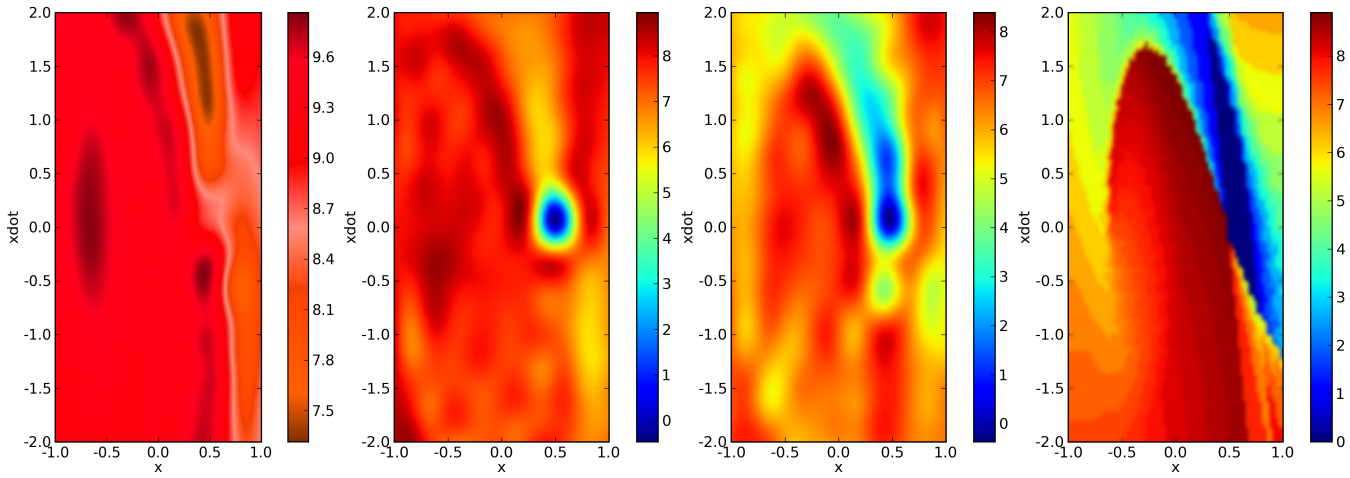


Figure 3-5: From left to right: Approximate cost-to-go computed by BRE(GP) for iterations 1, 2 , and 3; exact cost-to-go computed using value iteration.

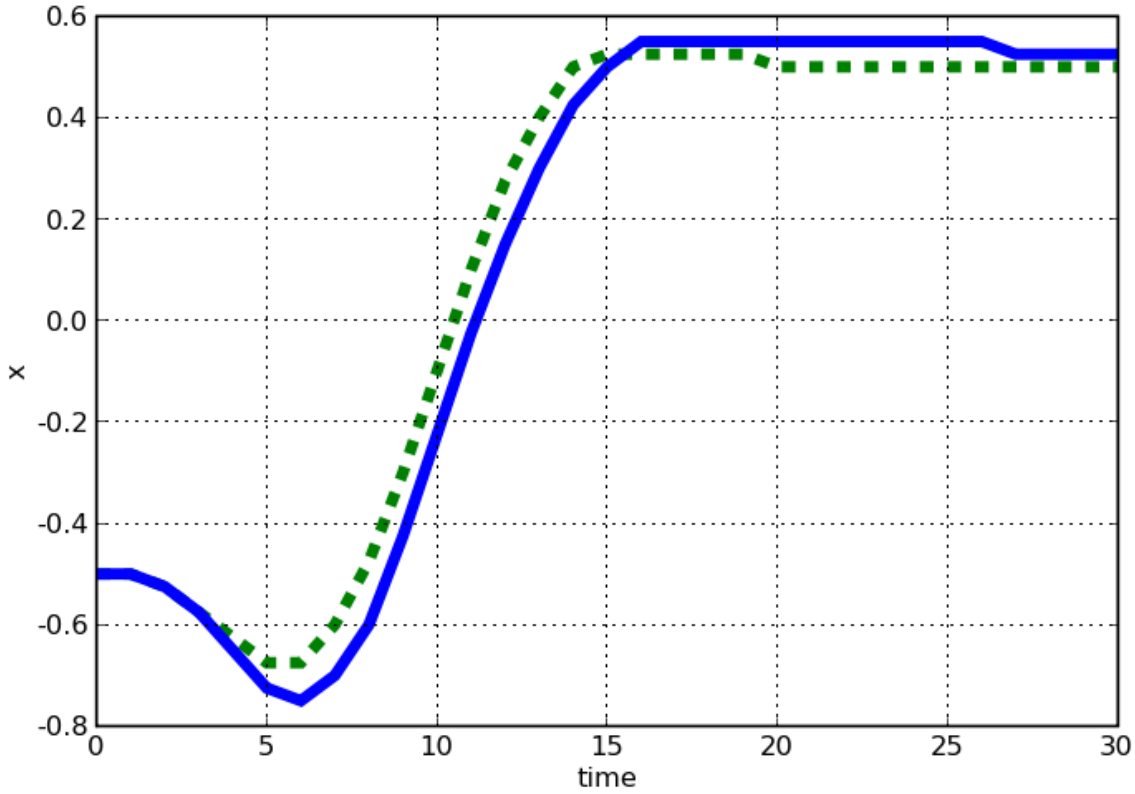


Figure 3-6: System response under the optimal policy (dashed line) and the policy computed by BRE(GP) after 3 iterations (solid line).

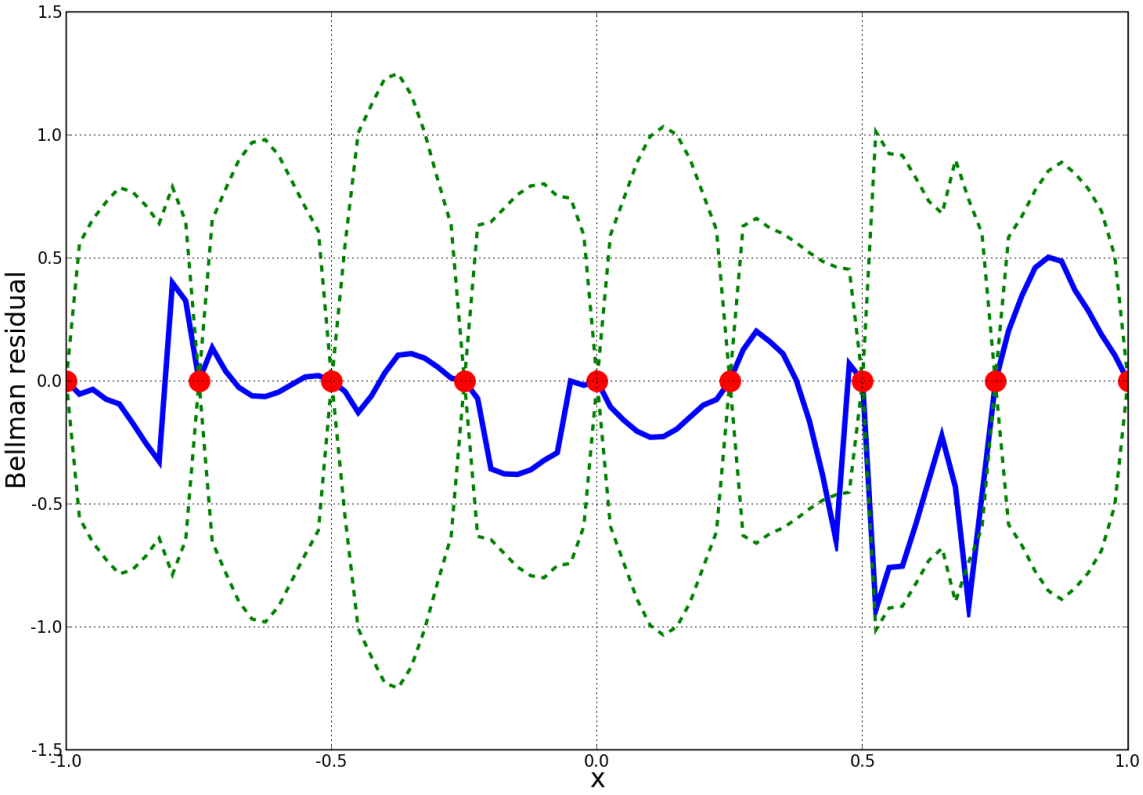


Figure 3-7: Bellman residuals (solid blue line) as a function of x ($\dot{x}$ is fixed to zero in this plot). Sample states are represented by the red dots; notice that the Bellman residuals at the sample states are exactly zero as expected. 2σ error bounds computed by BRE(GP) are shown by the dashed green lines.

kernel parameters, but also that the resulting policy can be of higher quality than could be obtained using time-consuming empirical testing.

As discussed, in addition to automatically learning the kernel parameters, BRE(GP) also computes error bounds on the cost-to-go solution. Figure 3-7 plots the Bellman residuals $BR(i)$, as well as the 2σ error bounds $2E(i)$ computed in line 18 of the BRE(GP) algorithm. For clarity, the plot is given as a function of x only; $\dot{x}$ was fixed to zero. The sampled states

$$\{(x, 0) \mid x = -1.0, -0.75, \ldots, 0.75, 1.0\}$$

are represented by red dots in the figure. Notice that, as expected, the Bellman residuals at the sampled states are exactly zero. Furthermore, the error bounds

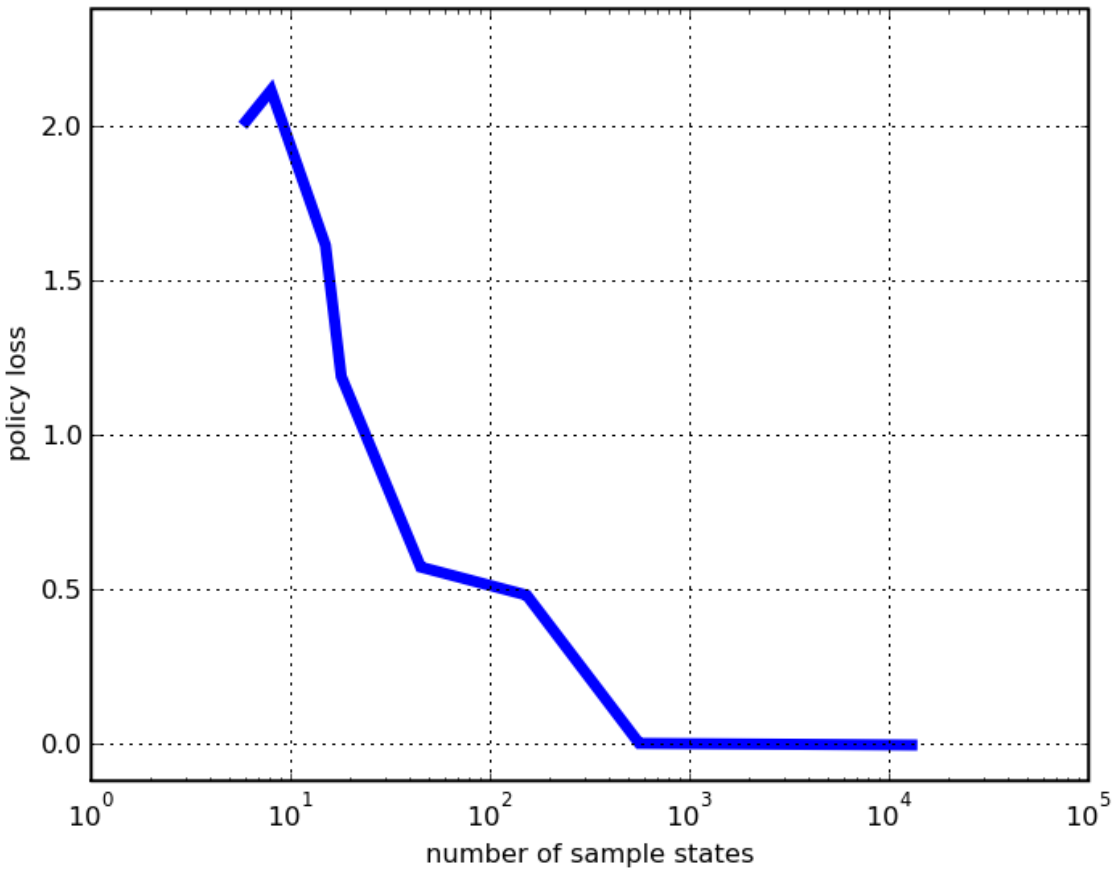


Figure 3-8: Policy loss vs. number of sampled states for the BRE(GP) algorithm. The total size of the state space is 13,041 (1.3041×10^4). Note that BRE(GP) finds the optimal policy (i.e. zero policy loss) well before the entire state space is sampled.

computed by BRE(GP), shown as the dashed green lines, do a good job of bounding the Bellman residuals at non-sampled states. Recall that these bounds represent the posterior variance of the Bellman residuals. Thus, for the 2σ case shown in the figure, the bounds represent a 95% confidence interval, so we expect that 95% of the Bellman residuals should lie within the bounds. There are two cases where the residuals lie outside the bounds (at $x = 0.45$ and $x = 0.70$), and there are a total of 81 points in the graph. This leads to an empirical confidence interval of

$$\frac{81 - 2}{81} = 97.53\%,$$

which is in excellent agreement with the bounds computed by BRE(GP).

A further series of tests were conducted to explore the quality of the policies produced by BRE(GP) as a function of the number of sample states. In these tests, a uniform sampling of the two-dimensional state space was used, and tests were conducted for as few as 8 sample states and as many as 13,041 (which was the total size of the state space, so the entire space was sampled in this test). For each test, the cost-to-go of the resulting policy, starting at the initial condition ($x = -0.5, \dot{x} = 0.0$), was compared with the cost-to-go of the optimal policy. Figure 3-8 plots the difference between these two costs (this value is called the *policy loss*) as a function of n_s, the number of sample states. Note that for this example problem, Corollary 9 guarantees that the policy loss must go to zero as n_s goes to 13,041, since in this limit the entire state space is sampled and BRE(GP) reduces to exact policy iteration, which always converges to the optimal policy. Figure 3-8 confirms this prediction, and in fact demonstrates a stronger result: the policy loss goes to zero well *before* the entire space is sampled. While this is behavior is not guaranteed to always occur, it does suggest that there may be a "critical number" of sample states, above which BRE(GP) yields the optimal policy.

3.4 Kernel Selection

The kernel $k(i, i')$ plays an important role in the BRE algorithms presented in the preceding chapters. Since the algorithms learn a cost-to-go function $\widetilde{J}_\mu(i)$ which is an element of the RKHS $\mathcal{H}_k$, the kernel $k(i, i')$ implicitly determines the structure of the cost-to-go function that is ultimately produced by the algorithms. Intuitively, $k(i, i')$ can be interpreted as a similarity measure that encodes our assumptions about how strongly correlated the cost-to-go values of states i and i' are.

In order to guarantee that the Bellman residuals are identically zero at the sample states, it is important that $k(i, i')$ be nondegenerate. Functional forms for a variety of nondegenerate kernels are known; Table 3.1 lists several of these. Note that all of these kernels contain a number of free parameters, such as the positive semidefinite scaling matrix Σ, that can be automatically optimized by the BRE(GP) algorithm.

Kernel Name	Functional Form
Radial basis function	$k(i,i') = \exp\left(-(i-i')^T\Sigma(i-i')\right)$
Exponential	$k(i,i') = \exp\left(-\sqrt{(i-i')^T\Sigma(i-i')}\right)$
γ-exponential	$k(i,i') = \exp\left(-((i-i')^T\Sigma(i-i'))^\gamma\right)$
Matérn	$k(i,i') = \frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}(i-i')^T\Sigma(i-i')\right)^\nu K_\nu\left(\frac{\sqrt{2\nu}}{l}(i-i')^T\Sigma(i-i')\right)$
Rational quadratic	$k(i,i') = \left(1+(i-i')^T\Sigma(i-i')/(2\alpha)\right)^{-\alpha}$
Neural network	$k(i,i') = \sin^{-1}\left(\frac{2i^T\Sigma i'}{\sqrt{(1+2i^T\Sigma i')(1+2i'^T\Sigma i)}}\right)$

Table 3.1: Nondegenerate kernel function examples [128, Sec. 4.2.3]

So-called non-stationary kernels, which include features such as characteristic length-scales that vary over the state space, can also be accomodated [128, Sec. 4.2.3]. Furthermore, it is well-known that new, "composite" kernels can be constructed by taking linear combinations [128, Sec. 4.2.4]: if $k_1(i,i')$ and $k_2(i,i')$ are kernels, and Ω_1 and Ω_2 are positive weights, then

$$k_3(i,i') = \Omega_1 k_1(i,i') + \Omega_2 k_2(i,i')$$

is also a valid kernel. Again, BRE(GP) can be used to learn Ω_1 and Ω_2, automatically determining an appropriate weighting for the problem at hand.

All of the kernels presented in Table 3.1 can be described as "geometric" in the sense that they compute a dot product—either of the form $(i-i')^T\Sigma(i-i')$ or $i^T\Sigma i'$—in the state space. This dot product encodes a notion of distance or proximity between states in the state space. Thus, these geometric kernels are appropriate for many problems, such as the mountain car problem used as the example in this chapter, where a natural distance metric can be defined on the state space. For problems that do not admit a natural distance metric, another approach may be taken. An example of such a problem is the two-room robot navigation problem shown in Figure 3-9. In this problem, a robot must navigate to the goal location, shown in green, by moving between adjacent points on the two-dimensional grid. The robot cannot move through the vertical wall in the center of the grid. Because of the presence of the wall, a kernel

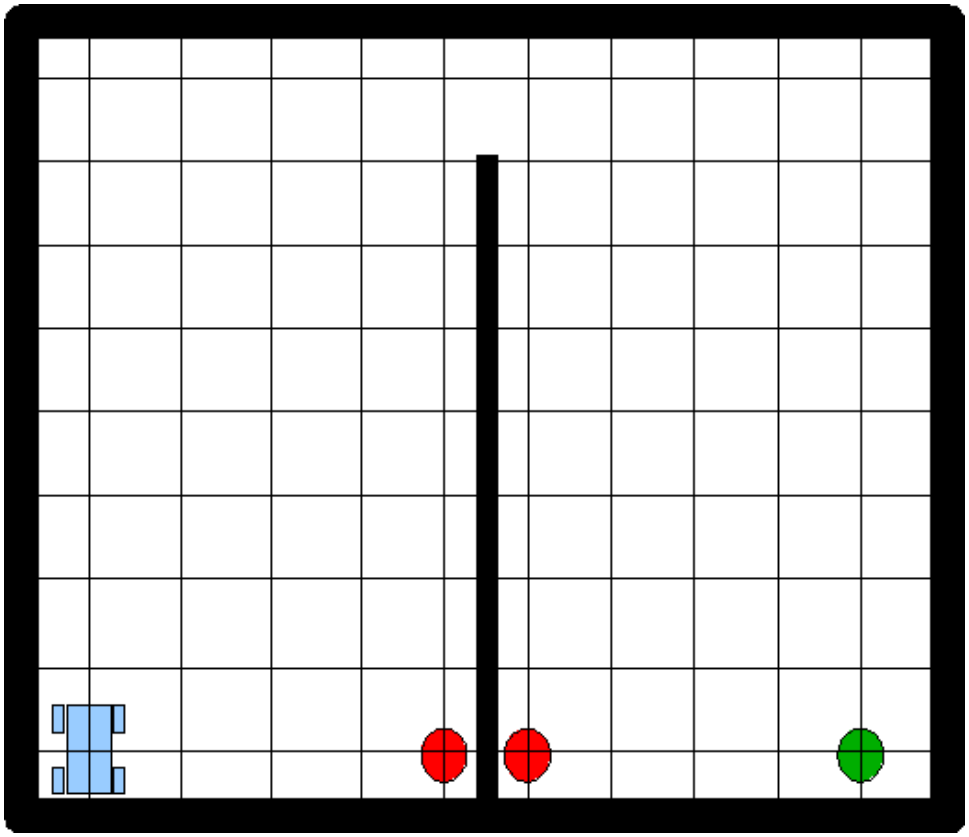


Figure 3-9: Two-room robot navigation problem.

based on a normal Euclidean distance metric of the form $\sqrt{(i-i')^T\Sigma(i-i')}$ (where i and i' are two-dimensional position vectors) would work poorly, since it would tend to predict high correlation in the cost-to-go between two states on opposite sides of the wall (shown in red). However, since all feasible paths between these two highlighted states are long, the cost-to-go values of these two states are in reality very weakly correlated. To solve this problem, a new type of kernel function can be constructed that relies on exploiting the graph structure of the state space. More precisely, for a fixed policy, the state transition dynamics of the MDP are described by a Markov chain, where each state i is represented as a node in the graph and is connected to states that are reachable from i in one step with positive probability. Instead of using a distance metric based on the geometry of the problem, one can define a distance metric based on the graph which measures the number of steps necessary to move from one state to another. A number of authors have shown that these graph-

based methods give rise to kernel functions that naturally capture the structure of the cost-to-go function and have demonstrated good results in applications like the two-room robot navigation problem [16, 17, 102, 103, 146, 150, 151]. Note, however, that some of these methods require carrying out dynamic programming on the state space in order to find the shortest paths between states. In order to be able to do this efficiently, some of the above methods require specialized structure in the state space or other assumptions that limit their applicability to general MDPs. This issue will be discussed further in Chapter 4, where we shall show how the multi-stage extension of the BRE approach can automatically construct a kernel that captures the local graph structure of the state space for general MDPs.

To this point in the discussion, we have been assuming that the underlying state space, upon which the kernel is defined, is a subset of $\mathbb{R}^n$; that is, a normal Euclidean space. However, it is worth noting that kernels can be defined over more structured spaces in addition to Euclidean spaces. For example, researchers have investigated defining kernels over strings (sequences of characters from a well-defined alphabet) for purposes such as text classification [96, 97] and protein sequence analysis [95, 125, 126]. Kernels defined over tree structures have also been investigated [164, 166]. These kernels could be useful in the ADP setting if one was dealing with an MDP whose states were naturally represented as structured objects such as strings or trees.

Kernels can also be explicitly constructed from a pre-existing feature mapping $\underline{\Phi}(\cdot)$, if an appropriate set of features is already known for the application at hand. This may be the case for a number of well-studied problems such as chess [144], checkers [136], and tetris [24, 155], where a standard set of features for the problem has been established; or for problems where intuition or previous experience can be used to construct a reasonable feature set. In these cases, a kernel based on the features is constructed simply by computing the inner product between feature vectors: $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$. In applying the BRE algorithms, however, care must be taken to ensure that the kernel is non-degenerate. Since using a finite set of features will result in a degenerate kernel (whose rank is equal to the number of features used, assuming linear independence between features), an additional term

must be added to the kernel to make it non-degenerate. One possible way to do this is to add a small Kronecker delta term to the kernel: $\eta\delta(i, i') + k(i, i')$, where $\eta > 0$ is a weighting factor that may be interpreted as a type of regularization. This approach is equivalent to adding a small multiple of the identity matrix to the Gram matrix $\mathbb{K}$ computed by BRE, and functions to ensure that the Gram matrix is full-rank and invertible, as required by the BRE algorithm. Initial experiments in applying the BRE algorithms to the problem of tetris, using a kernel constructed from a standard set of 22 basis functions (see, for example, [24]), have shown the importance of including the regularization term in order to ensure numerical stability of the algorithm. However, if a good set of (a finite number of) features is already known, other ADP approaches that assume a finite-dimensional cost approximation architecture may be a more appropriate choice than BRE, which assumes an infinite-dimensional architecture.

The BRE algorithms can make use of geometric kernels, graph-based kernels, kernels defined over structured objects, kernels constructed using pre-existing features, or even linear combinations of many different kernel types, depending which type(s) are most appropriate for the application.

3.5 Summary

This chapter has demonstrated how kernel-based techniques can be used to design a new class of approximate dynamic programming (ADP) algorithms that carry out Bellman residual elimination (BRE). These BRE algorithms learn the structure of the cost-to-go function by forcing the Bellman residuals to zero at a chosen set of representative sample states. Essentially, our approach reduces the dimensionality of the full fixed-policy Bellman equation to a much smaller linear system that is practical to solve, and uses the solution to the smaller system in order to infer the cost-to-go everywhere in the state space.

The BRE approach has a number of advantages and desirable features, which we summarize here:

- The Bellman residuals are exactly zero at the sample states $\widetilde{\mathcal{S}}$. By exploiting

knowledge of the underlying MDP model, the BRE algorithms avoid the need to carry out trajectory simulations and thus do not suffer from simulation noise effects. Since there is no simulation noise, there is no danger of overfitting by forcing the Bellman residuals to zero.

- The BRE algorithms converge to the optimal policy in the limit of sampling the entire state space ($\widetilde{\mathcal{S}} \rightarrow \mathcal{S}$). This property is a direct consequence of the fact that the Bellman residuals at the sample states are exactly zero. Note that this feature of BRE depends crucially on the use of non-degenerate kernels that effectively provide an infinite-dimensional feature space; other ADP algorithms based on finite-dimensional function approximation architectures cannot in general assure convergence to the optimal policy, since in general, the optimal cost-to-go function cannot be represented exactly in the chosen architecture.

- The BRE algorithm based on Gaussian process regression, denoted BRE(GP), provides error bounds that can be used for adaptive re-sampling of the state space. BRE(GP) can also automatically learn any free hyperparameters in the kernel.

- The computational requirements of carrying out BRE scale with the number of sample states chosen, which is under the designer's control.

- The BRE algorithms are naturally distributable, allowing for the running time of the algorithm to be reduced by running it on a cluster of multiple processors. A distributed implementation of BRE is described in detail in Section 7.1.3.

- In many other ADP algorithms based on linear cost-to-go approximation architectures (such as LSPI [93], LSPE [23, 114], LSTD [39, 40], etc), the feature vectors enter into the algorithm as *outer products* of the form $\underline{\Phi}(i)\underline{\Phi}(i')^T$. This approach works well when the number of features is small, but if it is desirable to use a large number of features, difficulties are encountered since computing the outer product becomes computationally expensive. In contrast, by working in the dual space, the BRE algorithm computes only *inner products* of the

feature vectors, which can be computed efficiently using kernels even when the effective number of features is very large or even infinite. Thus, BRE allows the use of very high-dimensional feature vectors in a computationally tractable way.

Application of our BRE algorithms to a classic reinforcement learning problem indicates that they yield nearly optimal policies while requiring few sample states (and therefore, can be computed efficiently). The kernel parameter learning mechanism employed by BRE(GP) was demonstrated to be effective at finding appropriate values of the kernel parameters. This is an important feature of the BRE(GP) algorithm, especially in higher-dimensional and more complex problems where suitable values of the kernel parameters may not be known a priori. In later chapters, we will demonstrate the application of BRE to larger and more complex problems dealing with autonomous planning in multi-UAV systems.

Appendix: Proofs

This appendix gives proofs for the lemmas, theorems, and corollaries presented in this chapter.

Lemma 1 *Assume the vectors* $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$ *are linearly independent. Then the vectors* $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$*, where* $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j)$*, are also linearly independent.*

Proof. Consider the real vector space $\mathcal{V}$ spanned by the vectors $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$. It is clear that $\underline{\Psi}(i)$ is a linear combination of vectors in $\mathcal{V}$, so a linear operator $\hat{B}$ that maps $\underline{\Phi}(i)$ to $\underline{\Psi}(i)$ can be defined:

$$\underline{\Psi}(i) = \hat{B}\underline{\Phi}(i).$$

Since

$$\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j),$$

the matrix of $\hat{B}$ is

$$(I - \alpha P^{\mu}),$$

where I is the identity matrix and P^{μ} is the probability transition matrix for the policy μ. Since P^{μ} is a stochastic matrix, its largest eigenvalue is 1 and all other eigenvalues have absolute value less than 1; hence all eigenvalues of αP^{μ} have absolute value less than or equal to $\alpha < 1$. Since all eigenvalues of I are equal to 1, $(I - \alpha P^{\mu})$ is full rank and $dim(ker(\hat{B})) = 0$. Therefore,

$$dim(\{\underline{\Psi}(i) \,|\, i \in \mathcal{S}\}) = dim(\{\underline{\Phi}(i) \,|\, i \in \mathcal{S}\}) = n_s$$

so the vectors $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$ are linearly independent. □

Theorem 2 *Assume that the kernel* $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ *is nondegenerate. Then the associated Bellman kernel defined by* $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$*, where* $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j)$*, is also nondegenerate.*

Proof. Since $k(i,i')$ is nondegenerate, the vectors $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$ are linearly independent. Therefore, Lemma 1 applies, and the vectors $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$ are linearly independent, immediately implying that $\mathcal{K}(i,i')$ is nondegenerate. □

Theorem 3 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \qquad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ are identically zero at every state $i \in \widetilde{\mathcal{S}}$.

Proof. Since $k(i,i')$ is nondegenerate, Theorem 2 applies and $\mathcal{K}(i,i')$ is also nondegenerate. Therefore, the Gram matrix $\mathbb{K}$ of $\mathcal{K}(i,i')$ is positive definite and invertible. It follows that, in Line 8 of Algorithm 2, there is a unique solution for $\underline{\lambda}$. Having found a feasible solution $\underline{\lambda}$ for the dual problem given by [Eq. (3.23)], Slater's condition is satisfied and strong duality holds. Therefore, the primal problem [Eq. (3.19)] is feasible, and its optimal solution is given by Eq. (3.21). Feasibility of this solution implies

$$BR(i) = -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = 0 \quad \forall i \in \widetilde{\mathcal{S}}$$

as claimed. □

Corollary 4 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = J_\mu(i) \quad \forall i \in \mathcal{S}.$$

That is, the cost-to-go function $\widetilde{J}_\mu(i)$ is exact.

Proof. Applying Theorem 3 with $\widetilde{\mathcal{S}} = \mathcal{S}$, we have

$$BR(i) = 0 \quad \forall i \in \mathcal{S}.$$

Using the definition of the Bellman residual,

$$BR(i) \equiv \widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i),$$

immediately implies that

$$\widetilde{J}_\mu(i) = T_\mu \widetilde{J}_\mu(i) \quad \forall i \in \mathcal{S}.$$

Therefore, $\widetilde{J}_\mu(i)$ satisfies the fixed-policy Bellman equation, so

$$\widetilde{J}_\mu(i) = J_\mu(i) \quad \forall i \in \mathcal{S}$$

as claimed. □

Corollary 5 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then BRE(SV) (given in Algorithm 3) is equivalent to exact policy iteration.*

Proof. By Corollary 4, the cost-to-go function $\widetilde{J}_\mu(i)$ produced by BRE(SV) is equal to the exact cost-to-go $J_\mu(i)$, at every state $i \in \mathcal{S}$. Since the policy improvement step (Line 9) is also exact, BRE(SV) carries out exact policy iteration by definition, and converges in a finite number of steps to the optimal policy. □

Theorem 6 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then there exists a linear, invertible mapping $\hat{A}$ from $\mathcal{H}_k$, the RKHS of k, to $\mathcal{H}_\mathcal{K}$, the RKHS of the associated Bellman kernel $\mathcal{K}$:*

$$\begin{aligned} \hat{A}: \qquad & \mathcal{H}_k \to \mathcal{H}_\mathcal{K} \\ & \hat{A}\widetilde{J}_\mu(\cdot) = \widetilde{W}_\mu(\cdot). \end{aligned}$$

Furthermore, if an element $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ is given by

$$\widetilde{J}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle,$$

then the corresponding element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ *is given by*

$$\widetilde{W}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle, \tag{3.45}$$

and vice versa:

$$\begin{aligned} \widetilde{J}_\mu(\cdot) &= \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ &\Updownarrow \\ \widetilde{W}_\mu(\cdot) &= \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle. \end{aligned}$$

Proof. Recall from the proof of Lemma 1 that the relation between $\underline{\Phi}(i)$ and $\underline{\Psi}(i)$ [Eq. (3.27)] is defined by the invertible linear operator $\hat{B}$:

$$\underline{\Psi}(i) = \hat{B}\underline{\Phi}(i)$$

Using the definition of $\hat{B}$ and linearity of the inner product $\langle \cdot, \cdot \rangle$, Eq. (3.45) can be rewritten as

$$\begin{aligned} \widetilde{W}_\mu(\cdot) &= \langle \underline{\Theta}, \hat{B}\underline{\Phi}(\cdot) \rangle \\ &= \hat{B}\langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ &= \hat{B}\widetilde{J}_\mu(\cdot), \end{aligned}$$

which establishes the invertible linear mapping between $\mathcal{H}_k$ and $\mathcal{H}_\mathcal{K}$, as claimed. Therefore, the linear mapping $\hat{A}$ in the statement of the theorem exists and is equal to the linear mapping $\hat{B}$ from Lemma 1:

$$\hat{A} = \hat{B}.$$

Comparing Eqs. (3.31) and (3.36), notice that we can convert between a cost-to-go function $\widetilde{J}_\mu(\cdot)$ and its corresponding function $\widetilde{W}_\mu(\cdot)$ by replacing $\underline{\Phi}(\cdot)$ with $\underline{\Psi}(\cdot)$ in the

inner product:

$$\begin{aligned} \widetilde{J}_\mu(\cdot) &= \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ &\Updownarrow \\ \widetilde{W}_\mu(\cdot) &= \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle \end{aligned}$$

□

Theorem 7 *Assume that the kernel* $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ *is nondegenerate. Then the cost-to-go functions* $\widetilde{J}_\mu(i)$ *computed by BRE(GP) (on line 17) satisfy*

$$\widetilde{J}_\mu(i) = g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \qquad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ *are identically zero at every state* $i \in \widetilde{\mathcal{S}}$.

Proof. Note that line 13 of the BRE(GP) algorithm computes the weights $\underline{\lambda}$ as

$$\underline{\lambda} = \mathbb{K}(\underline{\Omega})^{-1} \underline{g}^\mu.$$

Aside from the purely notational difference emphasizing the dependence of the Gram matrix $\mathbb{K}(\underline{\Omega})$ on the kernel parameters $\underline{\Omega}$, the weights $\underline{\lambda}$ and the cost-to-go function $\widetilde{J}_\mu(i)$ computed by BRE(GP) are identical to those computed by BRE(SV). Therefore, Theorem 3 applies directly, and the Bellman residuals are identically zero at every state $i \in \widetilde{\mathcal{S}}$. □

Corollary 8 *Assume that the kernel* $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ *is nondegenerate, and that* $\widetilde{\mathcal{S}} = \mathcal{S}$. *Then the cost-to-go function* $\widetilde{J}_\mu(i)$ *produced by BRE(GP) satisfies*

$$\widetilde{J}_\mu(i) = J_\mu(i) \qquad \forall i \in \mathcal{S}.$$

That is, the cost-to-go function $\widetilde{J}_\mu(i)$ *is exact.*

Proof. Applying Theorem 7 with $\widetilde{\mathcal{S}} = \mathcal{S}$, we have

$$BR(i) = 0 \quad \forall i \in \mathcal{S}.$$

Using the definition of the Bellman residual,

$$BR(i) \equiv \widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i),$$

immediately implies that

$$\widetilde{J}_\mu(i) = T_\mu \widetilde{J}_\mu(i) \quad \forall i \in \mathcal{S}.$$

Therefore, $\widetilde{J}_\mu(i)$ satisfies the fixed-policy Bellman equation, so

$$\widetilde{J}_\mu(i) = J_\mu(i) \quad \forall i \in \mathcal{S}$$

as claimed. □

Corollary 9 *Assume that the kernel* $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ *is nondegenerate, and that* $\widetilde{\mathcal{S}} = \mathcal{S}$*. Then BRE(GP) is equivalent to exact policy iteration.*

Proof. By Corollary 4, we have that the cost-to-go produced by BRE(GP), $\widetilde{J}_\mu(i)$ is equal to the exact cost-to-go $J_\mu(i)$, at every state $i \in \mathcal{S}$. Since the policy improvement step (Line 19) is also exact, the algorithm carries out exact policy iteration by definition, and converges in a finite number of steps to the optimal policy. □

Chapter 4

Multi-Stage and Model-Free Bellman Residual Elimination

The previous chapter developed the basic ideas behind the Bellman residual elimination approach, under the assumption that the system model is known. In this chapter, we show how the basic, model-based approach can be extended in several ways. First, a multi-stage extension is discussed, in which Bellman residuals of the form $|\widetilde{J}_\mu(i) - T_\mu^n \widetilde{J}_\mu(i)|$ are eliminated (the algorithms in the previous chapter correspond to the special case where n, the number of stages, is equal to one). Second, a model-free BRE variant is developed, in which knowledge of the system model is not required.

4.1 Multi-Stage Bellman Residual Elimination

Before developing the multi-stage extension to BRE, it is useful to examine the associated Bellman kernel $\mathcal{K}(i, i')$ in some detail to understand its structure. Eq. (3.28) shows that $\mathcal{K}(i, i')$ can be viewed as the inner product between the feature mappings $\underline{\Psi}(i)$ and $\underline{\Psi}(i')$ of the input states i and i', respectively. In turn, Eq. (3.8) shows that $\underline{\Psi}(i)$ represents a new feature mapping that takes into account the local graph structure of the MDP, since it is a linear combination of $\underline{\Phi}$ features of both the state i and all of the successor states j that can be reached in a single step from i (these

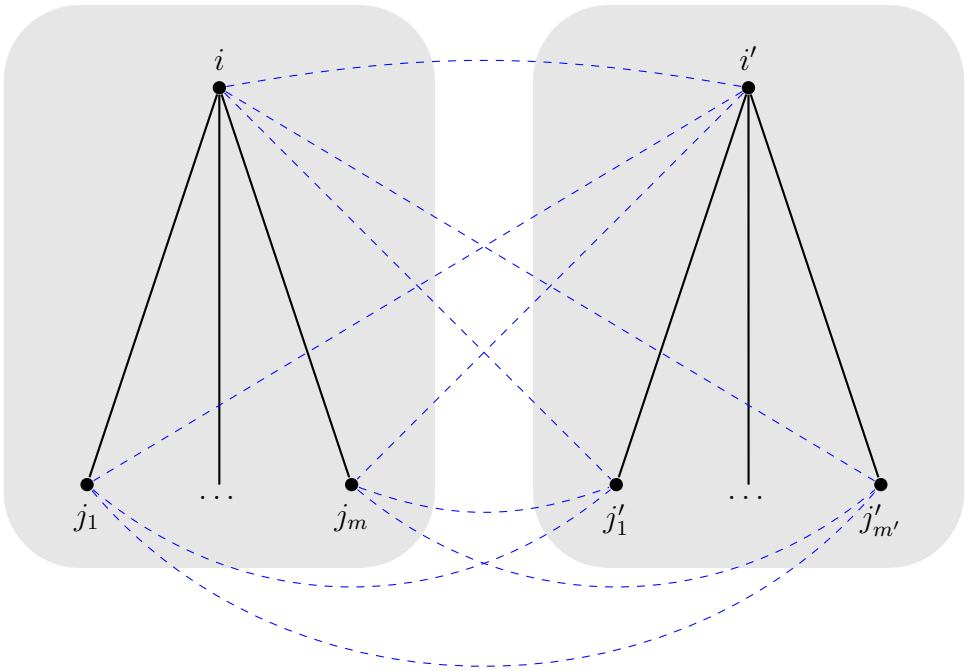


Figure 4-1: Graph structure of the associated Bellman kernel $\mathcal{K}(i, i')$ [Eq. (3.28)]. The one-stage successor states of i and i' are $\{j_1, \ldots, j_m\}$ and $\{j'_1, \ldots, j'_{m'}\}$, respectively. The associated Bellman kernel is formed by taking a linear combination of the base kernel $k(\cdot, \cdot)$ applied to all possible state pairings (shown by the dashed blue lines) where one state in the pair belongs to the set of i and its descendants (this set is shown by the left shaded rectangle), and the other belongs to the set of i' and its descendants (right shaded rectangle).

are all the states j for which P_{ij}^{μ} are nonzero).

Figure 4-1 is a graphical depiction of the associated Bellman kernel. Using the figure, we can interpret the associated Bellman kernel as measuring the total "overlap" or similarity between i, i', and all immediate (one-stage) successor states of i and i'. In this sense, the associated Bellman kernel automatically accounts for a limited amount of local graph structure in the state space. Of course, we would expect that by examining more states in the vicinity of i and i', a more accurate representation of the similarity of i and i' would be obtained. A natural way to do this is to expand the depth of the successor state tree that is considered in the calculation of the associated Bellman kernel (this amounts to adding more layers of states in Figure 4-1). The next section explains how this can be accomplished.

4.1.1 Development of Multi-Stage BRE

We have seen that the associated Bellman kernel $\mathcal{K}(i, i')$ plays a central role in BRE. This kernel was derived starting from the definition of the Bellman residuals [Eq. (3.6)], which measure the error in solving the Bellman equation $J_\mu = T_\mu J_\mu$. The fact that $\mathcal{K}(i, i')$ incorporates information only about the one-stage successor states of i and i' is directly related to the fact that T_μ defines a relationship between the cost-to-go of a state and its one-stage successors [Eq. (2.8)]. A more general expression for the associated Bellman kernel that incorporates more successor states (and therefore, more of the graph structure of the MDP) can be derived by starting with a multi-stage form of the Bellman equation. This equation is generated by repeatedly applying the dynamic programming operator T_μ:

$$\begin{aligned} T_\mu^2 J_\mu &= T_\mu(T_\mu J_\mu) = T_\mu J_\mu = J_\mu \\ &\vdots \\ T_\mu^l J_\mu &= J_\mu. \end{aligned} \tag{4.1}$$

Then, by taking a convex combination of n equations of the form (4.1), we obtain

$$\sum_{l=1}^{n} \gamma_l T_\mu^l J_\mu = J_\mu, \tag{4.2}$$

where the weights γ_l satisfy $\gamma_l \geq 0$ and $\sum_{l=1}^{n} \gamma_l = 1$. Finally, we define the generalized Bellman residual $BR^n(i)$ to be the error incurred in solving Eq. (4.2):

$$BR^n(i) \equiv \sum_{l=1}^{n} \gamma_l \left(\widetilde{J}_\mu(i) - T_\mu^l \widetilde{J}_\mu(i) \right). \tag{4.3}$$

Using the definition of T_μ and the functional form of the approximate cost-to-go function $\widetilde{J}_\mu(i)$ [Eq. (3.4)], the individual terms $\widetilde{J}_\mu(i) - T_\mu^l \widetilde{J}_\mu(i)$ can be expressed as

$$\widetilde{J}_\mu(i) - T_\mu^l \widetilde{J}_\mu(i) = \widetilde{J}_\mu(i) - T_\mu \left(T_\mu^{l-1} \widetilde{J}_\mu(i) \right)$$

$$
\begin{aligned}
&= \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu T_\mu \left(T_\mu^{l-2} \widetilde{J}_\mu(j_1) \right) \right) \\
&\vdots \\
&= \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \cdots + \alpha \sum_{j_l \in \mathcal{S}} P_{j_{l-1} j_l}^\mu \widetilde{J}_\mu(j_l) \right) \right) \\
&= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \cdots + \alpha \sum_{j_l \in \mathcal{S}} P_{j_{l-1} j_l}^\mu \langle \underline{\Theta}, \underline{\Phi}(j_l) \rangle \right) \right) \\
&= \langle \underline{\Theta}, \underline{\Psi}^l(i) \rangle - g_i^{l,\mu}, \qquad (4.4)
\end{aligned}
$$

where

$$
\underline{\Psi}^l(i) \equiv \underline{\Phi}(i) - \alpha^l \sum_{j_1, \ldots, j_l \in \mathcal{S}} \left(P_{ij_1}^\mu \ldots P_{j_{l-1} j_l}^\mu \right) \underline{\Phi}(j_l) \qquad (4.5)
$$

is an l-stage generalization of the feature mapping $\underline{\Psi}(i)$, and

$$
g_i^{l,\mu} \equiv g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \cdots + \alpha \sum_{j_{l-1} \in \mathcal{S}} P_{j_{l-2} j_{l-1}}^\mu g_{j_{l-1}}^\mu \right) \qquad (4.6)
$$

is the expected cost of state i over l stages. Using Eq. (4.4) allows us to express the generalized Bellman residual $BR^n(i)$ as

$$
\begin{aligned}
BR^n(i) &= \sum_{l=1}^n \gamma_l \left(\langle \underline{\Theta}, \underline{\Psi}^l(i) \rangle - g_i^{l,\mu} \right) \\
&= \left\langle \ \underline{\Theta} \ , \ \sum_{l=1}^n \gamma_l \underline{\Psi}^l(i) \ \right\rangle - \sum_{l=1}^n \gamma_l g_i^{l,\mu}.
\end{aligned}
$$

Therefore, in direct analogy with the derivation of BRE presented earlier, the condition of exactly satisfying the Bellman equation [Eq. (4.2)] at the sample states $\widetilde{\mathcal{S}}$ is equivalent to finding a function $\widetilde{W}_\mu^n(i) = \langle \ \underline{\Theta} \ , \ \sum_{l=1}^n \gamma_l \underline{\Psi}^l(i) \ \rangle$ satisfying

$$
\widetilde{W}_\mu^n(i) = \sum_{l=1}^n \gamma_l g_i^{l,\mu} \quad \forall i \in \widetilde{\mathcal{S}}. \qquad (4.7)
$$

$\widetilde{W}_\mu^n$ belongs to a RKHS whose kernel is the *n-stage associated Bellman kernel*:

$$\begin{aligned}\mathcal{K}^n(i,i') &= \left\langle \sum_{l=1}^{n} \gamma_l \underline{\Psi}^l(i) \ , \ \sum_{l'=1}^{n} \gamma_{l'} \underline{\Psi}^{l'}(i') \right\rangle \\ &= \sum_{l=1}^{n}\sum_{l'=1}^{n} \gamma_l \gamma_{l'} \langle \underline{\Psi}^l(i), \underline{\Psi}^{l'}(i') \rangle \end{aligned} \tag{4.8}$$

As before, the representer theorem implies that $\underline{\Theta}$ can be expressed as

$$\underline{\Theta} = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(\sum_{l=1}^{n} \gamma_l \underline{\Psi}^l(i) \right),$$

so that once the coefficients $\{\lambda_{i'}\}_{i' \in \widetilde{\mathcal{S}}}$ are found by solving Eq. (4.7) with a kernel-based regression technique, the resulting cost-to-go function is given by:

$$\begin{aligned}\widetilde{J}_\mu(i) &= \left\langle \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(\sum_{l=1}^{n} \gamma_l \underline{\Psi}^l(i') \right) \ , \ \underline{\Phi}(i) \right\rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \sum_{l=1}^{n} \gamma_l \langle \underline{\Psi}^l(i'), \underline{\Phi}(i) \rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \sum_{l=1}^{n} \gamma_l \left\langle \underline{\Phi}(i') - \alpha^l \sum_{j_1,\ldots,j_l \in \mathcal{S}} \left(P^\mu_{i' j_1} \ldots P^\mu_{j_{l-1} j_l} \right) \underline{\Phi}(j_l) \ , \ \underline{\Phi}(i) \right\rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \sum_{l=1}^{n} \gamma_l \left(k(i',i) - \alpha^l \sum_{j_1,\ldots,j_l \in \mathcal{S}} \left(P^\mu_{i' j_1} \ldots P^\mu_{j_{l-1} j_l} \right) k(j_l, i) \right). \end{aligned} \tag{4.9}$$

Therefore, the generalized, n-stage BRE procedure for evaluating the fixed policy μ is as follows:

1. Solve the regression problem [Eq. (4.7)] using the n-stage associated Bellman kernel $\mathcal{K}^n(i,i')$ [Eq. (4.8)] and any kernel-based regression technique to find the coefficients $\{\lambda_{i'}\}_{i' \in \widetilde{\mathcal{S}}}$.

2. Use the coefficients found in step 1 to compute the cost-to-go function $\widetilde{J}_\mu$ [Eq. (4.9)].

One can verify that with $n = 1$ this procedure reduces to the single-stage variant of BRE presented earlier. Furthermore, and again in direct analogy with single-stage BRE, it is possible to prove that the generalized Bellman residuals [Eq. (4.3)] of the resulting cost-to-go function $\widetilde{J}_\mu$ obtained by the n-stage BRE procedure are always identically zero at the sample states $\widetilde{\mathcal{S}}$.

4.1.2 Computational Complexity

The computational complexity of n-stage BRE is dominated by two factors: first, computing $\mathcal{K}^n(i, i')$ over every pair of sample states $(i, i') \in \widetilde{\mathcal{S}} \times \widetilde{\mathcal{S}}$ (this information is often called the *Gram matrix* of the kernel), and second, solving the regression problem. As illustrated in Fig. (4-2), computing $\mathcal{K}^n(i, i')$ involves enumerating each n-stage successor state of both i and i' and evaluating the base kernel $k(\cdot, \cdot)$ for each pair of successor states. If β is the average branching factor of the MDP, each state will have β^n n-stage successor states on average, so computing $\mathcal{K}^n(i, i')$ for a single pair of states (i, i') involves $\mathcal{O}(\beta^{2n})$ operations. Therefore, if $n_s \equiv |\widetilde{\mathcal{S}}|$ is the number of sample states, computing the full Gram matrix costs $\mathcal{O}(\beta^{2n} n_s^2)$ operations. The cost of solving the regression problem clearly depends on the regression technique used, but typical methods such as Gaussian process regression involve solving a linear system in the dimension of the Gram matrix, which involves $\mathcal{O}(n_s^3)$ operations. Thus, the total complexity of n-stage BRE is of order

$$\mathcal{O}(\beta^{2n} n_s^2 + n_s^3). \tag{4.10}$$

4.1.3 Graph Structure of the n-stage Associated Bellman Kernel

To understand the structure of the n-stage associated Bellman kernel $\mathcal{K}^n(i, i')$, it is helpful to first examine the generalized feature mapping $\underline{\Psi}^l(i)$ [Eq. (4.5)]. $\underline{\Psi}^l(i)$ is composed of a linear combination of features of the state i itself (this is the term $\underline{\Phi}(i)$), plus features of all states reachable from i in exactly l steps (these are the

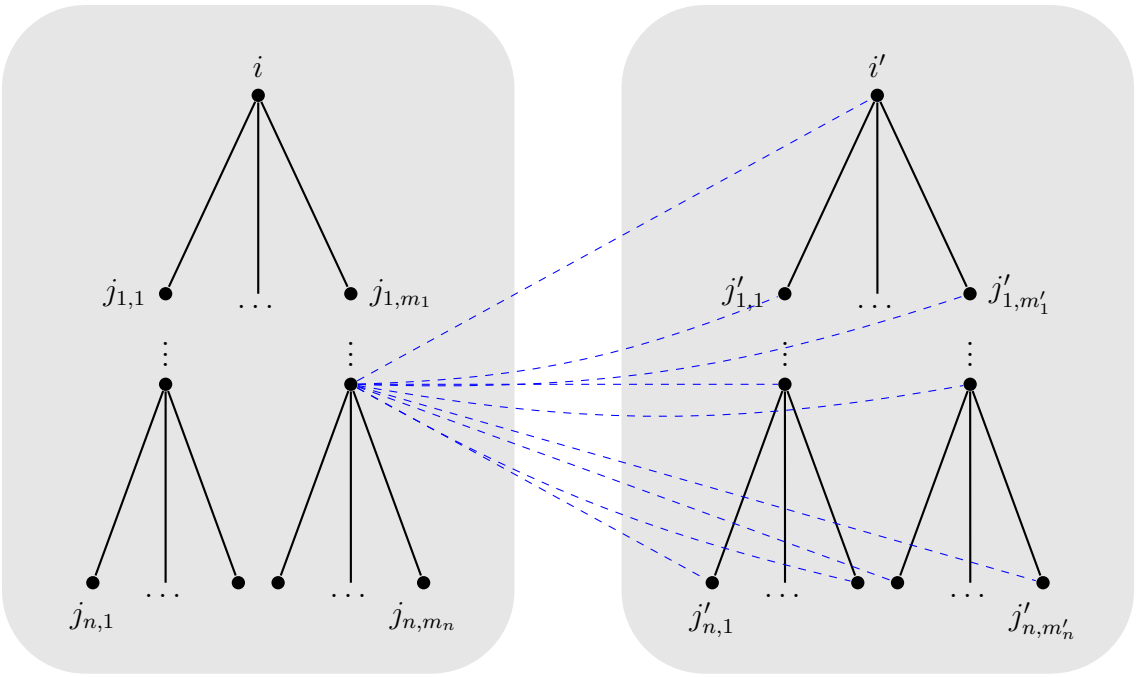


Figure 4-2: Graph structure of the n-stage associated Bellman kernel $\mathcal{K}^n(i, i')$ [Eq. (4.8)]. The possible successor states of i are shown as a tree, where the l^{th} level in the tree represents all m_l states reachable in exactly l steps, starting from i (a similar tree is shown for the successors of i'). The depth of the tree is n. The n-stage associated Bellman kernel is formed by taking a linear combination of the base kernel $k(\cdot, \cdot)$ applied to all state pairings where one state belongs to the set of i and its descendants (left shaded rectangle), and the other belongs to the set of i' and its descendants (right shaded rectangle). For clarity in the figure, only pairings with a fixed state from the left shaded rectangle are depicted (dashed blue lines).

terms $\underline{\Phi}(j_l)$ appearing in the summation). The features of these successor states j_l are weighted by their total probability of being reached; this probability is given by the product $(P^{\mu}_{ij_1} \dots P^{\mu}_{j_{l-1}j_l})$. Thus, the features of the most important (i.e., most probable) successor states receive a higher weighting.

Now examining Eq. (4.8), note that the n-stage associated Bellman kernel consists of a sum of inner products of the form $\langle \underline{\Psi}^l(i), \underline{\Psi}^{l'}(i') \rangle$. Each inner product $\langle \underline{\Psi}^l(i), \underline{\Psi}^{l'}(i') \rangle$ represents the total, probabilistically-weighted similarity between the l-stage successor states of i and the l'-stage successor states of i'. As a result of summing over all values of l and l' up to n, the full n-stage associated Bellman kernel effectively computes the total weighted similarity between all successor states of i and i', up to a depth of n. Figure 4-2 shows a graphical depiction of the n-stage

associated Bellman kernel, highlighting how this similarity between successor states is computed.

By the preceding discussion, $\mathcal{K}^n(i, i')$ represents a natural kernel for use in approximate dynamic programming, since it automatically accounts for the inherent graph structure in the MDP state space. Indeed, its construction ensures that for states i and i' which share many common successor states (i.e. states that can be quickly reached starting from both i and i'), the resulting overlap or similarity computed by $\mathcal{K}^n(i, i')$ will be high. Conversely, if i and i' do not share many common successor states, the similarity will be low.

4.1.4 Related Work

Several authors have explored other ideas for explicitly incorporating graph structure of the state space into an approximate dynamic programming algorithm. In [150, 151], the authors proposed using "geodesic Gaussian kernels" which explicitly compute the graph distance between states in the MDP using the Dijkstra algorithm, and demonstrated improved performance over standard Gaussian kernels in several robotics applications. However, that approach is limited primarily to deterministic problems, whereas the BRE approach presented this chapter naturally incorporates stochastic state transitions by weighting successor states with their transition probabilities, as discussed earlier.

Another idea which has been studied is based on spectral graph theory [102, 103]. In this approach, the state space is modeled as an undirected graph, and the eigenfunctions of the Laplacian operator on this graph are used as basis functions for representing the cost-to-go function $\widetilde{J}_\mu$. The authors demonstrate good performance using these methods in a number of applications. However, since calculation of the eigenfunctions requires working with the Laplacian matrix of the entire graph, these methods may encounter difficulties when the state space is very large. Furthermore, the assumption that the state space is an undirected graph implies that all actions are reversible, which may not be true in some problems. In contrast, the n-stage associated Bellman kernel can be computed using only local information (namely, the

n-stage successor states), and thus may be easier to compute when the state space is large. Furthermore, our approach captures the directedness of the state space graph (since it uses the state transition probabilities P_{ij}^{μ}, which are inherently directional), and thus avoids the need to assume reversibility.

4.2 Model-Free BRE

The family of n-stage BRE algorithms presented to this point require knowledge of the system dynamics model, encoded in the probabilities P_{ij}^{μ}, to evaluate the policy μ. These probabilities are used in the calculation of the cost values $g_i^{l,u}$ [Eq. (4.6)], the associated Bellman kernel $\mathcal{K}^n(i,i')$ [Eq. (4.8)], and the approximate cost-to-go $\widetilde{J}_{\mu}(i)$ [Eq. (4.9)]. In particular, in order to solve the BRE regression problem given by Eq. (4.7), it is necessary to compute $g_i^{l,u}$ for each sample state $i \in \widetilde{\mathcal{S}}$, as well as compute the Gram matrix of the associated Bellman kernel. By exploiting the model information, BRE is able to construct cost-to-go solutions $\widetilde{J}_{\mu}(i)$ for which the Bellman residuals are exactly zero.

However, there may be situations in which it is not desirable or possible to use system model information. Clearly, one such situation is when an exact system model is unknown or unavailable. In this case, one typically assumes instead that a *generative model* is available, which is a "black box" simulator that can be used to sample state trajectories of the system under a given policy. A second situation concerns the computational requirements necessary to carry out n-stage BRE. Recall that the computational complexity of n-stage BRE is $\mathcal{O}(\beta^{2n} n_s^2 + n_s^3)$, where β is the average branching factor of the MDP. If n or β is large, it may become intractable to compute $\mathcal{K}^n(i,i')$ directly, even though model information may be available.

To address these situations, we now develop a variant of n-stage BRE that uses only simulated trajectories from a generative model, instead of using data from the true underlying system model P_{ij}^{μ}. This variant of BRE therefore represents a true, model-free, reinforcement learning algorithm. The key idea behind model-free BRE is to simulate a number of trajectories of length n, starting from each of the sample

states in $\widetilde{\mathcal{S}}$, and use this information to build stochastic approximations to the cost values $g_i^{l,u}$, kernel Gram matrix $\mathbb{K}$, and cost-to-go $\widetilde{J}_\mu(i)$. We use the notation T_l^{iq} to denote the l^{th} state encountered in the q^{th} trajectory starting from state $i \in \widetilde{\mathcal{S}}$, where l ranges from 0 to n (the length of the trajectory), and q ranges from 1 to m (the number of trajectories starting from each state $i \in \widetilde{\mathcal{S}}$).

Using the trajectory data T_l^{iq}, stochastic approximations of each of the important quantities necessary to carry out BRE can be formed. First, examining Eq. (4.6), notice that the cost values $g_i^{l,\mu}$ can be expressed as

$$\begin{aligned} g_i^{l,\mu} &= \mathbb{E}\left[\sum_{l'=0}^{l} \alpha^{l'} g(i_{l'}, \mu(i_{l'})) \mid i_0 = i\right] \\ &\approx \frac{1}{m}\sum_{q=1}^{m}\sum_{l'=0}^{l} \alpha^{l'} g(T_{l'}^{iq}, \mu(T_{l'}^{iq})), \end{aligned} \tag{4.11}$$

where the expectation over future states has been replaced by a Monte Carlo estimator based on the trajectory data. In the limit of sampling an infinite number of trajectories ($m \to \infty$), the approximation given by Eq. (4.11) converges to the true value of $g_i^{l,\mu}$.

A similar approximation can be constructed for the associated Bellman kernel by starting with Eq. (4.5):

$$\begin{aligned} \underline{\Psi}^l(i) &= \underline{\Phi}(i) - \alpha^l \mathbb{E}\left[\underline{\Phi}(i_l) \mid i_0 = i\right] \\ &\approx \underline{\Phi}(i) - \frac{\alpha^l}{m}\sum_{q=1}^{m} \underline{\Phi}(T_l^{iq}). \end{aligned} \tag{4.12}$$

Substituting Eq. (4.12) into Eq. (4.8) gives

$$\mathcal{K}^n(i,i') \approx \sum_{l=1}^{n}\sum_{l'=1}^{n} \gamma_l \gamma_{l'} \left\langle \underline{\Phi}(i) - \frac{\alpha^l}{m}\sum_{q=1}^{m} \underline{\Phi}(T_l^{iq}), \underline{\Phi}(i') - \frac{\alpha^{l'}}{m}\sum_{q'=1}^{m} \underline{\Phi}(T_{l'}^{i'q'}) \right\rangle \tag{4.13}$$

In the limit of infinite sampling, Eq. (4.13) converges to $\mathcal{K}^n(i,i')$.

Finally, an approximation to $\widetilde{J}_\mu(i)$ [Eq. (4.9)] is needed:

$$\begin{aligned}\widetilde{J}_\mu(i) &= \sum_{i'\in\widetilde{\mathcal{S}}}\lambda_{i'}\sum_{l=1}^{n}\gamma_l\left(k(i',i)-\alpha^l\mathbb{E}\left[k(j_l,i)\mid j_0=i'\right]\right)\\ &\approx \sum_{i'\in\widetilde{\mathcal{S}}}\lambda_{i'}\sum_{l=1}^{n}\gamma_l\left(k(i',i)-\frac{\alpha^l}{m}\sum_{q'=1}^{m}k(T_l^{i'q'},i)\right).\end{aligned} \tag{4.14}$$

The procedure for carrying out model-free, n-stage BRE can now be stated as follows:

1. Using the generative model of the MDP, simulate m trajectories of length n starting from each of the sample states $i \in \widetilde{\mathcal{S}}$. Store this data in T_l^{iq}.

2. Solve the regression problem [Eq. (4.7)] using stochastic approximations to $g_i^{l,\mu}$ and $\mathcal{K}^n(i,i')$ (given by Eqs. (4.11) and (4.13), respectively), and any kernel-based regression technique to find the coefficients $\{\lambda_{i'}\}_{i'\in\widetilde{\mathcal{S}}}$.

3. Use the coefficients found in step 1 to compute a stochastic approximation to the cost-to-go function $\widetilde{J}_\mu$, given by Eq. (4.14).

4.2.1 Computational Complexity

One of the motivations for developing the model-free BRE variant was to reduce the computational complexity of running BRE when either n or β is large. To see that this is indeed the case, note that the overall complexity is still dominated by building the kernel Gram matrix and solving the regression problem, just as in model-based BRE. Furthermore, the complexity of solving the regression problem is the same between both methods: $\mathcal{O}(n_s^3)$. The main savings gained though using model-free BRE comes about because computing an element of the associated Bellman kernel using Eq. (4.13) now requires only $\mathcal{O}(m^2)$ operations, instead of $\mathcal{O}(\beta^{2n})$ as in the model-based case. In essence, in model-free BRE, the n-stage successor states for each sample state $i \in \widetilde{\mathcal{S}}$ (of which there are on average β^n) are approximated using only m simulated trajectories. Thus, computing the full Gram matrix requires $\mathcal{O}(m^2 n_s^2)$ operations,

and the total complexity of model-free BRE is of order

$$\mathcal{O}(m^2 n_s^2 + n_s^3). \tag{4.15}$$

Comparing this to the complexity for model-based BRE, $\mathcal{O}(\beta^{2n} n_s^2 + n_s^3)$, note that the exponential dependence on n has been eliminated.

4.2.2 Correctness of Model-Free BRE

The following theorem and lemma establish two important properties of the model-free BRE algorithm presented in this section.

Theorem 10. *In the limit of sampling an infinite number of trajectories (i.e. $m \to \infty$), the cost-to-go function $\widetilde{J}_\mu(i)$ computed by model-free BRE is identical to the cost-to-go function computed by model-based BRE.*

Proof. In order to approximate the quantities g_i^μ, $\mathcal{K}(i, i')$, and $\widetilde{J}_\mu(i)$ in the absence of model information, model-free BRE uses the stochastic approximators given by Eqs. (4.11), (4.13), and (4.14), respectively. Examining these equations, note that in each, the state trajectory data T_l^{iq} is used to form an empirical distribution

$$\hat{P}_{ij}^\mu = \frac{1}{m} \sum_{q=1}^{m} \delta(T_1^{iq}, j),$$

where $\delta(i, j)$ is the Kronecker delta function. This distribution is used as a substitute for the true distribution P_{ij}^μ in computing g_i^μ, $\mathcal{K}(i, i')$, and $\widetilde{J}_\mu(i)$. Since by assumption the individual trajectories are independent, the random variables $\{T_1^{iq} | q = 1, \ldots, m\}$ are independent and identically distributed. Therefore, the law of large numbers states that the empirical distribution converges to the true distribution in the limit of an infinite number of samples:

$$\lim_{m \to \infty} \hat{P}_{ij}^\mu = P_{ij}^\mu.$$

Therefore, as $m \to \infty$, Eqs. (4.11), (4.13), and (4.14) converge to the true values g_i^μ,

$\mathcal{K}(i, i')$, and $\widetilde{J}_\mu(i)$. In particular, the cost-to-go function $\widetilde{J}_\mu(i)$ computed by model-free BRE converges to the cost-to-go function computed by model-based BRE as $m \to \infty$, as desired. □

Using results shown in [28, 29], which prove that model-based BRE computes a cost-to-go function $\widetilde{J}_\mu(i)$ whose Bellman residuals are exactly zero at the sample states $\widetilde{\mathcal{S}}$, we immediately have the following lemma:

Lemma *In the limit $m \to \infty$, model-free BRE computes a cost-to-go function $\widetilde{J}_\mu(i)$ whose Bellman residuals are exactly zero at the sample states $\widetilde{\mathcal{S}}$.*

Proof. The theorem showed that in the limit $m \to \infty$, model-free BRE yields the same cost-to-go function $\widetilde{J}_\mu(i)$ as model-based BRE. Therefore, applying the results from [28, 29], it immediately follows that the Bellman residuals $\widetilde{J}_\mu(i)$ are zero at the sample states. □

4.3 Simulation Results

4.3.1 Robot Navigation

A "grid world" robot navigation problem (see, for example, [103, 133, 150]) was used to test the performance of the n-stage BRE method. In this problem, a robot moves in a room modeled as a a two-dimensional, 21×11 discrete grid. At any point on the grid, the robot may choose to try to move one unit up, right, left, or down. The environment is stochastic, such that the robot moves to its intended point with probability 0.8, and moves to either side of its intended point with probability 0.1 for each side (this effect might be due to wheel slippage or navigation errors, for example). The room is divided by a wall into left and right halves which are joined only by a small passageway in the middle of the room. The goal of the robot is to navigate to a goal state in the far corner of the right-hand room.

Figure 4-3 illustrates the effect of varying n on the structure of the n-stage associated Bellman kernel for this example problem. The figure shows the value of kernel $\mathcal{K}^n(i, i')$ over the state space for a fixed state $i' = (9, 8)$ in the center of the room.

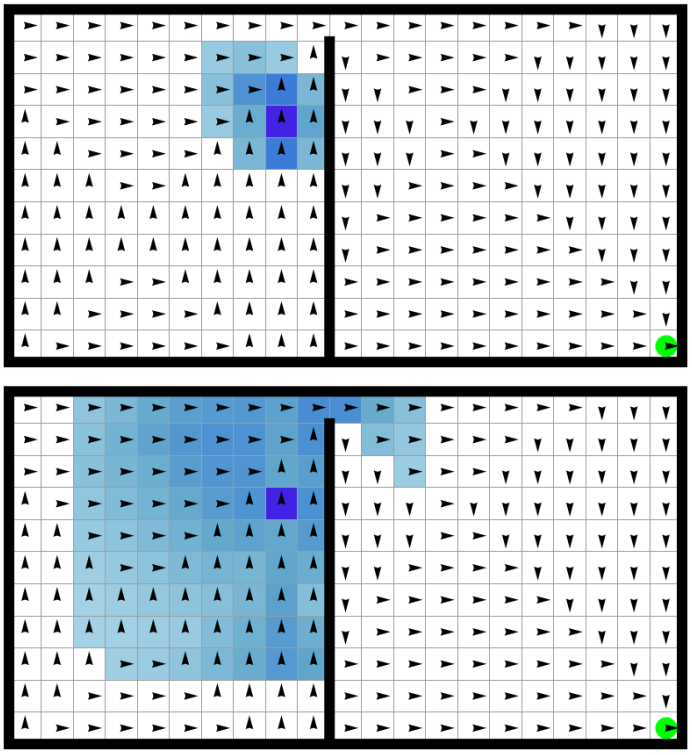


Figure 4-3: n-stage associated Bellman kernel centered at $(9, 8)$ (highlighted in purple) for the optimal policy in the robot navigation problem, for various values of n (top: $n = 1$, bottom: $n = 5$). The optimal policy is shown as the small arrows, and the goal state is marked with a green dot. Areas for which the kernel is nonzero are shaded blue, and areas for which it is zero are white.

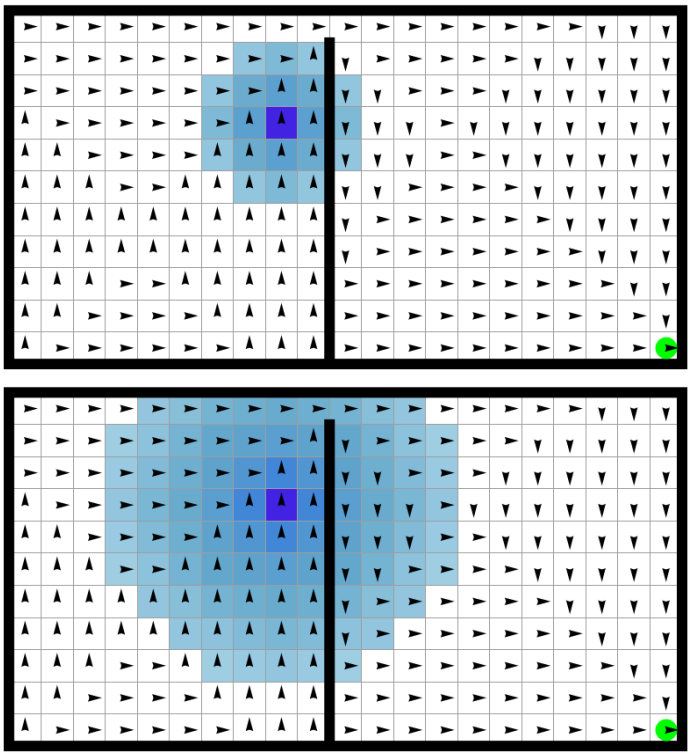


Figure 4-4: Standard RBF kernel $\exp\left(-||i - i'||^2/\gamma\right)$, centered at $(9, 8)$, for $\gamma = 1.0$ (top) and $\gamma = 4.0$ (bottom). Unlike the n-stage associated Bellman kernel, the RBF kernel incorrectly predicts high similarity between states on opposite sides of the wall.

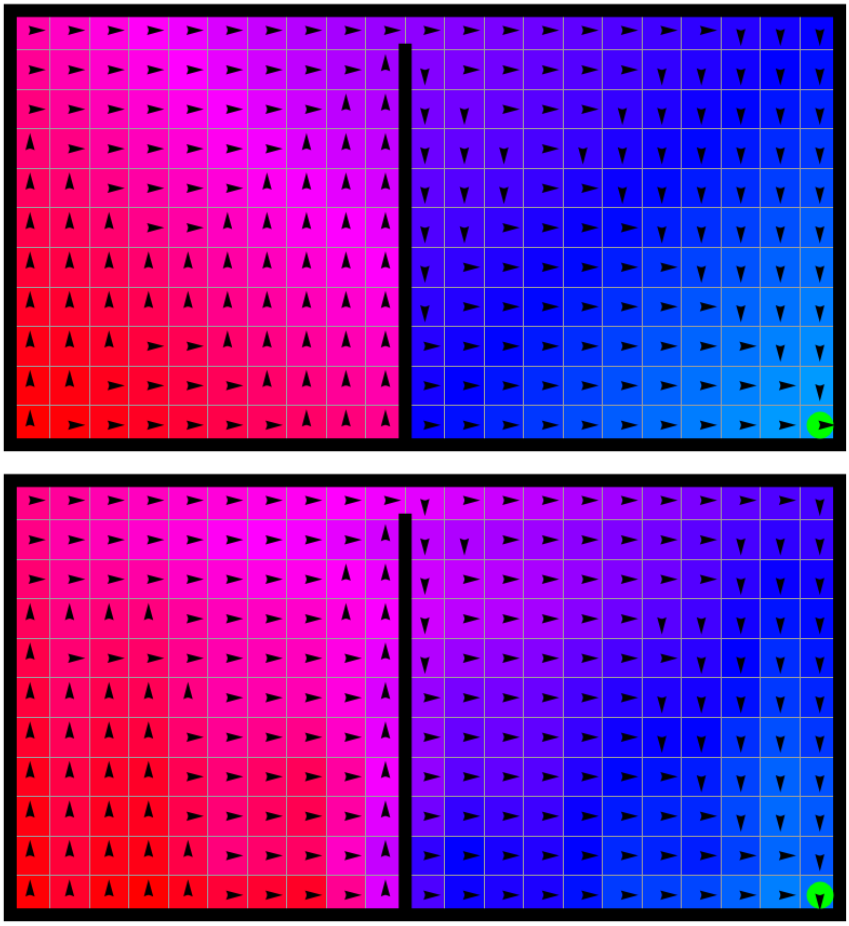


Figure 4-5: Optimal policy and cost-to-go for the robot navigation problem (top), and policy and cost-to-go generated by 4-stage BRE (bottom).

In the figure, the base kernel $k(i, i')$ is taken to be the Kronecker delta function, the simplest kernel possible. Notice that the value of the kernel is zero immediately to the right of the wall, correctly implying little similarity in cost-to-go between these states. This remains true even as n increases and the kernel's "footprint" grows, leading to better generalization over the state space and higher-quality policies, as we will discuss shortly. In contrast, Figure 4-4 shows the behavior of a standard RBF kernel based on Euclidean distance. This kernel, being unaware of the structure of the state space, incorrectly predicts high similarity between states on opposite sides of the wall. This highlights the importance of using a kernel which is aware of the structure of the state space.

To further illustrate the benefits of using the structured, n-stage associated Bell-

man kernel over standard RBF kernels, a series of experiments were performed. First, exact value iteration was used to compute the optimal policy for the robot navigation problem. Then, approximate policy iteration using basic, single-stage BRE and a standard RBF kernel were used to compute an approximate policy. Finally, approximate policy iteration using 4- and 6- stage BRE was used to compute an approximate policy. A comparison of the policies and associated cost-to-go functions for the optimal solution and the 4-stage BRE solution is shown in Figure 4-5. In the figure, the colors drawn in each of the squares represents the cost-to-go function, where red indicates higher cost and blue indicates lower cost. The policy is shown by the arrows in each square, indicating the direction of movement.

A number of simulations of each policy were run, and the average time to reach the goal (averaged over all possible starting states) was computed. The average time for the optimal policy, the single-stage RBF policy, and the 4- and 6-stage BRE policies were 14.9, 27.2, 17.6, and 16.3, respectively, showing that multi-stage BRE yields near-optimal performance and significantly outperforms the policy based on the RBF kernel. Furthermore, as discussed, 6-stage BRE slightly outperforms 4-stage BRE due to the larger "footprint" and generalization ability of the 6-stage associated Bellman kernel; this performance gain is achieved at the expense of higher computational requirements to compute the kernel.

4.3.2 Chain Walk

A further series of tests were carried out to compare the performance of the BRE algorithms presented in this chapter to LSPI [93], a well-known approximate dynamic programming algorithm. LSPI uses a linear combination of basis functions to represent approximate Q-values of the MDP, and learns a weighting of these basis functions using simulated trajectory data. In learning the weights, LSPI can take advantage of system model information if it is available, or can be run in a purely model-free setting if not. Thus, our tests compared the performance of four different algorithms: model-based BRE, model-free BRE, model-based LSPI, and model-free LSPI. The LSPI implementation used for these tests is the one provided by the authors in [93].

The benchmark problem used was the "chain-walk" problem [91, 93], which has a one-dimensional state space and two possible actions ("move left" and "move right") in each state. In the first set of experiments, a total of 50 states were used, similar to the experiments presented in [93]. In order to make the comparisons between BRE and LSPI as similar as possible, the same cost-to-go representation was used in both algorithms. More precisely, in LSPI, five radial basis functions (with standard deviation $\sigma = 12$), with centers at $x = 1, 11, 21, 31, 41$, were used; while in BRE, the same radial basis kernel (with the same σ) was used and the sample states were taken as $\widetilde{\mathcal{S}} = \{1, 11, 21, 31, 41\}$. This ensures that neither algorithm gains an unfair advantage by being provided with a better set of basis functions. Furthermore, for these tests, only single-stage BRE ($n = 1$) was used, since we found that single-stage BRE consistently found optimal or very close to optimal policies, and increasing n therefore had no chance to further improve performance.

The algorithms were compared on two different performance metrics: quality of the approximate policy produced by each algorithm (expressed as a percentage of states in which the approximate policy matches the optimal policy), and running time of the algorithm (measured in the total number of elementary operations required, such as additions and multiplications). The policies produced LSPI and the model-free variant of BRE depend on the amount of simulated data provided to the algorithm, so each of these algorithms was run with different amounts of simulated data to investigate how the amount of data impacts the quality of the resulting policy. Furthermore, since the simulated data is randomly generated according to the generative model, each algorithm was run multiple times to examine the effect of this random simulation noise on the algorithm performance.

The results of the comparison tests are shown in Fig. 4-6. Model-based BRE, shown as the filled blue diamond, finds the optimal policy. Furthermore, its running time is faster than any other algorithm in the test by at least one order of magnitude. In addition, it is the only algorithm that is free from simulation noise, and it therefore consistently finds the optimal policy every time it is run, unlike the other algorithms which may yield different results over different runs. Thus, if a system model is

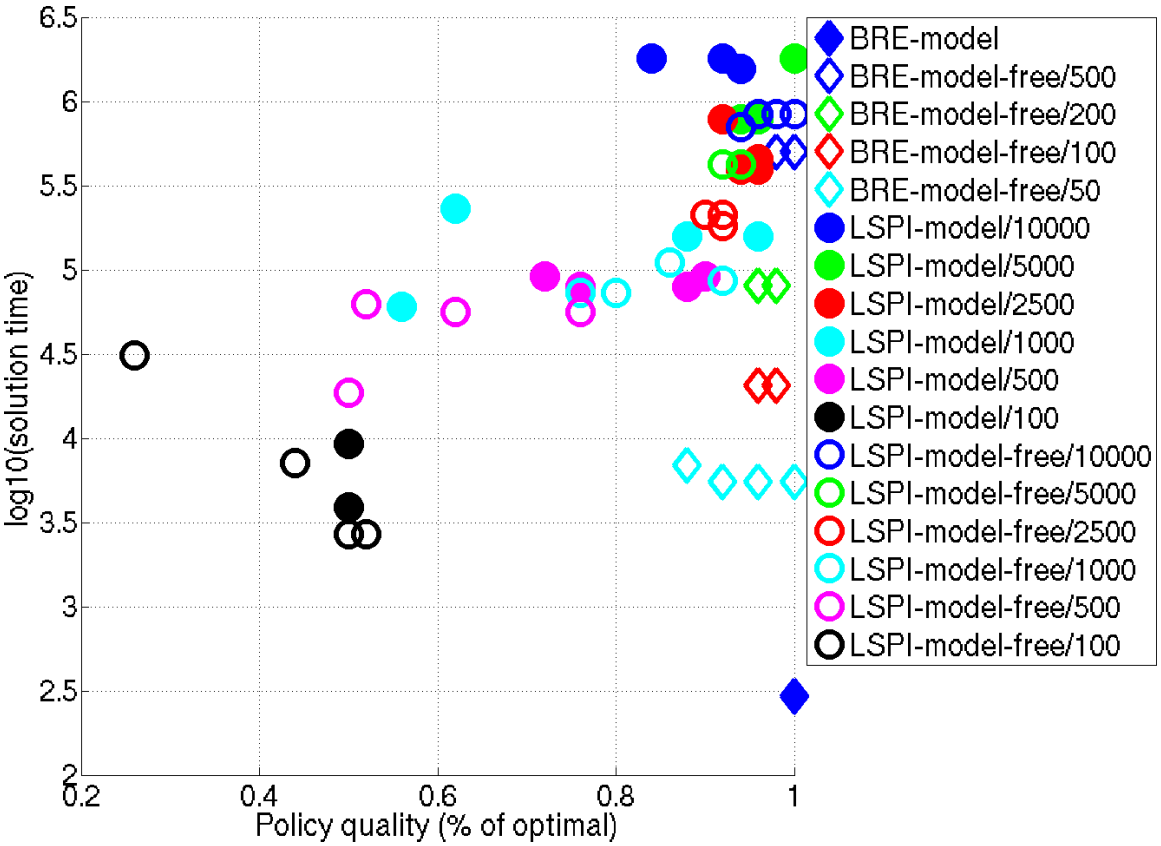


Figure 4-6: Comparison of BRE vs. LSPI for the "chain-walk" problem. Numbers after the algorithm names denote the amount of simulation data used to train the algorithm. Note that model-based BRE does not require simulation data.

available, model-based BRE has a clear advantage over the other algorithms in the chain-walk problem.

LSPI also found the optimal policy in a number of the tests, confirming similar results that were reported in [93]. However, the amount of simulation data required to consistently find a near-optimal policy was large (between 5,000 and 10,000 samples), leading to long run times. Indeed, for any of the LSPI variants that consistently found policies within 10% of optimal, all were between two and four orders of magnitude slower than model-based BRE. As the amount of simulation data is decreased in an attempt to reduce the solution time of LSPI, the quality of the produced policies becomes lower on-average and also more inconsistent across runs. For simulated data sets of less than about 1,000, the policy quality approaches (and sometimes drops below) 50% of optimal, indicating performance equivalent to (or worse than) simply

guessing one of the two actions randomly. Allowing LSPI access to the true system model does appear to improve the performance of the algorithm slightly; in Fig. 4-6, model-based LSPI generally yields a higher quality policy than model-free LSPI for a given data set size (although there are several exceptions to this). However, the results indicate the model-based BRE is significantly more efficient than model-based LSPI at exploiting knowledge of the system model to reduce the computation time needed to find a good policy.

Finally, examining the results for model-free BRE, the figure shows that this algorithm yields consistently good policies, even when a small amount of simulation data is used. As the amount of simulation data is reduced, the variability of the quality of policies produced increases and the average quality decreases, as would be expected. However, both of these effects are significantly smaller than in the LSPI case. Indeed, the worst policy produced by model-free BRE is still within 12% of optimal, while using only 50 simulation data. In contrast, LSPI exhibited a much greater variability and much lower average policy quality when the amount of simulation data was reduced.

In a second set of experiments, the size of the state space was increased (to $10,000$) to make the problem more challenging. The number of basis functions (in the case of LSPI) and sample states (in the case of BRE) was increased accordingly, using an evenly spaced grid of 100 radial basis functions with standard deviation $\sigma = 2,000$. In applying the BRE algorithms to this problem, it was noted that one-stage BRE encountered numerical difficulties due to the Gram matrix being near-singular. This issue is due to the fact in the larger chain-walk problem, it is necessary to use a wider kernel (one with larger σ) in order to generalize over many states that are not sampled. However, since one-stage BRE only considers one-stage successors of the sample states, the kernels centered at a sample state and the successor states overlap almost completely, leading to near-singularity of the Gram matrix. To address this issue, the n-stage, model-free variant of BRE was applied instead, for $n = 200$. It was found that these algorithms eliminated the numerical issues (due to the fact that they consider successor states that are further apart in the state space, as measured by the

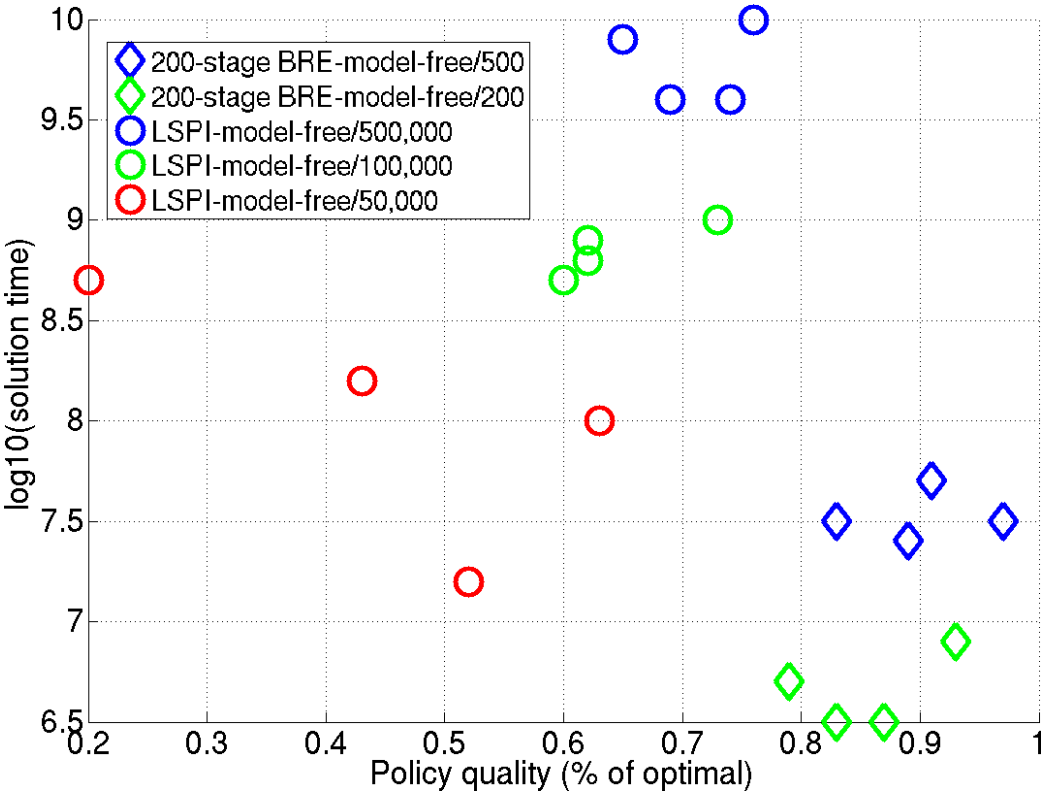


Figure 4-7: Comparison of 200-stage BRE vs. LSPI for a larger "chain-walk" problem. Numbers after the algorithm names denote the amount of simulation data used to train the algorithm.

normal Euclidean distance), allowing the algorithms to proceed. Note that for these large values of n, performing model-based BRE is impractical since the algorithm would be required to perform computations for each of the β^n successor states of each sample state (for the chain-walk problem, $\beta = 2$); however, as discussed earlier, the model-free BRE variant avoids this difficulty by sampling the successor states. Similar numerical issues, related to near-singularity in the linear system that LSPI solves, were also noted with the LSPI algorithm in this larger problem if insufficient amounts of training data were used. To address the numerical issues for the LSPI algorithm, it was necessary to increase the amount of training data substantially (using between $50,000$ and $500,000$ samples).

Results of the larger chain-walk experiments are shown in Figure 4-7. Similar to the results for the smaller-scale chain-walk experiments, the BRE algorithms tend to give more consistent, higher quality policies compared to LSPI, while requiring

less computational effort. Note that memory usage issues in the reference Matlab implementation of LSPI prevented using more than $500,000$ training samples.

4.4 Summary

This chapter has presented several extensions of the BRE approach to approximate dynamic programming. The first extension, multi-stage BRE, eliminates Bellman residuals of the form $|\widetilde{J}_\mu(i) - T_\mu^n \widetilde{J}_\mu(i)|$, where $n \geq 1$ is an integer. As a result of this multi-stage extension, a specialized kernel, called the n-stage associated Bellman kernel, naturally arises. This kernel automatically accounts for the inherent graph structure in the state space by accounting for the total "overlap" between all successor states up to a depth of n. Our approach has several advantages over other approaches which are also designed to exploit graph structure in the state space. Applying the multi-stage BRE approach to a stochastic robot navigation problem shows that the n-stage associated Bellman kernel is a natural and intuitive representation of similarity on the state space, and can produce approximate policies that have significantly better performance than those generated using standard RBF kernels.

The second extension extends BRE to the model-free case, in which knowledge of the system transition probabilities P_{ij}^μ is not required. Instead of relying on knowledge of the system model, the model-free variant uses trajectory simulations or data from the real system to build stochastic approximations to the cost values $g_i^{l,\mu}$ and n-stage associated Bellman kernel $\mathcal{K}^n(i, i')$ needed to carry out BRE. It is straightforward to show that in the limit of carrying out an infinite number of simulations, this approach yields the same results as the model-based BRE approach, and thus enjoys the same theoretical properties of that approach (including the important fact that the Bellman residuals are identically zero at the sample states $\widetilde{\mathcal{S}}$.

Furthermore, experimental comparison of BRE against another well-known approach, LSPI, indicates that while both approaches find near-optimal policies, both model-based and model-free BRE appears to have several advantages over LSPI in the benchmark problem we tested. In particular, model-based BRE appears to be able to

efficiently exploit knowledge of the system model to find the policy significantly faster than LSPI. In addition, model-based BRE is free of simulation noise, eliminating the problem of inconsistent results across different runs of the algorithm. Even when model information is not available, model-free BRE still finds near-optimal policies more consistently and more quickly than LSPI.

Chapter 5

Multi-Agent Health Management and the Persistent Surveillance Problem

UAVs are becoming increasingly sophisticated in terms of hardware capabilities. Advances in sensor systems, onboard computational platforms, energy storage, and other enabling technologies have made it possible to build a huge variety of UAVs for a range of different mission scenarios [6, 116]. Many of the mission scenarios of interest, such as persistent surveillance, are inherently long-duration and require coordination of multiple cooperating UAVs in order to achieve the mission objectives. In these types of missions, a high level of autonomy is desired due to the logistical complexity and expense of direct human control of each individual vehicle. Currently, autonomous mission planning and control for multi-agent systems is an active area of research [69, 77, 92, 113, 119]. Some of the issues in this area are similar to questions arising in manufacturing systems [22, 74] and air transportation [7, 14, 25, 48, 72, 76, 131]. While these efforts have made significant progress in understanding how to handle some of the complexity inherent in multi-agent problems, there remain a number of open questions in this area.

This chapter investigates one important question that is referred to as the *health management problem for multi-agent systems* [98, 99]. Designs of current and future

UAVs increasingly incorporate a large number of sensors for monitoring the health of the vehicle's own subsystems. For example, sensors may be installed to measure the temperature and current of electric motors, the effectiveness of the vehicle's control actuators, or the fuel consumption rates in the engine. On a typical UAV, sensors may provide a wealth of data about a large number of vehicle subsystems. By making appropriate use of this data, a health-aware autonomous system may be able to achieve a higher level of overall mission performance, as compared to a non-health-aware system, by making decisions that account for the current capabilities of each agent. For example, in a search and track mission, utilization of sensor health data may allow an autonomous system to assign the UAVs with the best-performing sensors to the search areas with the highest probability of finding the target.

Utilization of the current status of each vehicle is an important aspect of the health management problem. Another important aspect is the ability not only to react to the current status, but to consider the implications of future changes in health status or failures on the successful outcome of the mission. This predictive capability is of paramount importance, since it may allow an autonomous system to avoid an undesirable future outcome. For example, if a UAV tracking a high value target is known to have a high probability of failure in the future, the autonomous system may be able to assign a backup vehicle to track the target, ensuring that the tracking can continue even if one of the vehicles fails.

This chapter addresses these health management issues and develops a general framework for thinking about the health management problem. It then specializes the discussion to the persistent surveillance problem with a focus on modeling numerous realistic failure modes and constraints that are present in actual UAV missions. In particular, the persistent surveillance problem formulation includes uncertainty in the fuel consumption rates of the UAVs (due to random environmental factors, engine degradation, etc), communication constraints (which require a communications link to be maintained with the base location at all times), and randomly-occurring sensor failures (which render the UAV incapable of performing surveillance, although it may still be useful for other tasks such as communications relay). In addition, the

problem formulation can account for heterogeneous teams of UAVs, where different classes of UAVs have different capabilities. This work builds on previous health management techniques developed for the persistent surveillance problem [98]. While the previous work focused on embedding health-aware heuristics into an already-existing mission management algorithm, this chapter develops a new formulation of the problem in which health-aware behaviors emerge automatically. Simulated flight results are provided in this chapter which demonstrate the properties of the new problem formulation for small problem instances that can be solved exactly. Later chapters will focus on efficiently solving larger and more complex instances of the problem using our Bellman residual elimination algorithms, as well as present persistent surveillance flight results for a number of complex mission scenarios.

5.1 Health Management In Multi-Agent Systems

The term *health management* is often used in different contexts, so it can be difficult to define exactly what it means for a multi-agent system to incorporate health management techniques. To make the problem more precise, we can define a number of general properties that we would like a multi-agent system to exhibit. Once these properties are defined, potential design methodologies for incorporating health management into such systems can be evaluated.

In the context of multi-agent systems, health management refers to accounting for changing resource or capability levels of the agents. These changes may be caused by failures, degradations, or off-nominal conditions (of actuators, sensors, propulsion systems, etc), or by unpredictable events in the environment. Broadly speaking, we would like a multi-agent system to exhibit the following properties:

1. The system should be proactive. That is, the system should be capable of "looking into the future" to anticipate events that are likely to occur, and given this information, actively select a course of action that leads to a desirable state and/or avoids an undesirable state. In contrast, a reactive system is incapable of making such future predictions (or cannot effectively use such predictions if

they are available). Thus, a reactive system can only respond to failures after they occur, instead of trying to avoid them in the first place.

2. The system should manage health information at the group, not just the individual, level. In most multi-agent mission scenarios, there are strong coupling effects between vehicles that must be accounted for. For example, in the multi-UAV task assignment problem, failure of a single UAV may necessitate reassigning all the other UAVs to different targets in order to continue the mission. These coupling effects may be very complex, depending on the mission. Nevertheless, they must be considered if the system is to be robust to changing conditions.

5.1.1 Design Considerations

Given the above properties, we now consider some of their implications for design of multi-agent systems. First, Property 1 implies that the system must have a model of its environment in order to anticipate the system state in the future. This model should account for how the current control action will influence the future state. Furthermore, since many of the events of interest in the health management problem—such as failures—cannot be predicted with certainty, the model should be stochastic, and the system should account for the relative probability of possible future events in selecting actions. In general, then, the system model will be of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) \tag{5.1}$$

where $\mathbf{x}$ is the system state, $\mathbf{u}$ are the control inputs, $\mathbf{w}$ are random variables that capture the uncertainty in the system dynamics, and k is a discrete time index.

If the system is to proactively choose control actions that lead to desirable future states, it therefore must have some method of ascertaining whether a given state is desirable or not. A natural way to accomplish this is to define a cost function $g(\mathbf{x}, \mathbf{u})$ which maps state/action pairs $(\mathbf{x}, \mathbf{u})$ to a scalar cost value.

Property 2 also has implications for the design of health-enabled systems. Normally, the interdependence between agents plays a large role in overall mission performance. Therefore, the system model $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$ and cost function $g(\mathbf{x}, \mathbf{u})$ should capture this interdependence so that knowledge of how health related events such as failures can be exploited.

The considerations presented here lead naturally to the idea of applying MDP techniques to the multi-agent health management problem. MDPs provide an attractive method for achieving proactive behavior, since the problem solution method involves calculating the actions that minimize not only the current cost, but also the expected future cost [22]. Furthermore, formulating the health management problem as an MDP allows the interdependence between agents to be encoded naturally in the system model and cost function.

Consideration must be given to computational issues in the formulation of the system model. The choice of too complex a model may lead to an unnecessarily large state space, rendering the problem difficult to solve and implement in real-time. On the other hand, selecting a model which is too simplified may not adequately capture important aspects of the problem, leading to poor performance. Thus, a balance between model complexity and computational tractability must be struck in the formulation of the health management problem. In the following sections, we use the multi-vehicle persistent surveillance problem as an example to show how an appropriate dynamic program capturing the important aspects of the problem can be formulated and solved, and show that the resulting control policy exhibits the desirable properties discussed in this section.

5.2 Persistent Surveillance With Group Fuel Management

Health management techniques are an enabling technology for multi-vehicle persistent surveillance missions. In the model of this scenario considered here, there is a group

of n UAVs equipped with cameras or other types of sensors. The UAVs are initially located at a base location, which is separated by some (possibly large) distance from the surveillance location. The objective of the problem is to maintain a specified number r of requested UAVs over the surveillance location at all times.

The UAV vehicle dynamics provide a number of interesting health management aspects to the problem. In particular, management of fuel is an important health management concern. The vehicles have a certain maximum fuel capacity F_{max}, and we assume that the rate $\dot{F}_{burn}$ at which they burn fuel may vary stochastically during the mission due to aggressive maneuvering that may be required for short time periods, engine wear and tear, adverse environmental conditions, etc. Thus, the total flight time each vehicle may achieve on a full tank of gas is a random variable, and we would like to account for this uncertainty in the problem. If a vehicle runs out of fuel while in flight, it crashes and is lost. Finally, when the vehicle returns to base, it begins refueling at a deterministic rate given by $\dot{F}_{refuel}$ (i.e., the vehicles take a finite time to refuel).

Another health management concern we will model in the problem is the possibility for randomly-occurring vehicle failures. These may be due to sensors, engines, control actuators, or other mission-critical systems failing in flight.

Note that we assume the vehicles run low-level controllers that accept commands such as "take off", "land", and "fly to waypoint." By making this assumption, we are free to design the high-level decision making process that handles the persistent surveillance activity without having to complicate the problem formulation with low-level control considerations. This leads to a natural and hierarchical overall control architecture. More details about this control architecture will be presented in Chapter 7, which also discusses how the architecture can be implemented on a real hardware testbed.

5.2.1 MDP Formulation

Given the description of the persistent surveillance problem, a suitable MDP can now be formulated. The MDP is defined by its state vector $\mathbf{x}$, control vector $\mathbf{u}$, state

transition model $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$, cost function $g(\mathbf{x}, \mathbf{u})$, and discount factor α.

State Space

The state of each UAV is given by two scalar variables describing the vehicle's flight status and fuel remaining. The flight status y_i describes the UAV location,

$$y_i \in \{Y_b, Y_0, Y_1, \ldots, Y_s, Y_c\} \tag{5.2}$$

where Y_b is the base location, Y_s is the surveillance location, $\{Y_0, Y_1, \ldots, Y_{s-1}\}$ are transition states between the base and surveillance locations (capturing the fact that it takes finite time to fly between the two locations), and Y_c is a special state denoting that the vehicle has crashed.

Similarly, the fuel state f_i is described by a discrete set of possible fuel quantities,

$$f_i \in \{0, \Delta f, 2\Delta f, \ldots, F_{max} - \Delta f, F_{max}\} \tag{5.3}$$

where Δf is an appropriate discrete fuel quantity.

The total system state vector $\mathbf{x}$ is thus given by the states y_i and f_i for each UAV, along with r, the number of requested vehicles:

$$\mathbf{x} = (y_1, y_2, \ldots, y_n; f_1, f_2, \ldots, f_n; r)^T \tag{5.4}$$

The size of the state space N_{ss} is found by counting all possible values of $\mathbf{x}$, which yields:

$$N_{ss} = (n+1)\left((Y_s + 3)\left(\frac{F_{max}}{\Delta f} + 1\right)\right)^n \tag{5.5}$$

Control Space

The controls u_i available for the i^{th} UAV depend on its current flight status y_i.

- If $y_i \in \{Y_0, \ldots, Y_s - 1\}$, then the vehicle is in the transition area and may either move away from base or toward base: $u_i \in \{\text{“}+\text{”}, \text{“}-\text{”}\}$

- If $y_i = Y_c$, then the vehicle has crashed and no action for that vehicle can be taken: $u_i = \emptyset$
- If $y_i = Y_b$, then the vehicle is at base and may either take off or remain at base: $u_i \in \{\text{“take off”}, \text{“remain at base”}\}$
- If $y_i = Y_s$, then the vehicle is at the surveillance location and may loiter there or move toward base: $u_i \in \{\text{“loiter”}, \text{“}-\text{”}\}$

The full control vector $\mathbf{u}$ is thus given by the controls for each UAV:

$$\mathbf{u} = (u_1, \ldots, u_n)^T \tag{5.6}$$

State Transition Model

The state transition model (Equation 5.1) captures the qualitative description of the dynamics given at the start of this section. The model can be partitioned into dynamics for each individual UAV.

The dynamics for the flight status y_i are described by the following rules:

- If $y_i \in \{Y_0, \ldots, Y_s - 1\}$, then the UAV moves one unit away from or toward base as specified by the action $u_i \in \{\text{“}+\text{”}, \text{“}-\text{”}\}$ with probability $(1 - p_{crash})$, and crashes with probability p_{crash}.
- If $y_i = Y_c$, then the vehicle has crashed and remains in the crashed state forever afterward.
- If $y_i = Y_b$, then the UAV remains at the base location with probability 1 if the action “remain at base” is selected. If the action “take off” is selected, it moves to state Y_0 with probability $(1 - p_{crash})$, and crashes with probability p_{crash}.
- If $y_i = Y_s$, then if the action “loiter” is selected, the UAV remains at the surveillance location with probability $(1 - p_{crash})$, and crashes with probability p_{crash}. Otherwise, if the action “$-$” is selected, it moves one unit toward base with probability $(1 - p_{crash})$, and crashes with probability p_{crash}.
- If at any time the UAV's fuel level f_i reaches zero, the UAV transitions to the crashed state $(y_i = Y_c)$ with probability 1.

The dynamics for the fuel state f_i are described by the following rules:

- If $y_i = Y_b$, then f_i increases at the rate $\dot{F}_{refuel}$ (the vehicle refuels)
- If $y_i = Y_c$, then the fuel state remains the same (the vehicle is crashed)
- Otherwise, the vehicle is in a flying state and burns fuel at a stochastically modeled rate: f_i decreases at the rate $\dot{F}_{burn}$ with probability $p_{f_{nominal}}$ and decreases at the rate $2\dot{F}_{burn}$ with probability $(1 - p_{f_{nominal}})$.

Cost Function

The cost function $g(\mathbf{x}, \mathbf{u})$ has three distinct components due to loss of surveillance area coverage, vehicle crashes, and fuel usage, and can be written as

$$g(\mathbf{x}, \mathbf{u}) = C_{loc} \max\{0, (r - n_s(\mathbf{x}))\} + C_{crash} n_{crashed}(\mathbf{x}) + C_f n_f(\mathbf{x})$$

where:

- $n_s(\mathbf{x})$: number of UAVs in surveillance area in state $\mathbf{x}$,
- $n_{crashed}(\mathbf{x})$: number of crashed UAVs in state $\mathbf{x}$,
- $n_f(\mathbf{x})$: total number of fuel units burned in state $\mathbf{x}$,

and C_{loc}, C_{crash}, and C_f are the relative costs of loss of coverage events, crashes, and fuel usage, respectively.

5.3 Basic Problem Simulation Results

In order to solve the basic persistent surveillance MDP, a software framework was developed. The framework allows for generic MDPs to be programmed by specifying appropriate system transition, action, and cost functions. Once programmed, the framework applies the value iteration algorithm to iteratively solve Bellman's equation for the optimal cost-to-go $J^\star(\mathbf{x}_k)$. The results of this computation are stored on disk as a lookup table. Once $J^\star(\mathbf{x}_k)$ is found for all states $\mathbf{x}_k$, the optimal control action $\mathbf{u}^\star(\mathbf{x}_k)$ can be quickly computed by choosing the action which minimizes the expected future cost [22]:

$$\mathbf{u}^\star(\mathbf{x}_k) = \arg\min_{\mathbf{u}} \left(g(\mathbf{x}_k, \mathbf{u}) + \mathbb{E}_{\mathbf{x}_{k+1}} \left[\alpha J^\star(\mathbf{x}_{k+1}) \right] \right) \tag{5.7}$$

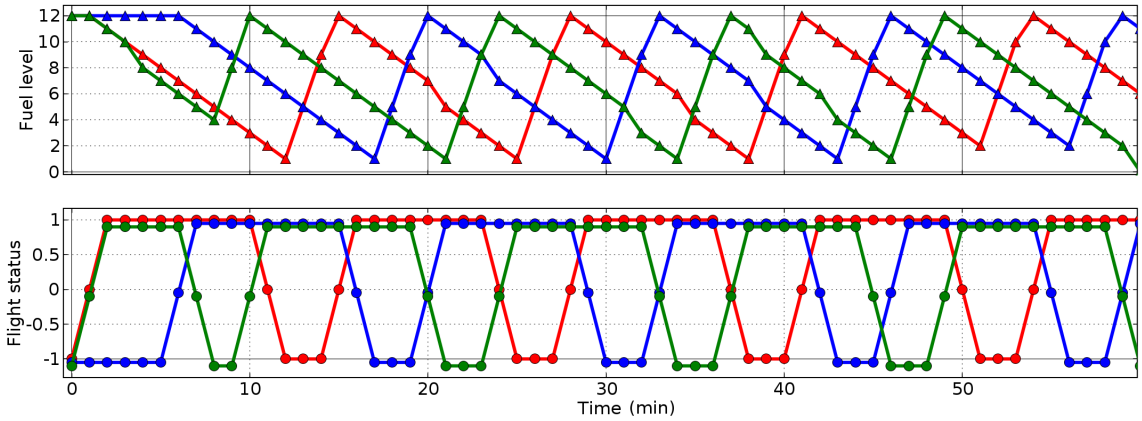


Figure 5-1: Simulation results for $n = 3$, $r = 2$. Note the off-nominal fuel burn events that can be observed in the fuel level plot (these are places where the graph has a slope of -2).

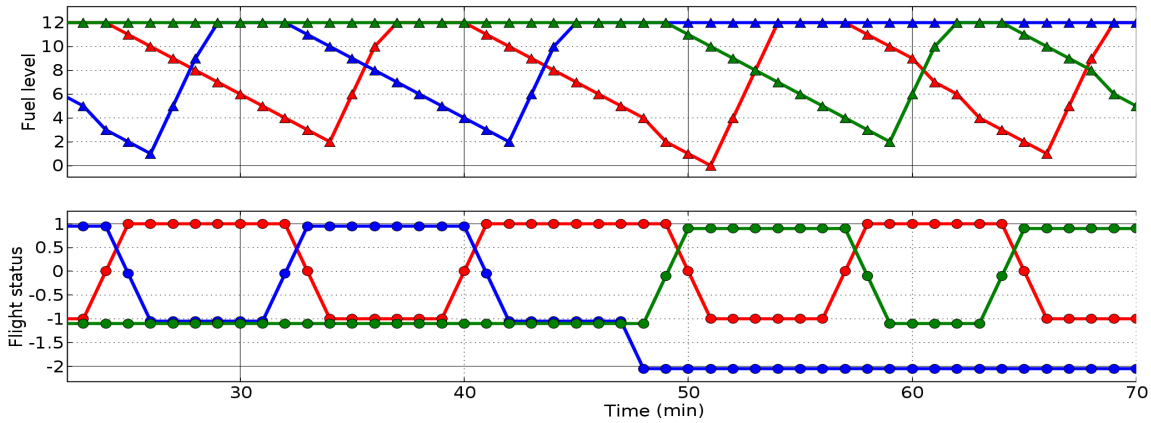


Figure 5-2: Simulation results for $n = 3$, $r = 1$. Response to vehicle crash.

While the controller runs very quickly once $J^\star(\mathbf{x}_k)$ is known, a potentially large amount of computation is required to find $J^\star(\mathbf{x}_k)$. In these experiments, the largest MDP solved was a three-vehicle problem ($n = 3$) with $s = 1$, $F_{max} = 16$, and $\Delta f = 1$, resulting in a state space size (Equation 5.5) of approximately 1.2 million states.

To test the performance of the health-enabled control strategy, $J^\star(\mathbf{x}_k)$ was calculated for several different sets of problem parameters, and the resulting optimal control law $\mathbf{u}^\star(\mathbf{x}_k)$ was found. The system dynamics were then simulated under the action of $\mathbf{u}^\star(\mathbf{x}_k)$. Unless otherwise specified, many of the problem parameters were held fixed at the following values: $Y_s = 1$, $p_{crash} = 0.01$, $p_{f_{nominal}} = 0.90$, $\Delta f = 1$, $\dot{F}_{burn} = 1$, $C_{loc} = 8.0$, $C_{crash} = 50.0$, and $C_f = 1.0$.

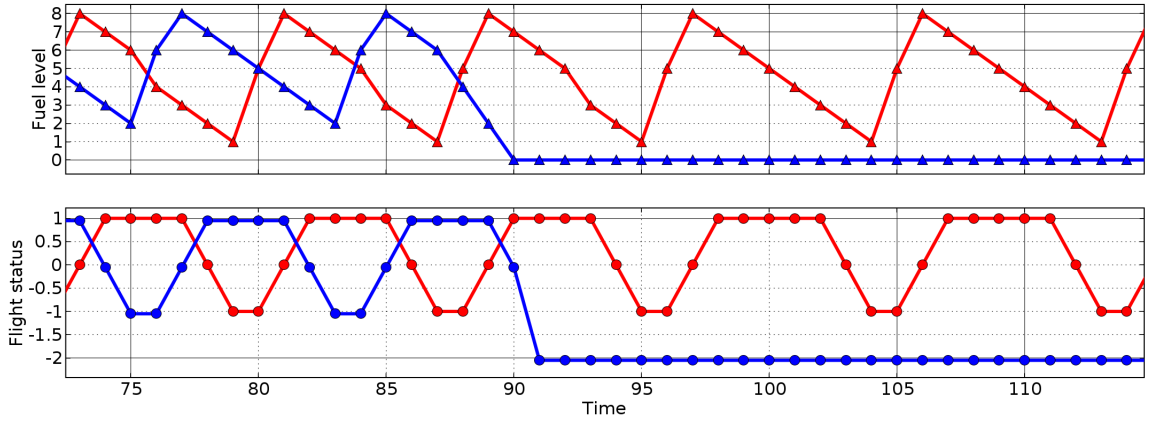


Figure 5-3: Simulation results for $n = 2$, $r = 1$. Response to vehicle crash.

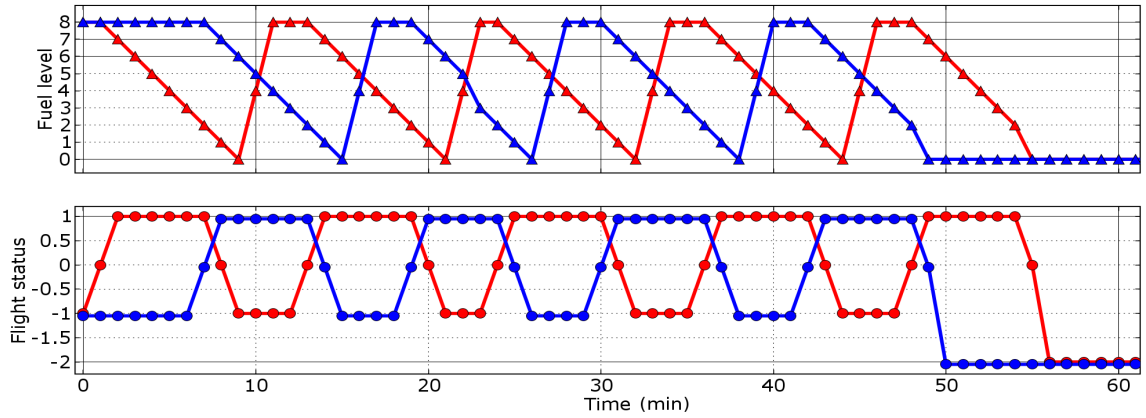


Figure 5-4: Simulation results for $n = 2$, $r = 1$. Control policy based on deterministic fuel burn.

Figure 5-1 shows a simulation result for the three-vehicle ($n = 3$) case with two vehicles requested ($r = 2$). In this experiment, the fuel capacity F_{max} was 12 and the refuel rate $\dot{F}_{refuel}$ was 4. The figure shows the flight status y_i for each of the UAVs in the lower graph (where -2 = crashed, -1 = base location, 0 = transition area, and 1 = surveillance location) and the fuel state f_i for each UAV in the top graph. These results exhibit a number of desirable behaviors. First, note that the system commands two UAVs to take off at time $t = 0$ and fly immediately to the surveillance area to establish initial coverage. If the system were to leave these two UAVs in the surveillance area until both were close to the minimum fuel level needed to return to base, they would both have to leave at the same time, resulting in coverage gap.

However, because the system anticipates this problem, it instead recalls the green UAV to base well before it has reached the minimum fuel level. In addition, it launches the blue UAV at the right moment so that the blue UAV arrives at the surveillance location precisely when the green UAV is commanded to return to base, resulting in continuous coverage throughout the vehicle swap. This initial command sequence allows the system to set up a regular pattern of vehicle swaps which results in the greatest possible coverage. Another desirable feature of the solution is that the system tries to arrange for the UAVs to return to base with a small reserve quantity of fuel remaining. This behavior is a proactive hedge against the uncertainty in the fuel burn dynamics, reducing the probability that the vehicle will run out of fuel before reaching base due to one or more higher-than-average fuel burn events.

Note that in this example, the parameters were chosen so that the ratio of fuel capacity to refueling rate of the UAVs was slightly less than the minimum necessary to maintain perfectly continuous coverage. In other words, it was a "hard" problem where even an optimal solution must exhibit small periodic coverage gaps, such as the one seen at time $t = 20$. In spite of the intentionally challenging problem setup, the system performed extremely well. It is especially interesting to note that, as discussed, the system exhibits a number of distinct and relatively sophisticated behaviors, such as bringing back vehicles early and the hedge against fuel burn uncertainty. These behaviors emerge naturally and automatically due to the formulation and solution method applied to the problem.

A further example is shown in Figure 5-2. This experiment was an "easy" problem where three UAVs ($n = 3$) with high fuel capacity and refuel rates ($F_{max} = 12$, $\dot{F}_{refuel} = 4$) were available to satisfy a single request ($r = 1$). In this example, the system initially sets up a regular switching pattern between only the red and blue UAVs. The green UAV is left in reserve at the base location since it is not needed to satisfy the requested number of vehicles, and commanding it to fly would consume extra fuel (recall that fuel burn is a small part of the cost function $g(\mathbf{x}, \mathbf{u})$). However, at time $t = 47$, the blue UAV suffers a random failure during take off and crashes (this failure was a result of having a nonzero value p_{crash} for the probability of random

crashes, which the system can do nothing to prevent). The system immediately commands the green UAV to take off to replace the blue UAV, while simultaneously leaving the red UAV in the surveillance area for an extra time unit to provide coverage until the green UAV arrives on station. After that point, the switching pattern is maintained using the green and red UAVs. Even though a catastrophic failure occurs in this example, coverage is maintained at 100% at all times.

Another set of simulation results is shown in Figure 5-3. This experiment was a hard problem where the two UAVs had limited fuel capacity ($F_{max} = 8$, $\dot{F}_{refuel} = 4$), and there was a single vehicle request ($r = 1$). In this scenario, vehicle crashes were intentionally difficult to avoid because the off-nominal fuel burn events consumed a significant fraction (25%) of the UAV's fuel. In the figure, we see that the system initially sets up a regular switching pattern for the UAVs, trying to keep the time that each UAV must spend in the surveillance area low to minimize the chances of a crash, while maintaining full coverage at the same time. Again, we see the hedging behavior with respect to the fuel dynamics; the system tries to have the vehicles arrive back at base with 1 or 2 units of fuel left. Despite this hedging behavior, starting at time $t = 87$, the blue vehicle experiences a very "unlucky" series of three off-nominal fuel burn events in a row (the probability of this event is $(1-p_{f_{nominal}})^3 = 0.001$), resulting in a crash. Interestingly, the system responds to this crash by taking the "risk" of leaving the remaining UAV out in the surveillance area for longer (5 time units instead of 4), attempting to provide the maximum amount of coverage. However, it only does this if the fuel consumption while in the surveillance area is nominal continuously; if off-nominal fuel burn occurs, it still brings the vehicle back after only 4 time units. This is another example of a subtle but complex behavior that emerges naturally from the planning formulation.

A final example, shown in Figure 5-4, illustrates the value of the problem formulation presented here, which accounts for the inherent uncertainty in the fuel burn dynamics. In this example, an optimal policy for the purely deterministic problem ($p_{f_{nominal}} = 1$), which does not account for this uncertainty, was calculated. This policy was then simulated using the true, uncertain, dynamics ($p_{f_{nominal}} = 0.90$). The

fuel state graph in Figure 5-4 reveals that the proactive hedging behavior is lost with the policy based on deterministic dynamics; the policy always brings vehicles back to base with exactly zero fuel remaining. Unfortunately, this strategy is highly risky, since any off-nominal fuel burn events which occur when the vehicle is returning to base are guaranteed to produce a crash. In the simulation shown, both vehicles crash after a short period of time. Indeed, in every simulation run to date, both UAVs end up crashing after a short period of time. The reason for this is that for each vehicle cycle, the probability of a crash is $1 - p_{f_{nominal}} = 0.100$, as opposed to the control policy based on uncertain fuel dynamics, which effectively reduces this probability to $(1 - p_{f_{nominal}})^3 = 0.001$. When viewed in this light, the uncertainty-based controller has a clear advantage.

5.4 Extensions of the Basic Problem

The basic persistent surveillance problem discussed to this point can be extended in a number of ways to more accurately model complex, real-world UAV missions. In this section, we extend the formulation to include a communications relay requirement, a failure model of the UAVs' sensors, and the possibility of using heterogeneous teams of UAVs with differing capabilities.

5.4.1 Communications Relay Requirement

The addition of a communications relay requirement is motivated by the fact that in many UAV applications, it is necessary to maintain a communications link between the UAVs performing the mission and a fixed based location. This link may be used by human operators and/or ground-based autonomous planning systems to send commands to the UAVs, or to collect and analyze real-time sensor data from the UAVs. For example, in a search-and-rescue mission with camera-equipped UAVs, a human operator may need to observe the real-time video feeds from each UAV in order to determine probable locations of the party to be rescued. Furthermore, in many cases, the communication range of each UAV may be limited, and in particular

may be less than the distance between base and the surveillance area. Therefore, in these situations, it is necessary to form a communications "chain" consisting of a spatially separated string of UAVs which relay messages back and forth between the base and the surveillance area [60, 61].

In order to model the requirement for establishment of a communications chain in the MDP formulation, the cost function $g(\mathbf{x}, \mathbf{u})$ is modified in the following way. Recall that the form of the cost function is

$$g(\mathbf{x}, \mathbf{u}) = C_{loc} \max\{0, (r - n_s(\mathbf{x}))\} + C_{crash} n_{crashed}(\mathbf{x}) + C_f n_f(\mathbf{x}),$$

where $n_s(\mathbf{x})$ is a function that counts the number of UAVs in the surveillance area, and the term $C_{loc} \max\{0, (r - n_s(\mathbf{x}))\}$ serves to penalize loss of surveillance coverage (i.e. having fewer UAVs in the surveillance area than are needed). To enforce the communications requirement, let $comm(\mathbf{x})$ be a function that indicates whether communications are possible between base and the surveillance area:

$$comm(\mathbf{x}) = \begin{cases} 1 & \text{if communications link exists in state } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}.$$

The functional form of $comm(\mathbf{x})$ should be chosen to reflect the communication range capabilities of each UAV. For example, if the base and surveillance locations are separated by 2 miles, and each UAV has a communication range of 1 mile, then $comm(\mathbf{x})$ should be 1 whenever there is a UAV halfway between the base and the surveillance location (since in this case, this UAV has just enough range to relay communications to both areas). In the results presented in this chapter, we use a communications model of this type for $comm(\mathbf{x})$, assuming that communications are possible anytime a UAV is halfway between base and the surveillance location. Note that more complex communications models can be easily incorporated by simply changing the form of $comm(\mathbf{x})$.

Once the particular form of $comm(\mathbf{x})$ is chosen, it is incorporated into the cost

function $g(\mathbf{x}, \mathbf{u})$ as follows:

$$g(\mathbf{x}, \mathbf{u}) = C_{loc} \max\{0, (r - n_s(\mathbf{x})comm(\mathbf{x}))\} + C_{crash} n_{crashed}(\mathbf{x}) + C_f n_f(\mathbf{x}).$$

Note the only change from the original cost function is through the term $n_s(\mathbf{x})comm(\mathbf{x})$. Thus, whenever a communications link is established, $comm(\mathbf{x})$ is 1 and the cost function behaves as before, penalizing loss of coverage. However, if communications are broken, $comm(\mathbf{x})$ is 0 and any UAVs that are in the surveillance location become useless to the mission since they cannot communicate with base. Therefore, in order to minimize the cost $g(\mathbf{x}, \mathbf{u})$, it is necessary to maintain UAVs in the surveillance area *and* maintain a communications link, as desired.

5.4.2 Sensor Failure Model

In order to perform the surveillance missions of interest in this chapter, UAVs may be equipped with a variety of sensors, such as visible-light cameras, infrared sensors, radars, etc. Of course, these sensors are not totally reliable, and in general may fail at any point during the mission. In order to develop a realistic, health-management problem formulation, it is necessary to account for the possibility of these failures in the MDP model. The qualitative description of our failure model is as follows. We assume that a UAV's sensor may fail at any point during the mission, and that the probability of failure at any given moment is described by a parameter $0 < p_{sensor\ fail} < 1$. When a sensor failure occurs, the UAV becomes useless for performing any tasks in the surveillance area. Note, however, that we assume the failure does not affect the UAV's communication subsystem, so that a UAV with a failed sensor can still perform a useful function by serving as a communications relay. Furthermore, upon returning to base, the UAV's sensor can be repaired.

In order to incorporate this failure model into the MDP, it is necessary to modify the state space $\mathcal{S}$, the system state transition model P, and the cost function $g(\mathbf{x}, \mathbf{u})$. The state space modification is straightforward; the state vector for every UAV is simply augmented with a binary variable that describes whether that UAV's sensor

is failed or not. In particular, the full state vector $\mathbf{x}$ is given by

$$\mathbf{x} = (y_1, y_2, \ldots, y_n; f_1, f_2, \ldots, f_n; s_1, s_2, \ldots, s_n; r)^T,$$

where $s_i \in \{0, 1\}$ describes the sensor state of UAV i. We use the convention that $s_i = 1$ indicates that the sensor is failed, and $s_i = 0$ indicates that it is operational.

Along with the augmented state space, the transition model P must also be updated to reflect the failure dynamics of the sensors. First, whenever a UAV is flying and its sensor is already failed ($s_i = 1$), it remains failed with probability 1; if its sensor is operational ($s_i = 0$), then it fails with probability $p_{sensor\ fail}$ and remains operational with probability $(1 - p_{sensor\ fail})$. Finally, if a UAV returns to the base location ($y_i = Y_b$), then its sensor is restored to operational status ($s_i = 0$) with probability 1.

The final requirement to incorporate the sensor failure model is to update the cost function $g(\mathbf{x}, \mathbf{u})$ to reflect the fact that UAVs with failed sensors are useless for performing tasks in the surveillance area. To do this, the function $n_s(\mathbf{x})$, which previously counted the total number of UAVs in the surveillance area, is modified to instead count the number of UAVs in the surveillance area *with operational sensors*.

5.4.3 Heterogeneous UAV Teams and Multiple Task Types

In the problem formulation developed to this point, it was assumed that all of the UAVs are of the same type and therefore have the same capabilities. As a consequence, there is only one type of UAV that can be requested (by changing the state variable r) to maintain coverage in the surveillance area. In many mission scenarios, this assumption may be too simplistic, and it may be necessary to account for several different classes of UAV types, as well as several different classes of tasks. For example, in a search and tracking mission, there may be two different classes of UAVs available: one class which has sensors and flight capabilities optimized for searching (the first task), and one class which is optimized for tracking (the second task). Depending on the phase of the mission, it may be necessary to be able to individually request

UAVs to accomplish either type of task. Furthermore, in general, the different types of UAVs may each be able to perform several different types of tasks, and some UAVs may perform certain task types better than others. In addition, the different types of tasks may be causally related to each other; for example, a search task may cause a tracking task to be generated when the searching UAV discovers a new object of interest that then must be tracked.

In order to extend the persistent surveillance problem formulation to account for these additional complexities, we first define N_{UAVs} as the total number of different UAV types available to accomplish the mission, and N_{tasks} as the total number of different task types that must be accomplished. To keep track of the number of requested vehicles for each task type, it is necessary to augment the state vector $\mathbf{x}$ with N_{tasks} request variables $r_1, \ldots, r_{N_{tasks}}$, where r_j represents the number of requested vehicles that can perform task type j. Additionally, assuming that there are n_i UAVs of type i ($i = 1, \ldots, N_{UAVs}$), the full state vector is now given by

$$\begin{aligned}
\mathbf{x} \;=\; & (y_{1,1}, \ldots, y_{1,n_1}; \ldots; y_{N_{UAVs},1}, \ldots, y_{N_{UAVs},n_{N_{UAVs}}}; \\
& f_{1,1}, \ldots, f_{1,n_1}; \ldots; f_{N_{UAVs},1}, \ldots, f_{N_{UAVs},n_{N_{UAVs}}}; \\
& s_{1,1}, \ldots, s_{1,n_1}; \ldots; s_{N_{UAVs},1}, \ldots, s_{N_{UAVs},n_{N_{UAVs}}}; \\
& r_1, \ldots, r_{N_{tasks}})^T
\end{aligned}$$

where $y_{i,k}$, $f_{i,k}$, and $s_{i,k}$ are the location, fuel state, and sensor status, respectively, of the k^{th} UAV of type i.

To model the UAVs' varying abilities to accomplish different types of tasks, we define the $N_{UAVs} \times N_{tasks}$ vehicle capability matrix

$$M = \begin{pmatrix} m_{1,1} & \ldots & m_{1,N_{tasks}} \\ \vdots & \ddots & \vdots \\ m_{N_{UAVs},1} & \ldots & m_{N_{UAVs},N_{tasks}} \end{pmatrix}, \tag{5.8}$$

where $m_{i,j} \in [0, 1]$ represents the capability of a UAV of type i to perform a task of type j. Note that setting $m_{i,j} = 0$ implies that a UAV of type i is completely

incapable of performing task type j, while setting $m_{i,j} = 1$ indicates the vehicle is fully capable of performing the task. The cost function $g(\mathbf{x}, \mathbf{u})$ is now modified as follows:

$$g(\mathbf{x}, \mathbf{u}) = C_{loc} W(\mathbf{x}) + C_{crash} n_{crashed}(\mathbf{x}) + C_f n_f(\mathbf{x}),$$

where the function $W(\mathbf{x})$ measures the cost of the best assignment of the UAVs in the surveillance area to the current task list (given by the request vector $(r_1, \ldots, r_{N_{tasks}})^T$). To specify $W(\mathbf{x})$, it is first necessary to define the $N_{UAVs} \times N_{tasks}$ assignment matrix

$$A = \begin{pmatrix} a_{1,1} & \ldots & a_{1,N_{tasks}} \\ \vdots & \ddots & \vdots \\ a_{N_{UAVs},1} & \ldots & a_{N_{UAVs},N_{tasks}} \end{pmatrix},$$

where $a_{i,j} \in \mathbb{Z}^\star$ is a non-negative integer representing the number of UAVs of type i assigned to tasks of type j. In order for A to be a valid assignment matrix, the total number of UAVs of type i assigned to all task types must be equal to $n_s(\mathbf{x}, i)$, the number of UAVs of type i present in the surveillance area:

$$\sum_{j=1}^{N_{tasks}} a_{i,j} = n_s(\mathbf{x}, i) \quad \forall i \in \{1, \ldots, N_{UAVs}\}.$$

$W(\mathbf{x})$ can now be defined by minimizing the total cost of assigning UAVs to tasks:

$$\begin{aligned} W(\mathbf{x}) \quad &= \quad \min_{a_{i,j} \in \mathbb{Z}^\star} \sum_{j=1}^{N_{tasks}} \max\{0, r_j - \sum_{i=1}^{N_{UAVs}} m_{i,j} a_{i,j} comm(\mathbf{x})\} \\ \text{subj. to} \quad &\sum_{j=1}^{N_{tasks}} a_{i,j} = n_s(\mathbf{x}, i) \quad \forall i \in \{1, \ldots, N_{UAVs}\} \end{aligned} \tag{5.9}$$

The optimization problem (5.9) is an integer programming problem similar to those found in many task assignment frameworks [3–5, 42, 45, 86]. When the number of UAVs in the surveillance area and tasks is not too large, (5.9) can be solved by enumerating the possible assignments. For larger problems, it can be solved using more sophisticated techniques presented in the references above.

As a final modification, it may be desirable to model the causal relationship between different task types, since certain types of tasks may naturally follow each other, cause events that lead to other tasks becoming unnecessary, etc. A natural way to model the relationships is by defining a Markov chain whose state space is the set of possible request vectors $(r_1, \ldots, r_{N_{tasks}})^T$. We assume that each task type $i \in [1, \ldots, N_{tasks}]$ is associated with a set of "creation" probabilities $\rho^+_{i,j}$ and "completion" probabilities $\rho^-_{i,j}$, where $j \in [1, \ldots, N_{tasks}]$. In particular, $\rho^+_{i,j}$ denotes the probability that, given a single task of type i that is currently active, a new task of type j will become active in the subsequent time step (in other words, r_j will increase by one). Similarly, $\rho^-_{i,j}$ denotes the probability that, given a single task of type i that is currently active, a currently active task of type j will be completed in the subsequent time step (in other words, r_j will decrease by one). The state transitions of the Markov chain are then formed by taking all possible creation and completion events that are possible for a given set of active tasks (denoted by the request vector $(r_1, \ldots, r_{N_{tasks}})^T$), and computing the probabilities of each event using the ρ values. The resulting Markov chain describes the (probabilistic) evolution of the request vector over time.

Notice that prior to making this modification, the request vector was assumed to be a constant; the new modification allows the request vector to change dynamically over time, via state transitions that are uncontrollable by the planner. Thus, in order to achieve optimal performance, the planner must anticipate the types and numbers of tasks that are likely to arise in the future, and allocate resources accordingly in order to meet this demand.

5.5 Summary

Health management in multi-agent systems is a complex and difficult problem, but any autonomous control system that is *health-aware* should be proactive (capable of using a model of the system to anticipate the future consequences of randomly-occurring failures or off-nominal events) as well as capable of exploiting the interde-

pendencies between agents at the group level. For this reason, MDPs are a natural framework for posing the health management problem. The basic persistent surveillance problem formulation demonstrates how a suitable MDP may be formulated to capture the important health-related issues in a problem, and simulation results of the basic problem demonstrate how desirable, proactive, group-aware behaviors naturally emerge when the MDP is solved.

While the basic persistent surveillance problem primarily focuses on group fuel management, the extensions to the basic problem capture further important aspects of complex, real-world UAV missions, including the need to deal with randomly-occurring sensor failures, communication constraints, and heterogeneous teams of UAVs with different capabilities. Computationally, this formulation is significantly more complex than the basic persistent surveillance problem, having both a larger state space and more complex dynamics. While the basic problem with only 3 UAVs has a state space of approximately 1.2 million states and can be solved exactly using value iteration, this solution approach quickly becomes intractable for larger, possibly heterogeneous UAV teams using the extended problem formulation. In Chapter 7, we describe how running the BRE approximate dynamic programming algorithms on a parallel, distributed computer cluster can be used to solve problems with very large state spaces (on the order of 10^{12} states) while maintaining near-optimal mission performance. Furthermore, we demonstrate performance benefits of our approach over a deterministic planning approach that does not account for randomness in the system dynamics model.

Chapter 6

Adaptive MDP Framework

As discussed in Chapter 5, MDPs are a powerful approach for addressing multi-agent control and planning problems, especially those in which the health state of the agents may be subject to stochastic dynamics caused by failures, unknown environmental effects, etc. The persistent surveillance MDP formulation developed in Chapter 5 relies on having an accurate estimate of the system model (i.e. state transition probabilities) in order to guarantee optimal behavior of the control policy. When solving the MDP offline (i.e. before the true system begins operating), inaccuracies in the system model can result in suboptimal performance when the control policy is implemented on the real system. A further issue is that that, throughout the course of actual UAV missions, it is likely that the parameters of the MDP will be time-varying, due to unmodeled changes in the system dynamics over time. For example, in the persistent surveillance problem, the probability of sensor failures occurring may slowly increase over time as the UAVs accumulate wear and tear.

The general problem of parametric uncertainty in MDPs has been investigated in [82, 115], which showed that robust MDP problems can be solved by developing robust counterparts to Value Iteration and Policy Iteration. In [115], the authors also showed that robust MDPs could be solved if scenarios—samples from a prior distribution defined over the set of possible models—were available as a description for the parametric uncertainty. Sampling-based solutions to Partially Observable Markov Decision Processes (POMDPs) with parametric uncertainty have been proposed in

[84]. Recent work [27] has extended sampling-based robust MDPs by proposing a new technique that requires far fewer total samples to achieve a robust solution. If the model parameters are time-varying, common solution methods such as those used in model-based adaptation [64, 85] can be used in updating the model.

The focus of this chapter is on implementation of an MDP-based control architecture that addresses the challenges of poorly known and/or time-varying models. Since the overall system may exhibit model changes in real-time, online adaptation is a necessary requirement for real-life UAV missions. Our adaptive MDP approach is similar in spirit to ideas from adaptive control [12, 15], in that the system model is continuously estimated, and an updated policy (i.e., the control law) is computed at every time step. However, unlike adaptive control, the system in question that must be controlled is modeled by a MDP, whose dynamics may be stochastic in the general case. An alternative approach to dealing with model uncertainty in this context would be an offline, minimax strategy, where the "worst case" model is assumed and the corresponding control policy implemented [49]. However, this approach can lead to overly conservative policies that do not achieve sufficient levels of performance if the true system model is different than the worst-case scenario. The flight results that will be presented in Chapter 7 demonstrate that the adaptive MDP architecture can achieve significant performance benefits over offline, minimax-like strategies. In addition, we shall demonstrate that the adaptive architecture is effective for dealing with changing system models such as vehicle degradation or damage.

Two factors influence the responsiveness (i.e. the speed with which an updated policy will begin to execute in response to a change in the system model) of the adaptive MDP architecture. First, the time needed to recompute the optimal policy, given a new model estimate, is clearly a factor in how soon the system will respond to the new model. In this chapter, we develop an approach that utilizes previous cost-to-go solutions in a "bootstrapping" fashion to reduce the time needed to recompute the policy. Any approximate dynamic programming technique, such as the BRE techniques developed in Chapters 3 and 4 can be leveraged in the bootstrapping framework in order to quickly recompute the policy. Second, the rate of convergence

of the model estimator also influences the responsiveness, since if a change in the model takes a long time to be detected, then the execution of the optimal policy will be delayed. Unfortunately, classical estimators [64, 85, 87], can be slow to respond to such changes. To avoid this slow response, we show the benefits of using a modified (discounted mean-variance) formulation to speed up the response of the estimator, and in turn improve the response time of the optimal policy [26].

6.1 Parametric Uncertainty in the Persistent Surveillance Problem

The problem of online MDP adaptation deals with situations in which the system dynamics model exhibits parametric uncertainty. That is, the model is governed by a set of parameters Ω which may not be well-known before the system begins operation, and may vary with time. The optimal cost-to-go function and policy of the MDP shall be denoted by $J^\star_\Omega$ and $\mu^\star_\Omega$, respectively, to emphasize their dependence on the parameter values. A concrete example of parametric uncertainty is in the case of the testing and implementation of a completely new system, where system identification may be needed to quantify the values of the parameters. Another example is one where the parameters may be evolving over time, and need to be estimated on-line. For the purposes of discussion, throughout this chapter, we will use the basic persistent surveillance problem described in Chapter 5 as an example, and our primary parameter of interest will be the nominal fuel flow transition probability p_{nom}. This parameter may be uncertain because we do not fully know the characteristics of the UAVs, or it may change during the course of the mission as the vehicles are damaged in flight, for example. Extending the adaptation approach to include other parameters in addition to the fuel flow probability is straightforward, and more complex adaptation scenarios will be presented in the flight results of Chapter 7.

The optimal policy and other characteristics of the persistent surveillance mission are very sensitive to the precise value of the parameter p_{nom}. Figure 6-1 demonstrates

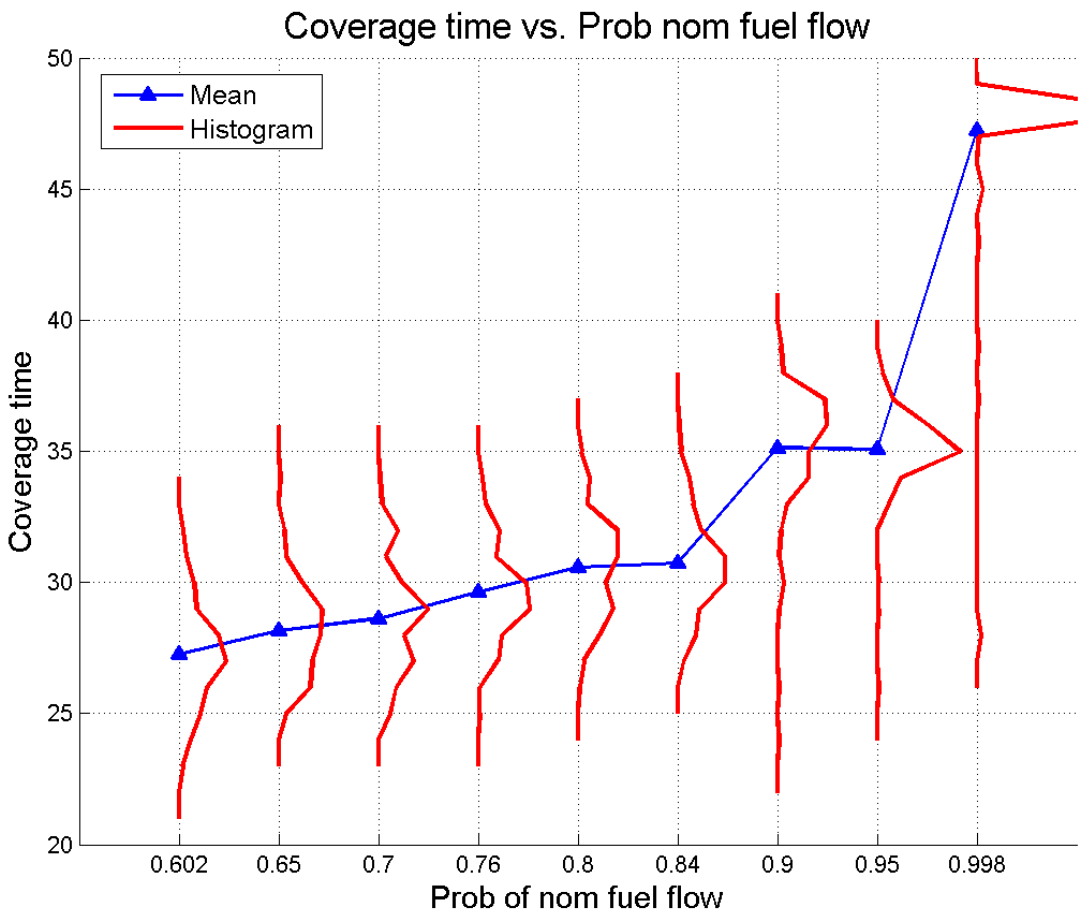


Figure 6-1: Persistent surveillance coverage time (mission length: 50) as a function of nominal fuel flow transition probability p_{nom}.

the sensitivity of the coverage time of the mission (the total number of time steps in which a single UAV was at the surveillance location) as a function of p_{nom}. For values of $p_{nom} < 0.9$, typical coverage times for a 50-time step mission can range from 25 to 30 time steps, while for values of $p_{nom} > 0.9$, the coverage times can increase to almost 47 time steps.

Figure 6-2 shows the impact of a mismatched transition model on the overall mission coverage times. For each pair of "Modeled" and "Actual" values of p_{nom} given in the figure, the total coverage time was evaluated as follows. First, a control policy based on the "Modeled" value was computed exactly, using value iteration. Then, the system was simulated several times, under the control of this policy, for the true

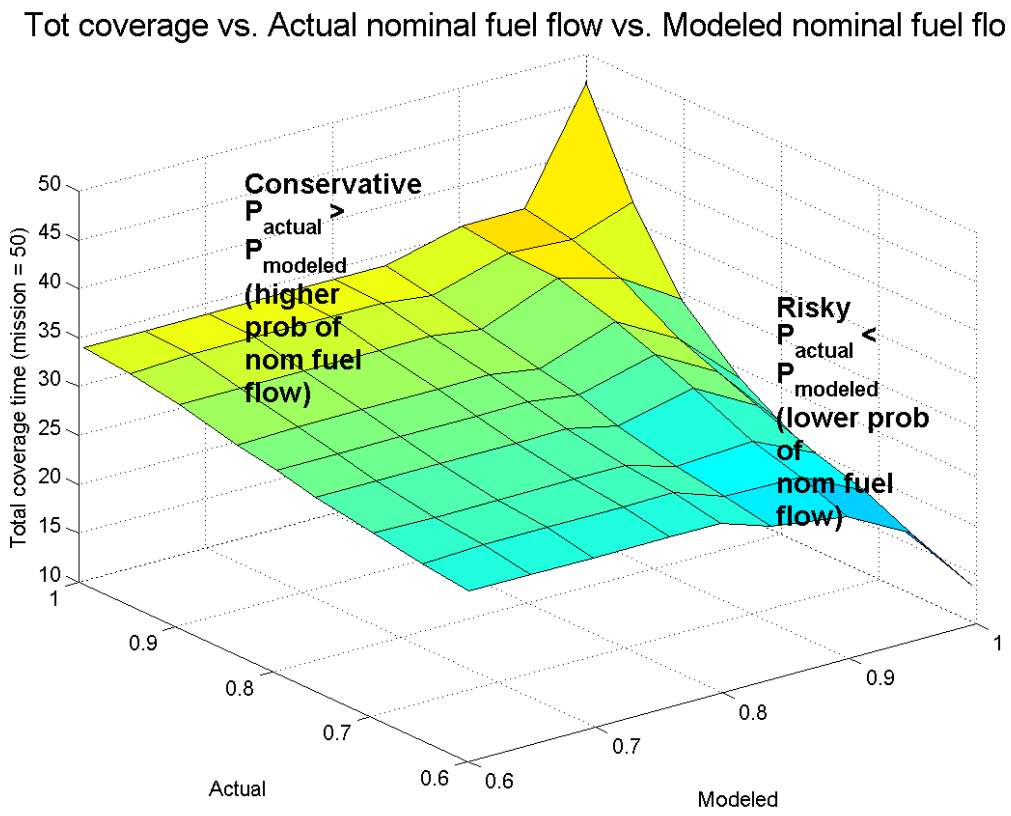


Figure 6-2: Sensitivity of persistent surveillance coverage time to modeling errors in the nominal fuel flow transition probability p_{nom}.

system whose p_{nom} was given by the "Actual" value. The resulting (averaged) mission coverage was then computed. Examining Figure 6-2, there are a number of interesting results. When the modeled p_{nom} is less than the actual p_{nom}, more conservative policies result: the control policy recalls the UAVs to base well before they are out of fuel, because it (incorrectly) assumes they will use a lot of fuel on the flight back to base. This results in fewer crashes, but also leads to decreased surveillance coverage since the vehicles spend less time in the surveillance area. Conversely, riskier policies are the result when the modeled p_{nom} is greater than the actual p_{nom}, since the control policy assumes the UAVs can fly for longer than they actually are capable of. This leads to significant coverage losses, since the UAVs tend to run out of fuel and crash more frequently.

The results presented in this section provide quantitative evidence of the impor-

tance of running a control policy that is based on an accurate model of the true system. The following section will develop an adaptive architecture that addresses the problem of model mismatch by updating the model online, and using this information to recompute the policy.

6.2 Development of the Adaptive Architecture

The prior results showed that value of the parameter p_{nom} has a strong effect on the optimal policy, and in particular, how mismatches between the true parameter value and the value used to compute the optimal policy can lead to degraded performance when implemented in the real system. Therefore, in order to achieve better performance in the real system, some form of adaptation mechanism is necessary to enable the planner to adjust the policy based on observations of the true parameter values. These observations cannot be obtained prior to the start of operation of the real system, so this adaptation must be done online.

The system architecture shown in Figure 6-3 was designed to achieve the goal of online policy adaptation. The architecture consists of two concurrent loops. In the control loop (top), the policy executor receives the current system state i, computes the corresponding control decision $\mu^{\star}_{\hat{\Omega}}(i)$, and applies this decision to the system. This part of the architecture is nearly identical to how a standard, static MDP control policy would be implemented. The single, important difference is that the policy executor receives periodic updates to its current policy $\mu^{\star}_{\hat{\Omega}}$ via the estimation/adaptation loop. In this loop, the parameter estimator receives state transition observations of the controlled system and uses this information to update its estimate of the system parameters. This estimate is then sent to the online MDP solver, which computes a new optimal policy based on the estimate. The updated policy is then transmitted to the policy executor. In this fashion, the policy being used to control the system is continually updated to reflect the latest knowledge of the system parameters. The two key elements of the adaptive MDP architecture are the parameter estimator and the online MDP solver. The next two sections discuss how each of the elements works.

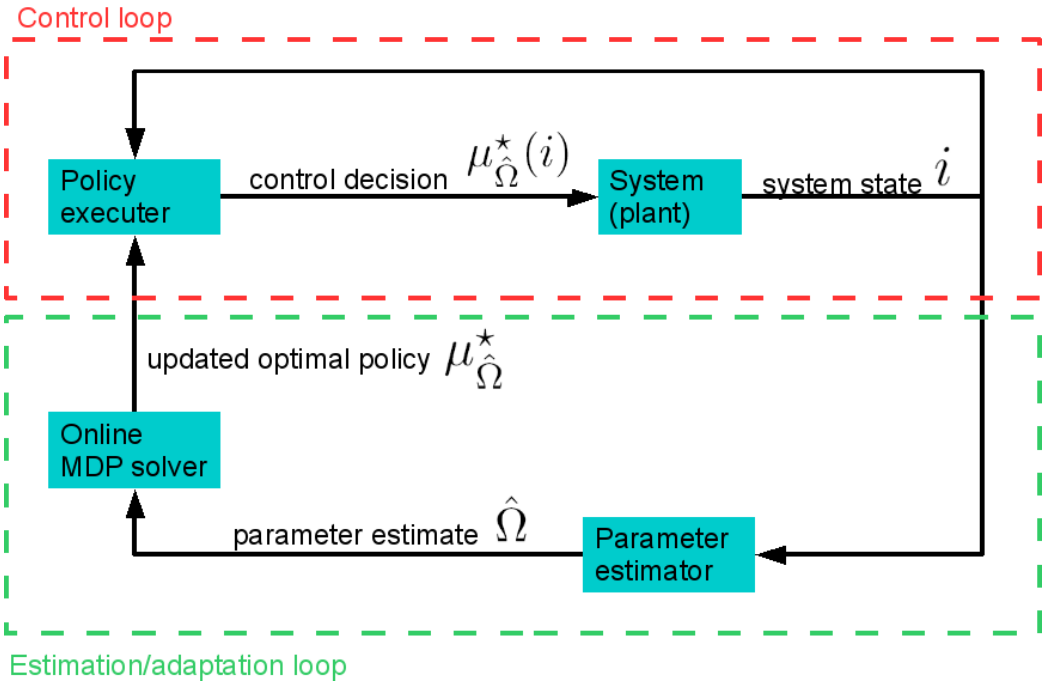


Figure 6-3: Adaptive MDP architecture, consisting of concurrent estimation/adaptation and control loops.

6.2.1 Parameter Estimator

When starting with a new vehicle that has not been extensively tested, the true probability p_{nom} may not be well known, and thus must be effectively identified. A very effective method to handle an uncertain probability, and develop and adaptation scheme provides a prior on this probability, and updates this prior with observed transitions as the mission progresses. For this problem, a reasonable prior is the Beta density (Dirichlet density if the fuel is discretized to 3 or more different burn rates), given by

$$f_B(p \mid \alpha) = K\ p^{\alpha_1 - 1}(1-p)^{\alpha_2 - 1}$$

where K is a normalizing constant that ensures $f_B(p \mid \alpha)$ is a proper density, and $(\alpha_1,\ \alpha_2)$ are parameters of the density that can be interpreted as prior information that we have on p_{nom}. For example, if 3 nominal transitions and 2 off-nominal transition had been observed, then $\alpha_1 = 3$ and $\alpha_2 = 2$. The maximum likelihood estimate

of the unknown parameter is given by

$$\hat{p}_{nom} = \frac{\alpha_1}{\alpha_1 + \alpha_2}$$

A particularly convenient property of the Beta distribution is its conjugacy to the Bernoulli distribution meaning that the updated Beta distribution on the fuel flow transition can be expressed in closed form. If the observation are distributed according to

$$f_M(p \mid \gamma) \propto p^{\gamma_1 - 1}(1-p)^{\gamma_2 - 1}$$

where γ_1 (and γ_2) denote the number of nominal (and off-nominal, respectively) transitions, then the posterior density is given by

$$f_B^+(p \mid \alpha^{'}) \propto f_B(p \mid \alpha)\ f_M(p \mid \alpha)$$

By exploiting the conjugacy properties of the Beta with the Bernoulli distribution, Bayes Rule can be used to update the prior information α_1 and α_2 by incrementing the counts with each observation, for example

$$\alpha_i^{'} = \alpha_i + \gamma_i \quad \forall i$$

Updating the estimate with $N = \gamma_1 + \gamma_2$ observations, results in the following Maximum A Posteriori (MAP) estimator

$$\textbf{MAP:} \quad \hat{p}_{nom} = \frac{\alpha_1^{'}}{\alpha_1^{'} + \alpha_2^{'}} = \frac{\alpha_1 + \gamma_1}{\alpha_1 + \alpha_2 + N}$$

The MAP estimator and is asymptotically unbiased, and we refer to it as the undiscounted estimator in this chapter. Recent work [26] has shown that probability updates that exploit this conjugacy property for the generalization of the Beta, the Dirichlet distribution, can be slow to responding to changes in the transition probability, and a modified estimator has been proposed that is much more responsive to time-varying probabilities. One of the main results is that the Bayesian updates on

the counts α can be expressed as

$$\alpha_i^{'} = \lambda\ \alpha_i + \gamma_i \quad \forall i$$

where $\lambda < 1$ is a discounting parameter that effectively fades away older observations. The new estimate can be constructed as before

$$\textbf{Discounted MAP:} \quad \hat{p}_{nom} = \frac{\lambda\alpha_1 + \gamma_1}{\lambda(\alpha_1 + \alpha_2) + N}$$

Figure 6-4 shows the response of the MAP estimator (blue) compared to the Discounted MAP (red) to a step change in a reference transition probability (shown in black), for $\lambda = 0.8$. The response speed of the discounted MAP is almost 10 times as fast as that of the undiscounted MAP. This implies that the optimal policy obtained with a discounted MAP estimator converges to the optimal policy much quicker, due to the fast estimator convergence.

6.2.2 Online MDP Solver

The online MDP solver in the architecture described in the previous section relies on the ability to rapidly re-solve the MDP as new parameter estimates become available. If the new solution cannot be computed quickly, then the advantages of the adaptive architecture - namely, the ability to adapt to unknown and/or changing system parameters - are lost. Furthermore, solving an MDP "from scratch" without any prior knowledge of the optimal cost or policy typically requires a large amount of computational effort and time. If the online MDP solver attempted to solve each new problem from scratch, it is unlikely that the new solution could be found quickly enough to be useful.

Fortunately, the following empirical observation can allow the online MDP solver to converge to a new solution much faster than it could by solving each MDP instance from scratch. The observation is that the optimal cost function from a previously-obtained solution (i.e. a solution for a different set of parameter values) is "close

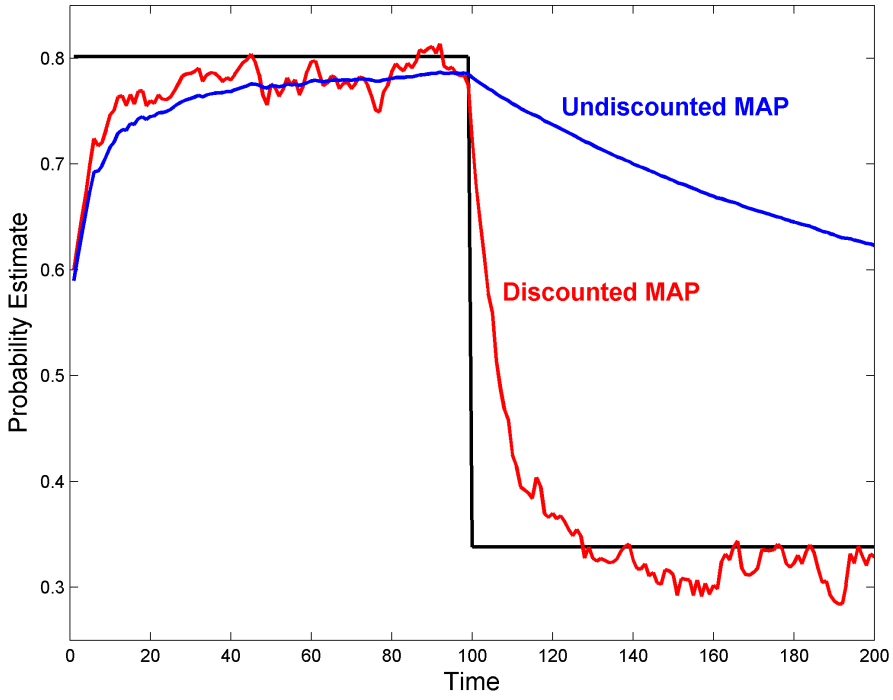


Figure 6-4: Response rate of the discounted MAP with $\lambda = 0.8$ (red) is almost 10 times faster than the undiscounted MAP (blue).

to" the optimal cost of the new problem in many cases. By initializing the solution process with a previously-obtained cost function ("bootstrapping"), it may be possible to converge to the new solution much more quickly.

A visualization of this idea is shown in Figure 6-5. In the figure, the axes represent the space of possible cost functions. The optimal cost for a problem with parameters Ω is shown as a black dot and labeled by $J^\star_\Omega$. The optimal cost for a problem with a slightly different set of parameters $\Omega + \Delta\Omega$ is shown as a gray dot and labeled by $J^\star_{\Omega+\Delta\Omega}$. These two costs are close to each other in the cost space. The progression of the value iteration solution algorithm is represented by the red, green, and blue trajectories, with each iteration of the algorithm shown as a small blue dot. Initializing value iteration with no prior knowledge corresponds in the diagram to starting at the origin (i.e. $J = 0$), as the red and green trajectories do. The number of iterations necessary for value iteration to find the optimal policy corresponds roughly to the distance that must be traversed in the cost space, so finding both $J^\star_\Omega$ and $J^\star_{\Omega+\Delta\Omega}$

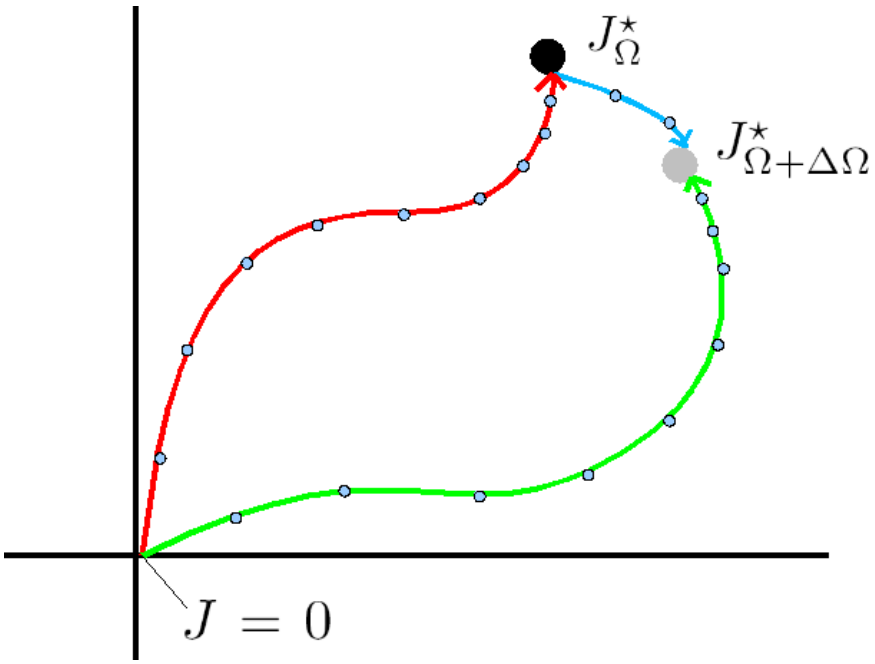


Figure 6-5: Visualization of the MDP solution "bootstrapping" process. The number of value iterations (and therefore, the time required) to find the cost of the perturbed problem, $J^\star_{\Omega+\Delta\Omega}$, is typically much smaller by starting from a previously computed solution $J^\star_\Omega$ as opposed to starting from scratch (i.e. $J = 0$). This is represented in the figure by the bootstrapping path (blue) being shorter than the non-bootstrapped path (green).

along the red and green trajectories, respectively, requires many iterations. However, if the previous solution $J^\star_\Omega$ is already known and the solution $J^\star_{\Omega+\Delta\Omega}$ is desired, value iteration may be initialized at the currently known solution and may thus converge to $J^\star_{\Omega+\Delta\Omega}$ more quickly, as represented by the blue trajectory.

A series of bootstrapping experiments were conducted using the persistent mission MDP as an example problem. In these experiments, the probability of nominal fuel usage p_{nom} was initialized at 0.9, and value iteration was run, starting from $J = 0$. At each iteration, the Bellman error $||TJ - J||$ was recorded and a test was run to detect whether the policy had converged to the optimal policy. Then, p_{nom} was set to a different value, and value iteration was begun again. However, this time value iteration was initialized with the optimal cost function for $p_{nom} = 0.9$. Again, the Bellman error and policy convergence were recorded for each iteration. Results are shown in Figures 6-6, 6-7, 6-8, and 6-9, which show the log of the Bellman error as a function

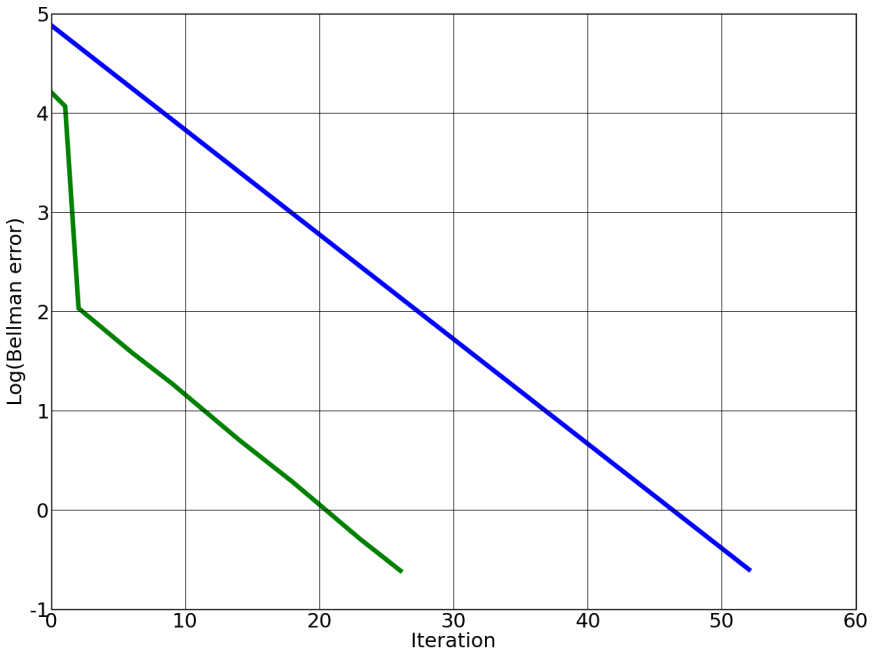


Figure 6-6: Value iteration bootstrapping (VIB) with $p_{nom} = 0.97$

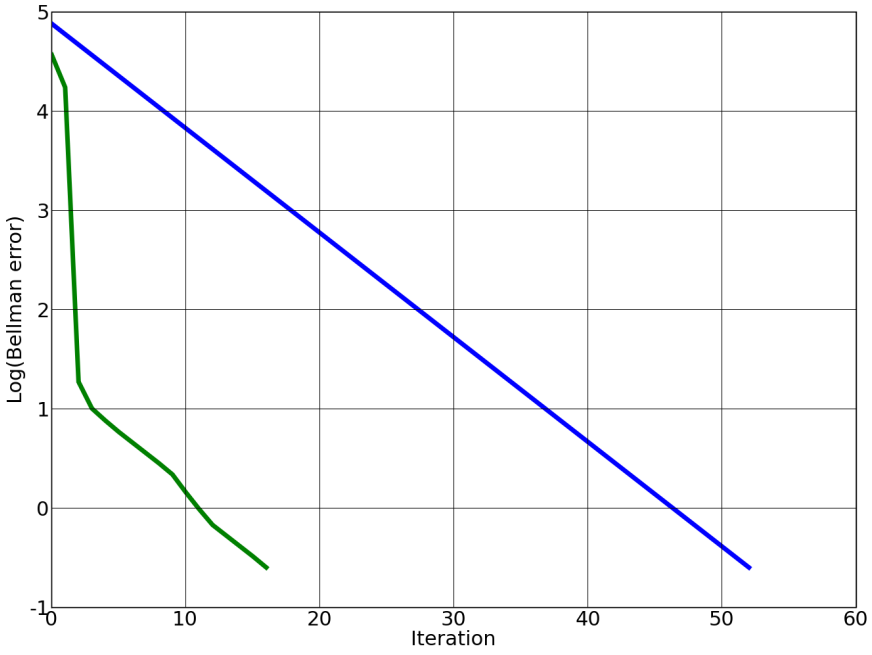


Figure 6-7: VIB with $p_{nom} = 0.8$

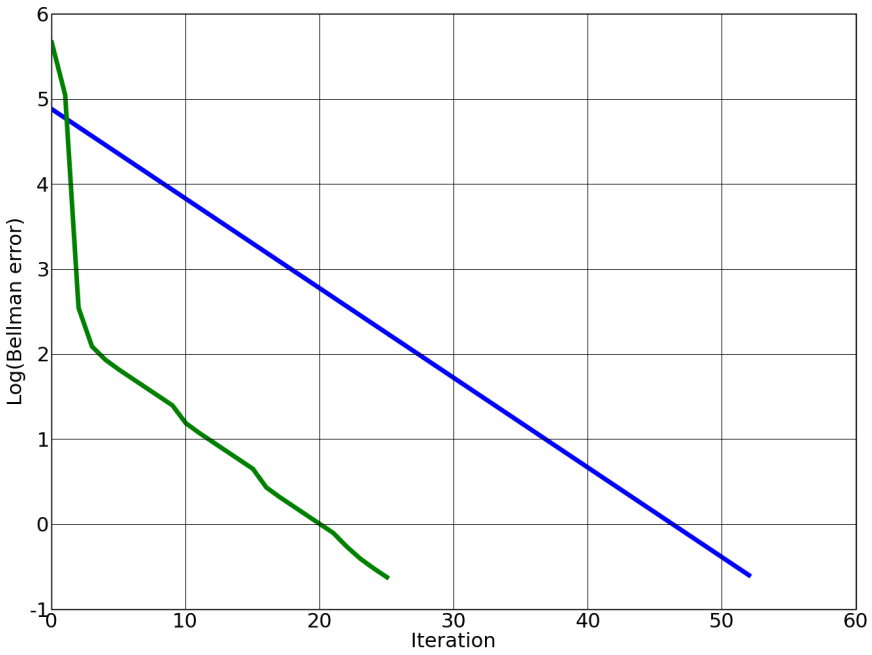


Figure 6-8: VIB with $p_{nom} = 0.6$

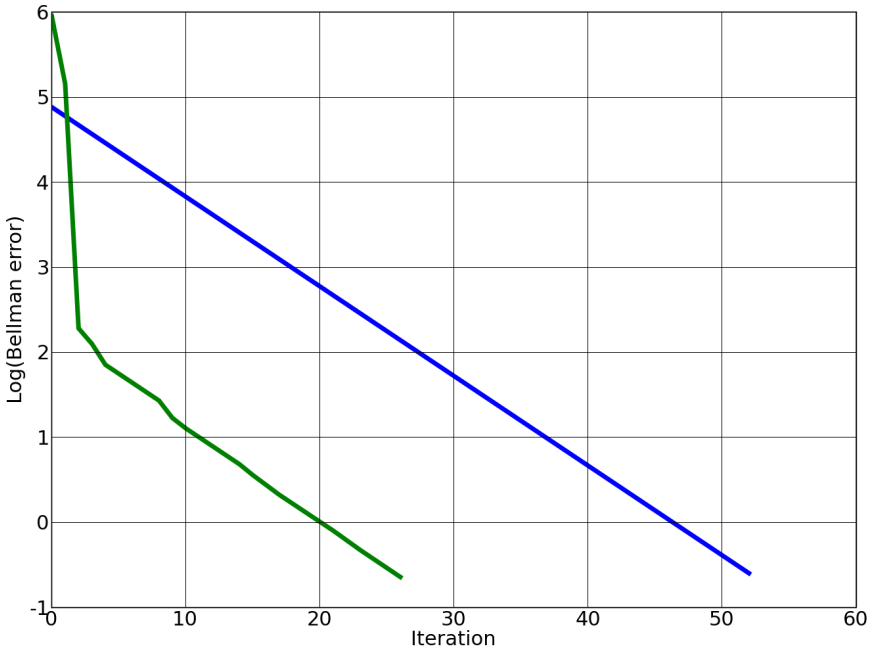


Figure 6-9: VIB with $p_{nom} = 0.5$

of iteration number for bootstrapped value iteration (shown in green), as well as non-bootstrapped value iteration for $p_{nom} = 0.9$. In all cases, bootstrapped value iteration converged much more quickly than non-bootstrapped value iteration. Indeed, the policy converged to the optimal policy within at most 5 iterations for bootstrapped value iteration, compared with more than 40 for non-bootstrapped value iteration. Our online MDP solver uses this bootstrapping technique to quickly recompute the new optimal policy when an updated model estimate arrives from the parameter estimator. In the current implementation, each value iteration for the two vehicle problem takes around 5 seconds and the policy is executed every 30 seconds. Therefore, the bootstrapping process makes it possible to continuously recompute the optimal policy in time for the next scheduled policy execution. Without bootstrapping, the solution process would be too slow (around 10 minutes) to accomplish this goal.

The bootstrapping procedure presented here is conceptually similar to prioritized sweeping methods for executing value iteration. Prioritized sweeping was originally proposed by Moore and Atkeson [109] and subsequently extended by a number of authors [8, 90, 120, 168, 170]. The basic idea behind prioritized sweeping is to update the values of states in an efficient order. In order to do this, prioritized sweeping maintains a queue of states, sorted in order of the states whose values are likely to change significantly in the next update. By updating the values of states in the order they are stored in the queue, the computational effort of carrying out value iteration is focused efficiently in areas of the state space where large changes of the value function are occurring. Like the value iteration bootstrapping procedure, prioritized sweeping generally leads to fast convergence of value iteration.

Unfortunately, in order to address larger problems, the computational savings generated by the bootstrapping procedure, prioritized sweeping, or any other exact solution approach, are likely to be insufficient to recompute the policy quickly enough to be useful. This is because, for problems with very large state spaces, even a single iteration of value iteration may be computationally intractable. However, the adaptive architecture presented in this chapter can be used in conjunction with any approximate dynamic programming method to further reduce the policy re-computation time

when exact methods (such as value iteration or policy iteration based methods) are impractical. Notice that, regardless of the algorithm that is used to recompute the policy, the key boostrapping observation remains valid: the cost-to-go (and therefore the policy) of the system under the old set of model parameters Ω is likely to be close to the cost-to-go of the system under the new parameters $\Omega + \Delta\Omega$. Therefore, any approximate dynamic programming algorithm can be bootstrapped by initializing it using the policy and cost function computed for the previous set of parameters. In particular, results presented in Chapter 7 will demonstrate the use of the BRE algorithm in the bootstrapping framework to quickly recompute the policy for a large and complex persistent surveillance problem. In this problem, the size of the state space is so large (around 10^{12}) that solving it using any exact method is intractable.

6.3 Summary

This chapter presented a framework for continuously estimating the dynamic model of a Markov decision process and adapting the policy on-line. This framework is useful in the cases where the initial model is poorly known and where the true model changes as the system is operating. Analysis of the persistent surveillance problem demonstrates the detrimental impact of modeling mismatches. The results of Chapter 7 will show that the adaptation approach can mitigate these effects even in the presence of a poorly known initial model and dynamic model changes, and furthermore, that the adaptive approach yields better performance over offline, minimax type approaches, which must trade-off performance versus robustness.

Chapter 7

Simulation and Flight Results

In order to validate the performance and applicability of the multi-UAV, health-aware persistent surveillance problem formulation, the adaptive MDP architecture, and the BRE approximate dynamic programming algorithms presented so far in the thesis, a series of experiments were conducted both in simulation and in a realistic flight test environment. This chapter presents these simulation and flight results.

7.1 Background and Experimental Setup

This section discusses details of the experimental setup, including the flight test facility, autonomous mission planning architecture, and implementation of the BRE algorithms.

7.1.1 RAVEN Flight Test Facility

The flight experiments presented in this chapter were conducted in the MIT Real-time indoor Autonomous Vehicle test ENvironment (RAVEN). RAVEN allows for rapid prototyping and testing of a variety of unmanned vehicle technologies, such as adaptive flight control [106, 107], automated UAV recharging [51], autonomous UAV air combat [105], and coordinated multi-vehicle search and track missions [30, 31, 33], in a controlled, indoor flight test volume. RAVEN utilizes a camera-based motion-

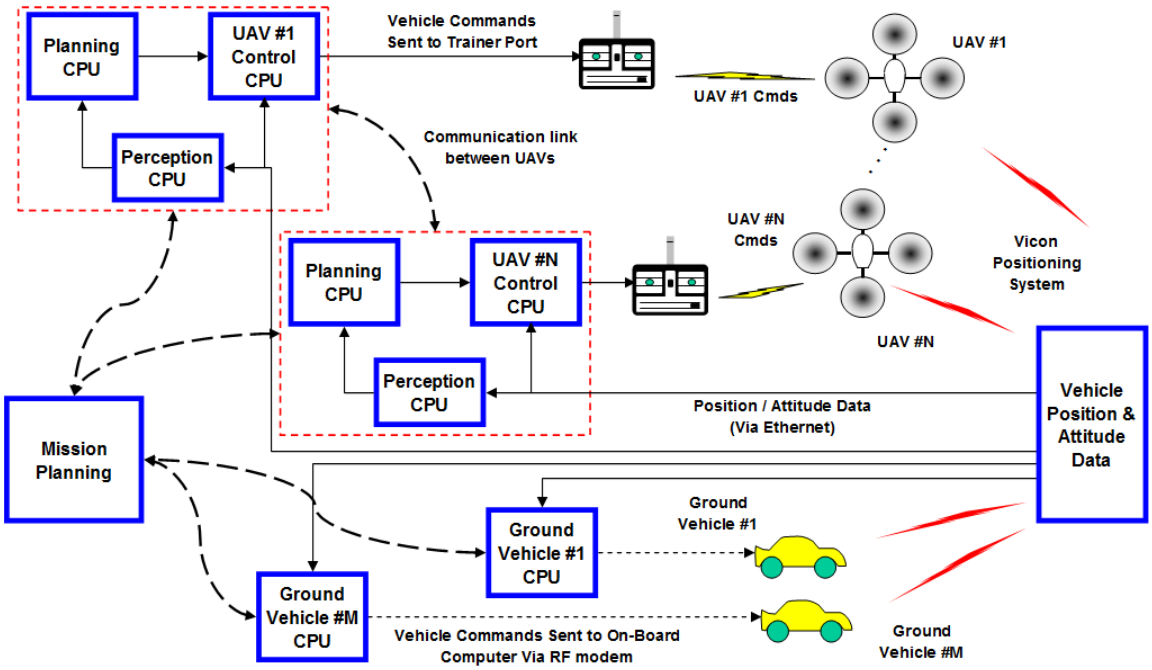


Figure 7-1: Distributed planning, perception, and control processing architecture for RAVEN [80].

capture system [165] to simultaneously track multiple air- and ground-based vehicles, and provide highly accurate position and orientation information about these vehicles in real-time. This information is then distributed to a group of command and control computers responsible for managing the autonomous execution of the mission. The RAVEN system architecture shown in Figure 7-1, which highlights the perception, planning, and control processing components. Figures 7-2 and 7-3 show the general layout of the RAVEN facility and an experiment demonstrating simultaneous control of 10 UAVs, respectively. For more details about RAVEN, see [80].

7.1.2 Multi UxV Planning and Control Architecture

The persistent surveillance problem formulation plays an important role in long-term, multi-UAV planning systems: namely, it solves the problem of how to robustly maintain a set of assets on-station and available to carry out the intended mission, even in the face of randomly-occurring failures and other adverse conditions. However, in order to achieve the goals of the mission, this persistency planning capability must

Figure 7-2: The RAVEN flight test facility.

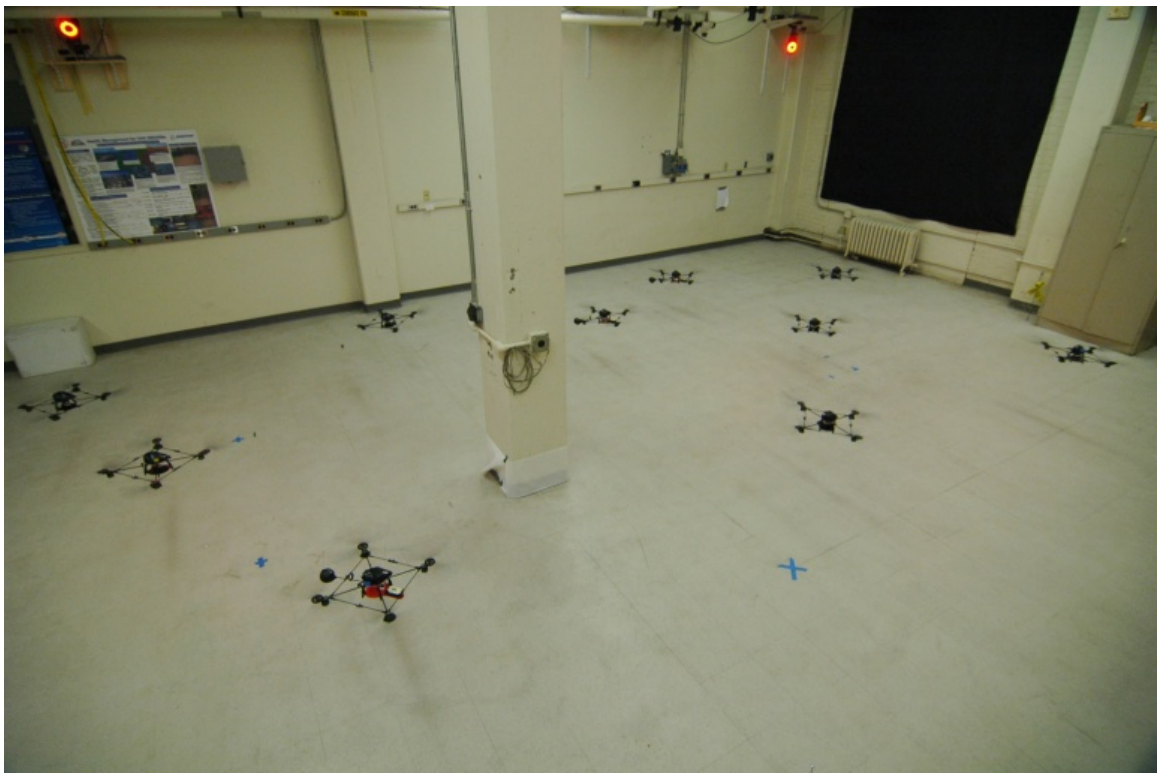


Figure 7-3: Simultaneous flight of 10 UAVs in RAVEN.

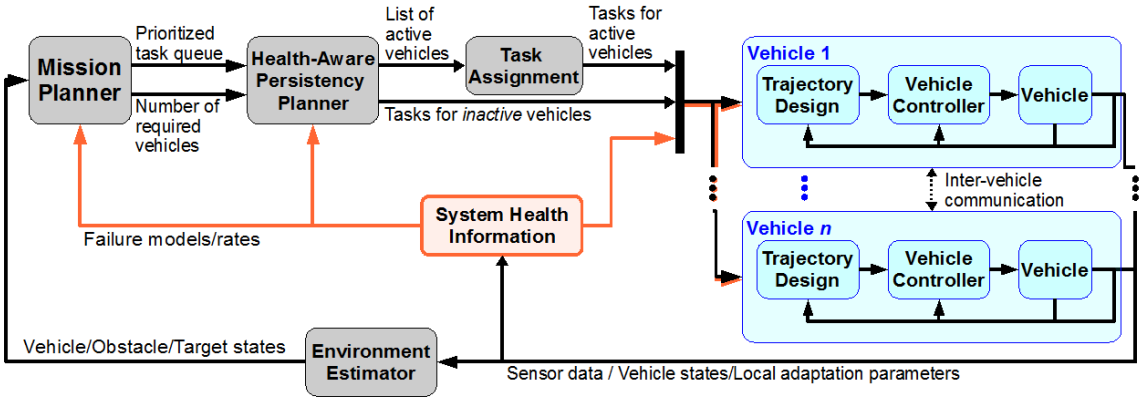


Figure 7-4: Multi-UxV planning and control architecture used in the heterogeneous persistent surveillance flight experiments.

be integrated into a larger planning and control architecture that also performs other important functions, such as generating tasks that will lead to successful completion of the mission, assigning these tasks to the on-station UAVs, and maintaining low-level stabilization and control of the UAVs themselves.

To address this problem, the multi-UAV mission planning architecture shown in Figure 7-4 was developed in order to carry out complex missions using a team of heterogeneous air- and ground-based UxVs. In the architecture, a mission planner generates a prioritized list of tasks necessary to accomplish the mission. The generated tasks can be either persistent in nature (i.e. long-term surveillance) or transient (i.e. inspect a target vehicle that has been discovered for a short period of time to determine its characteristics). This task list is sent to the task assignment algorithm; possible candidates for this algorithm are for example CBBA [45], RHTA [3, 5], or other task assignment algorithms. Additionally, the mission planner decides how many vehicles are needed to carry out the tasks effectively, and this information is communicated to the persistency planner in order to update the state variable r (number of requested vehicles). The persistency planner executes a policy computed for the persistent surveillance MDP formulation presented in Chapter 5; this policy may be computed exactly if the problem instance is small enough, or BRE or other approximate dynamic programming techniques may be used to compute an approximate policy for large-scale problems. Regardless of the mechanism used to compute

the policy, the persistency planner sends takeoff, landing, and movement commands to the UxVs in order to position them to satisfy the requirements on the number and type of vehicles that are needed to carry out the mission. At the same time, the persistency planner provides a list of active vehicles (i.e. those vehicles that are on station and ready to perform tasks) to the task assignment algorithm. In turn, the task assignment algorithm then uses the list of tasks and the list of active vehicles to assign tasks to the active vehicles. As vehicles become low on fuel or experience sensor failures or degradations, the persistency planner recalls vehicles from the surveillance area as necessary and updates the list of active vehicles accordingly. Given the lists of tasks and active vehicles able to perform those tasks, the task assignment algorithm assigns tasks to vehicles, and communicates the assignments to the vehicles themselves for execution. The adaptive MDP architecture discussed in Chapter 6 enters into Figure 7-4 through the "System Health Information" feedback channel. This channel's purpose is to continually estimate and update the models used in the persistency planner (and potentially in other mission components as well), using the adaptation scheme described in Chapter 6.

7.1.3 Parallel BRE Implementation

The BRE algorithms presented in the thesis were implemented on a high-performance parallel computer cluster. This section describes the parallel implementation and discusses several of the design considerations in applying the BRE algorithms to solve the large-scale persistent surveillance problems that are the subject of the simulation and flight experiments.

All of the various BRE algorithms presented in the thesis are naturally suited to parallel implementation, meaning that many of the steps involve multiple calculations that can be carried out independently, allowing multiple processors to simultaneously work on different pieces of the overall computation. In the following discussion, we will use the BRE(SV) algorithm, presented in Algorithm 3 on page 62, as an example to illustrate the parallelization procedure; all other BRE variants are parallelized in a similar way.

Examining Algorithm 3, notice that the main body of the algorithm consists of a single loop (Lines 8-12). Each of these lines can be parallelized, in the following ways:

Line 8 requires construction of the Gram matrix $\mathbb{K}$ of the associated Bellman kernel $\mathcal{K}(i, i')$, over the set of sample states $\widetilde{\mathcal{S}}$. Since the associated Bellman kernel is symmetric ($\mathcal{K}(i, i') = \mathcal{K}(i', i)$), the Gram matrix is also symmetric and contains $n_s(n_s + 1)/2$ unique elements (where $n_s = |\widetilde{\mathcal{S}}|$ is the number of sample states). Each of these elements is independent of the others, so they can be computed in parallel.

Line 9 requires construction of the single-stage cost values g_i^μ for all states $i \in \widetilde{\mathcal{S}}$. Again, since each of these values are independent of the others, they can all be computed in parallel.

Line 10 involves solving the linear system $\mathbb{K}\underline{\lambda} = \underline{g}^\mu$, using the Gram matrix and cost values computed in Lines 8 and 9. The problem of solving such linear algebra problems in parallel has been extensively studied due to its large practical importance [11, 62, 137]. Furthermore, the ScaLAPACK project [35, 46] is a freely available software package that provides a reference implementation of many parallel linear algebra routines. In our implementation, ScaLAPACK is employed to solve the linear system from Line 10 in parallel.

Line 11 computes the approximate cost-to-go function $\widetilde{J}_\mu(i)$ using Eq. (3.18) and the solution $\underline{\lambda}$ computed in Line 10. Examining this equation, note that $\widetilde{J}_\mu(i)$ consists of a sum of terms of the form

$$k(i', i) - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu k(j, i),$$

one for each state $i' \in \widetilde{\mathcal{S}}$. Each of these terms can be computed in parallel and combined at the end to form the final sum.

Line 12 computes a policy improvement step to get an improved policy $\mu(i)$, which involves minimizing the expected cost, of the form

$$\sum_{j \in \mathcal{S}} P_{ij}(u) \left(g(i,u) + \alpha \widetilde{J}_{\mu}(j) \right),$$

over all possible actions u. The expected costs for different actions are independent, and can thus also be computed in parallel.

As shown, all of the steps of the BRE algorithm can be performed in parallel, allowing the possibility to reduce the total time needed to run the algorithm by a factor of up to m, where m is the total number of processors available. To achieve this goal, a parallel, distributed software architecture has been developed for the purpose of running BRE in parallel. The software utilizes the OpenMPI framework [1], a high-performance message passing library, for communication and coordination between the parallel processes. This framework is highly scalable and is currently utilized in a number of supercomputing applications [68]. In addition to using the core MPI functionality provided by OpenMPI, an extension called STL-MPI [2] is used to allow objects from the C++ Standard Template Library to be easily transmitted and received between processes. The Intel Math Kernel Library [81] provides a high-performance implementation of the ScaLAPACK library and is used for solving the linear system in Line 10.

The current hardware setup used for the parallel BRE implementation consists of 12 workstations equipped with Intel Core2 Duo CPUs, for a total of 24 processors. The workstations run Gentoo linux [70] and are networked together using gigabit ethernet.

7.1.4 BRE Applied to the Persistent Surveillance Problem

In order to apply the BRE algorithm to the various persistent surveillance problems described in this chapter, the base kernel function $k(\cdot, \cdot)$, as well as a strategy for collecting the sample states $\widetilde{\mathcal{S}}$, must be specified. In designing the kernel function,

it is useful to take advantage of any problem-specific structure that allows the kernel to generalize over a large number of states. One important aspect of the persistent surveillance problem that can be exploited is the fact that all agents of the same UxV class are interchangeable. This interchangeability has an important consequence for the cost-to-go function: namely, the cost-to-go function is constant under exchange of agent states. More precisely, assuming only a single UxV class for the moment, the state $\mathbf{x}$ of the persistent surveillance problem can be expressed as

$$\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{q})^T,$$

where $\mathbf{x}_i$ is the state of the i^{th} UAV, N is the total number of UAVs, and $\mathbf{q}$ captures all state information not related to the UAVs themselves (i.e. task request numbers). Then, since the agents are interchangeable, it must be the case that the cost-to-go function $J(\cdot)$ satisfies

$$J((\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_j, \ldots \mathbf{x}_N, \mathbf{q})^T) = J((\mathbf{x}_1, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_i, \ldots \mathbf{x}_N, \mathbf{q})^T). \tag{7.1}$$

Intuitively, Eq. 7.1 captures the idea that it does not matter *which* particular agent happens to be in a given state; since all the agents have the same capabilities, permuting the list of all agent states has no impact on the cost-to-go function. This idea generalizes easily to the case when there are multiple UAV classes; in this case, agent states within the same class can be permuted without affecting the cost-to-go.

To exploit the interchangeable structure of the state space in the kernel function, it is sufficient to sort the individual states of each UAV (using an arbitrary sorting metric) class before processing them by the kernel function. This ensures that all state permutations that are identical in terms of the cost-to-go function are mapped to a unique permutation (namely, the permutation in which the states are sorted) before the kernel processes them. In the experiments presented in this chapter, we employ this sorting strategy along with a standard radial basis function kernel of the form $k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||^2/\gamma^2)$.

In order to generate sample states, sampling trajectories of several types of heuris-

tic policies was used. The heuristic policies ensure that relevant parts of the state space are preferentially sampled. These sample states are then used by the BRE algorithm to compute approximate policies. The results in this chapter will show that the BRE policies often demonstrate significant performance improvements over the heuristic policies used to generate the sample states.

7.2 Basic Persistent Surveillance Problem Results

The first series of flight experiments focused on demonstrating the properties of the overall mission planning architecture (Figure 7-4), including the online MDP adaptation component, using the basic persistent surveillance problem formulation. For these first flight experiments, small instances (involving only 2 or 3 UAVs) of the basic persistent surveillance problem were used, so that the state space of the persistent surveillance MDP was small enough to allow for the optimal policy to be computed exactly using value iteration. These small problems allow baseline results, which use the optimal policy, to be established. In later sections, we will move to using the more complex, extended version of the persistent surveillance problem, as well as increase the number of UAVs. As a result, it will no longer be possible to solve the MDP exactly, and we will instead apply BRE algorithms to quickly compute approximate policies for these problems. Furthermore, we will show that the approximate BRE policies give results that are near-optimal.

7.2.1 Baseline Results

To begin, we used the basic persistent surveillance problem formulation in conjunction with the mission planning architecture presented in Figure 7-4 to establish baseline performance and confirm that the results achieved in real flight experiments match the behavior observed in simulation (i.e. Figure 5-1).

In these experiments, the controller had access to $n = 3$ quadrotor UAVs and was commanded to maintain $r = 2$ UAVs over a surveillance area. The parameters of the experiments were adjusted to make the problem intentionally hard for the controller.

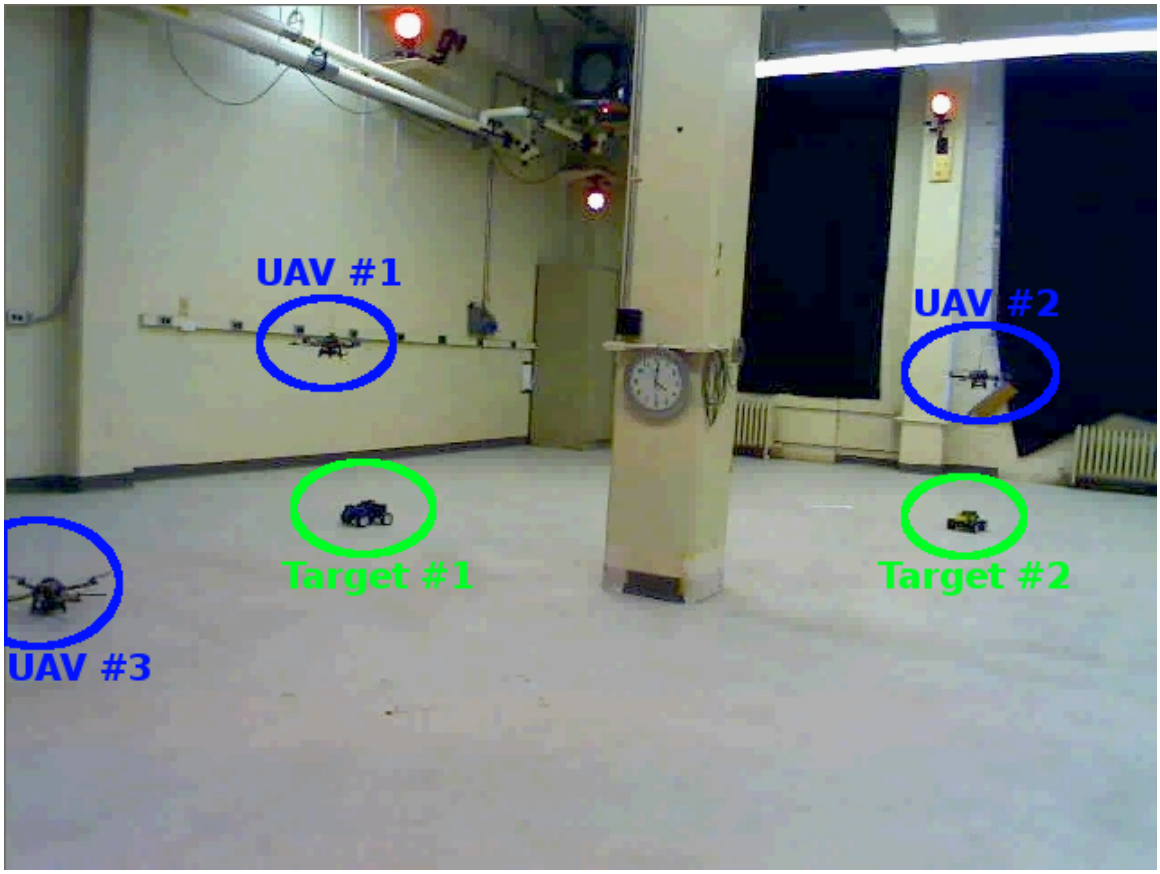


Figure 7-5: Persistent surveillance flight experiment ($n = 3$, $r = 2$), showing UAVs #1 and #2 tracking the two target vehicles, while UAV #3 waits at base.

That is, the distance from base to the surveillance area was large relative to the amount of fuel each UAV could carry, thus necessitating rapid swapping of the UAVs in order to maintain the desired coverage.

Figure 7-5 shows the basic setup of the flight experiment in RAVEN. The base area is located on the left side of the figure, and the the two requested UAVs in the surveillance area are tasked with tracking two ground targets that move about randomly in this area.

Flight results for a typical experiment are shown in Figure 7-6. Despite the difficulty of the problem, the performance of the controller is excellent, providing nearly continuous (97%) coverage over the 45-minute duration of the mission. Furthermore, the qualitative behavior of the flight controller is identical to that seen in simulation. This establishes that the MDP-based controller can be successfully implemented on

the RAVEN flight testbed.

7.2.2 Importance of Group Fuel Management

As discussed in Chapter 5, one of the key requirements of a health management system is the ability to consider the impact on the overall mission of a change in the health state of individual vehicles. This requirement is automatically satisfied by our MDP based approach, since the cost function of the MDP captures the interdependencies between vehicles on the overall mission. To highlight the importance of this requirement, we compare the flight results shown in Figure 7-6 with an alternative planning approach for the persistent surveillance problem [98]. This approach is based on a non-proactive, heuristic method that requires every UAV to keep the base location within its "operational radius", which is an estimated distance the UAV can fly before running out of fuel. Thus, while the heuristic ensures individual vehicle safety by always requiring that the vehicle is able to return to base when necessary for refueling, it does not consider the broader impact that refueling a vehicle has on the overall mission coverage.

The operational radius heuristic was used to command a group of three UAVs ($n = 3$) to track two ground vehicles ($r = 2$), in an identical experimental setup to that of the previous section. The heuristic commanded the UAVs to enter the surveillance area based on the request number r. When an individual vehicle's estimated flight time remaining dropped below 1 minute, the method recalled the UAV to base for refueling and commanded another one to take its place.

Flight results from the heuristic-driven experiment are shown in Figure 7-7. Notice the gap in coverage that occurs at time $t = 38$. This problem was caused because both UAVs in the surveillance area reached a low fuel state at nearly the same time. Since the heuristic method was not proactive in looking into the future to predict this conflict, it could not avoid the undesirable situation of needing to recall both vehicles at the same time, resulting in a loss of coverage. Also, since the heuristic method did not account for the transit time between the base and the surveillance location, gaps were observed in the coverage even during normal swap-out events. As a result, the

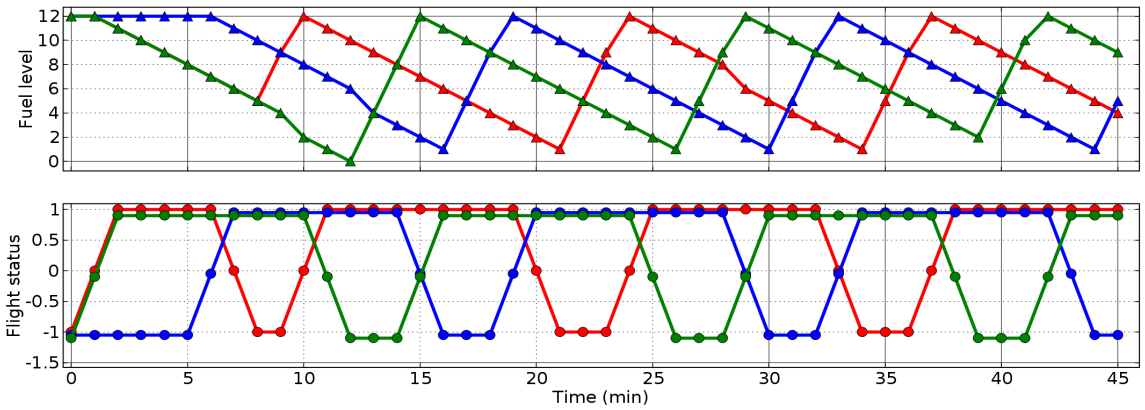


Figure 7-6: Persistent surveillance flight results for $n = 3$, $r = 2$.

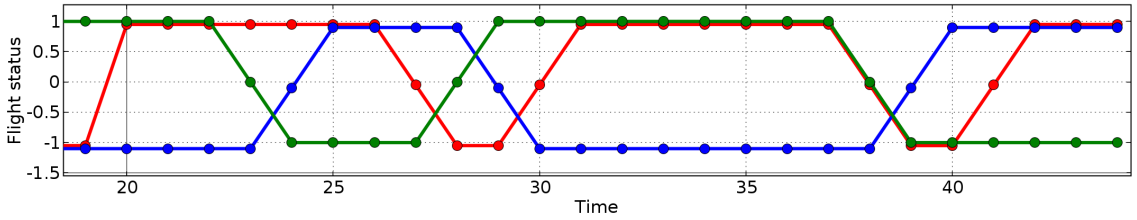


Figure 7-7: Persistent surveillance flight experiment, using a heuristic control policy that manages fuel for each UAV individually. Note the coverage gap beginning at $t = 37$.

heuristic's achieved coverage over the course of the 45-minute mission is only 85%, compared with 97% for the MDP-based control policy.

7.2.3 Adaptive Flight Experiments

In order to demonstrate the effectiveness of the adaptive MDP architecture in addressing the problem of uncertain and time-varying models in the basic persistent surveillance problem, a number of adaptive flight experiments were conducted. The flight experiments encompassed a number of scenarios to illustrate the benefits of the adaptive MDP architecture. The flight experiments all involved a two-UAV, basic persistent surveillance mission where the goal was to maintain one vehicle on-station at all times. The scenarios were designed to be challenging to solve, in the sense that the vehicles had very limited fuel capacities. Due to this limitation, any inefficiencies introduced by vehicle failures (such as increased fuel usage) or sub-optimal policy

decisions resulted in degraded surveillance performance, making it possible to clearly observe the benefits of the adaptive approach.

First Scenario

To begin, a simple scenario illustrating the importance of proper modeling was run. In this scenario, a static (non-adaptive) control policy, based on a nominal fuel flow parameter of $p_{nom} = 1.0$, was implemented. In reality, the true parameter value of the system was $p_{nom} = 0.8$. Flight results are shown in Figure 7-8. Since the control policy was overly optimistic about the fuel burn rate, vehicles do not return to base with any extra fuel reserves, making them vulnerable to off-nominal fuel burn events. As a result, both vehicles end up crashing relatively soon after the start of the mission.

The parameter mismatch in these flights corresponds to the risky region of Figure 6-2, where, by virtue of being overly optimistic with regards to the true parameter, there is a dramatic loss of coverage due to vehicle crashes.

Second Scenario

The first scenario illustrated the danger of implementing an overly optimistic policy. A second non-adaptive scenario shows that the opposite, conservative approach is safer, but also leads to sub-optimal mission performance. In this scenario, a static policy based on $p_{nom} = 0.0$ was implemented for the same system as the first scenario, where the true value was $p_{nom} = 0.8$. In this case, the policy is overly conservative. Since it assumes the vehicles will burn fuel at a high rate all the time, the vehicles return to base with very large fuel reserves. This is a safe strategy in that the vehicles never crash, but the overall mission performance is quite low, because the vehicles only stay on station for a short period of time before returning to base (see Fig. 7-9). Note that minimax-like strategies for computing the policy offline result in these types of conservative policies, which are safe but may not perform well.

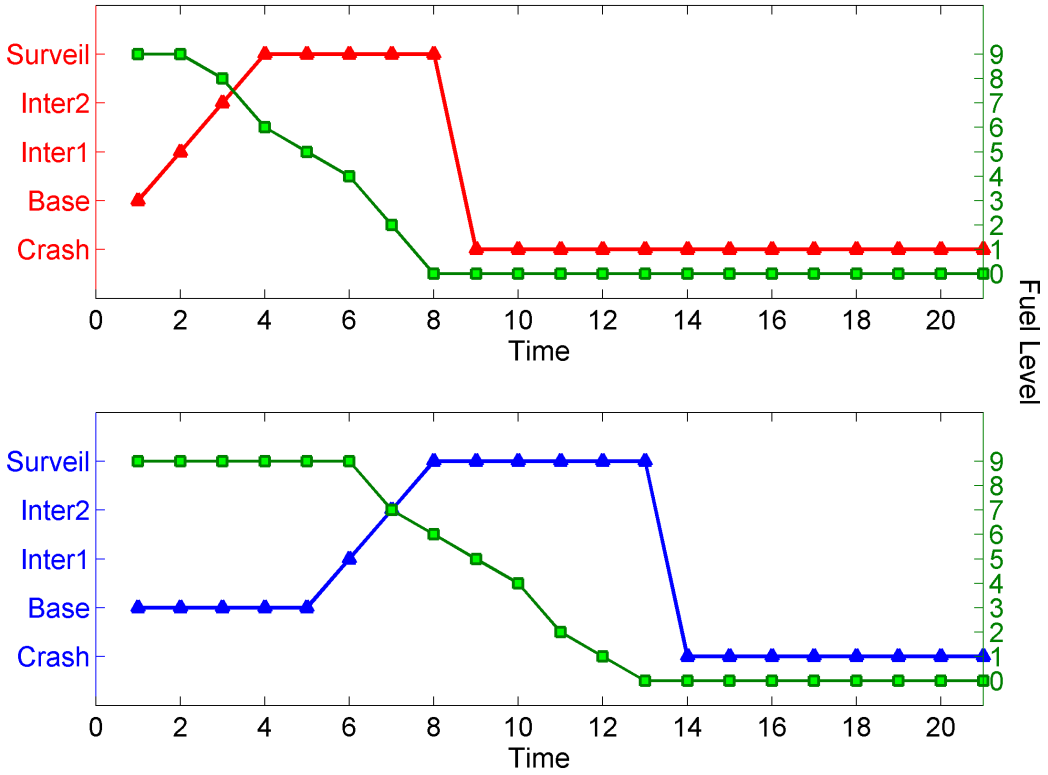


Figure 7-8: Static policy based on $p_{nom} = 1.0$. True value: $p_{nom} = 0.8$. Both vehicles crash shortly after the start of the mission.

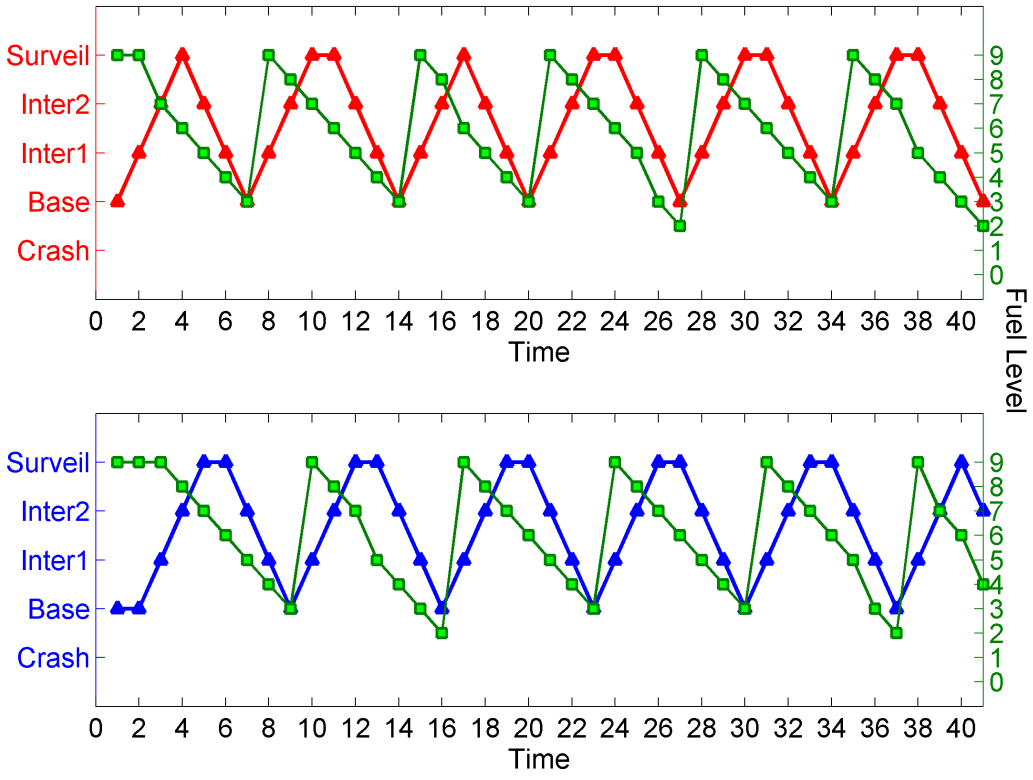


Figure 7-9: Static policy based on $p_{nom} = 0.0$. True value: $p_{nom} = 0.8$. This overly conservative policy avoids crashes at the expense of very low mission performance.

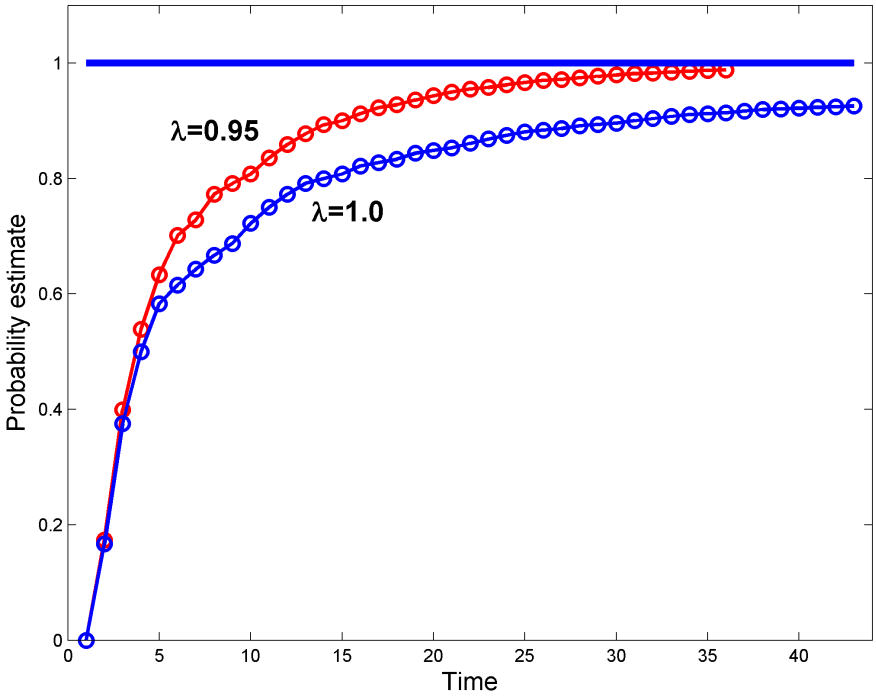
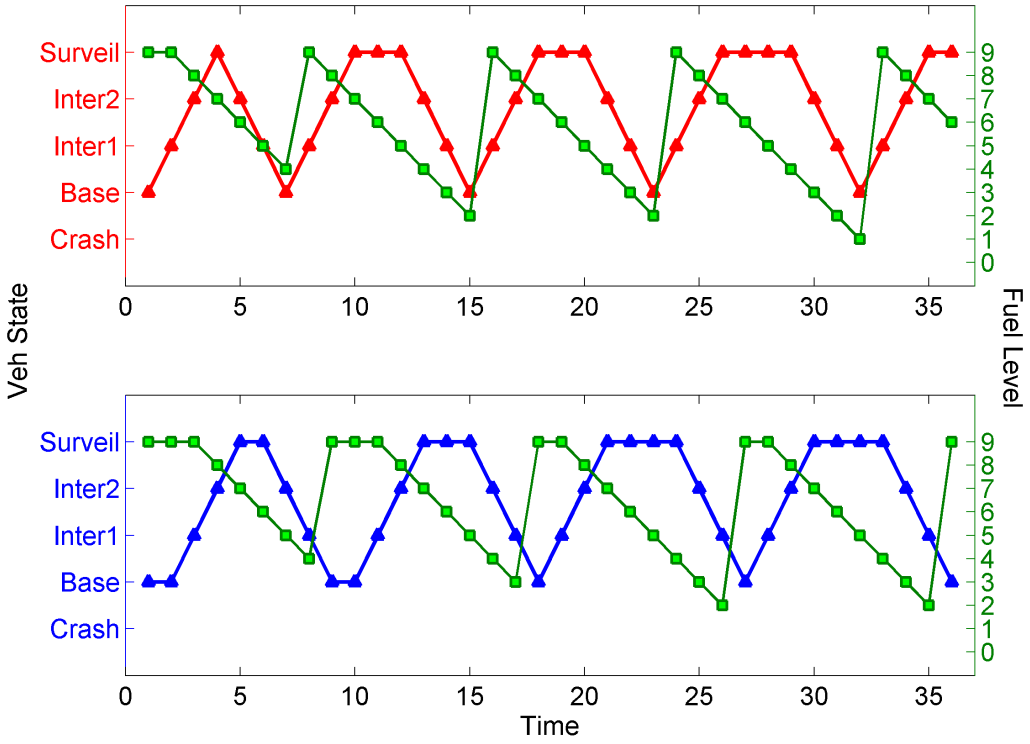


Figure 7-10: Undiscounted estimator (blue) is slower at estimating the probability than the discounted estimator (red)

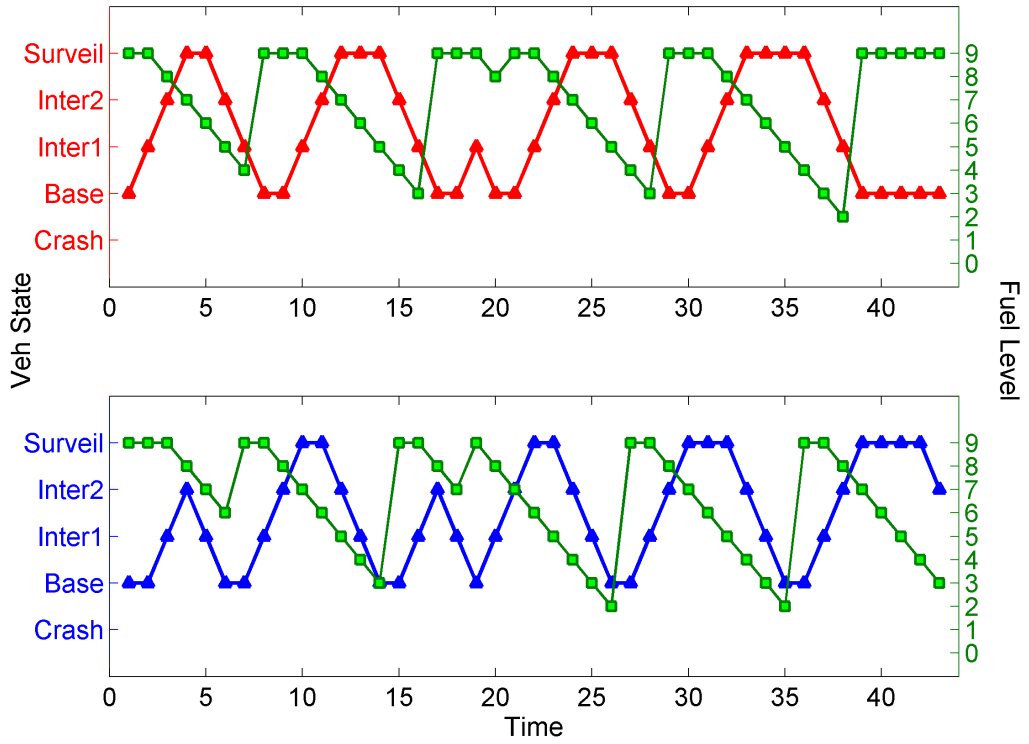
Third Scenario

Having established the difficulties in implementing both overly conservative and optimistic policies, a new set of scenarios were run in which the adaptation mechanism described in the previous section was used. In this scenario the estimator and policy were initialized at $p_{nom} = 0.0$, while the true value was $p_{nom} = 1.0$. The flight was started with the adaptation mechanism running. A plot of the estimated parameter value as a function of time is shown in Fig. 7-10, for two different values of the fading factor λ. The figure shows that the estimate converges to the true value of 1, as expected. Furthermore, the vehicle flight data is shown in Fig. 7-11. For these missions, the surveillance efficiency of the vehicles was defined as the ratio of the time the UAV spent on surveillance to the total mission time, $T_{\text{Surv}}/T_{\text{mission}}$.

Fig. 7-11 illustrates that the system policy continually improves the performance as better estimates of the fuel flow parameter are obtained. In particular, when the system starts operating, the policy is very conservative since its initial estimate is $p_{nom} = 0$. Therefore, the vehicles do not stay on station very long. However, as



(a) Adaptation with $\lambda = 0.95$



(b) Adaptation with $\lambda = 1$

Figure 7-11: Surveillance efficiency for the two estimators. The discounted (top), with surveillance efficiency of 72%, and undiscounted estimator (bottom) with surveillance efficiency of only 54%.

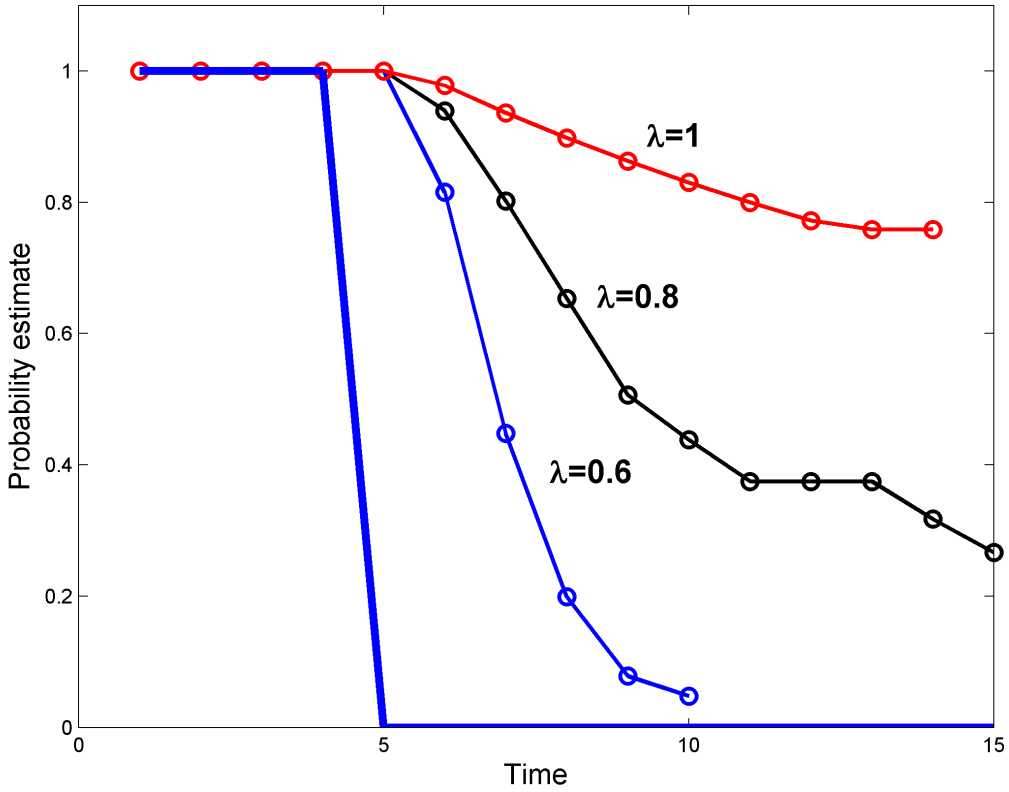
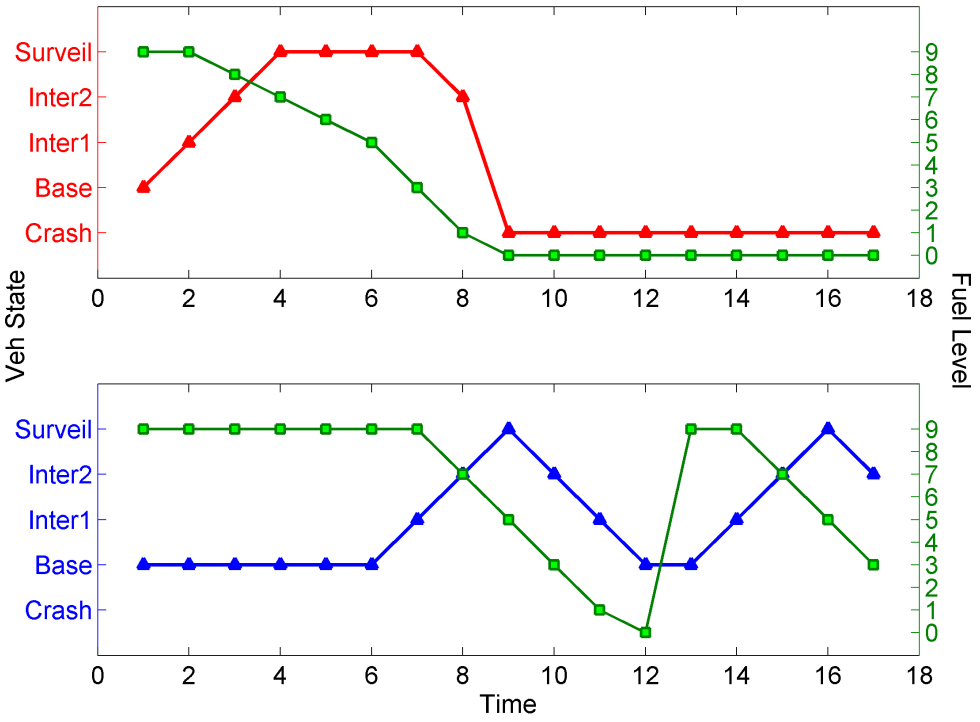


Figure 7-12: Step response from $p_{nom} = 1$ to $p_{nom} = 0$ for three values of λ, showing that $\lambda = 0.6$ has a response time of approximately 5 times steps, while $\lambda = 1$ has a very slow response time.

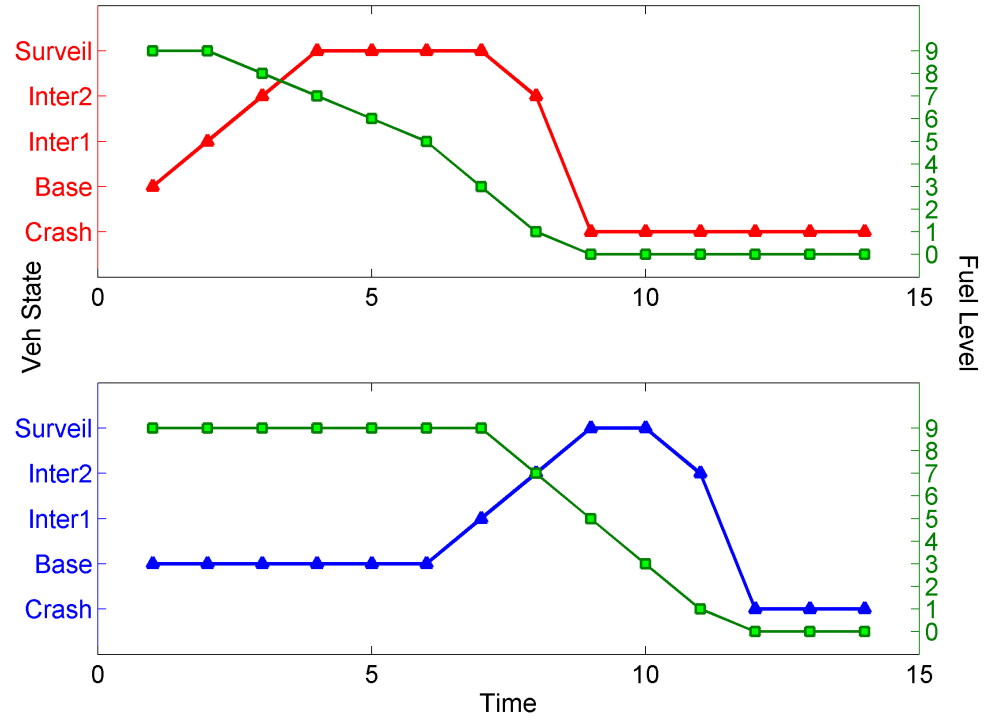
the estimate increases towards the actual value, the policy is updated and the vehicles stay on station longer, thereby increasing mission performance. This experiment demonstrates successful adaptation from an initially poor parameter estimate. Furthermore, it demonstrates the value of the discounted estimation approach, since the discounted estimator ($\lambda = 0.95$) converged to the true value more quickly than the undiscounted estimator ($\lambda = 1.0$). As a result, total mission efficiency for the discounted estimator was higher than the undiscounted estimator.

Fourth Scenario

The next scenario demonstrated the ability of the adaptation mechanism to adjust to actual model changes during the mission, such as might be observed if the vehicles were damaged in flight. In this scenario, the vehicles were initialized with a $p_{nom} = 1$



(a) Adaptation with $\lambda = 0.80$



(b) Adaptation with $\lambda = 1$

Figure 7-13: For $\lambda = 0.8$ (top), the red vehicle crashes as it runs out of fuel following a change in the fuel burn dynamics, but the faster estimator response allows an updated policy to be computed in time to prevent the blue vehicle from crashing. For $\lambda = 1$ (bottom), the estimator is too slow to respond to the change in the dynamics. As a result, both vehicles crash.

and the model was changed to $p_{nom} = 0$ after approximately 2 minutes (5 time steps), mimicking adversarial actions (such as anti-aircraft fire) and/or system degradation over time. The change in the probability estimate is shown in Figure 7-12 for three different choices of $\lambda = \{0.6, 0.8, 1\}$. It can be seen that the classical estimation ($\lambda = 1$) results in a very slow change in the estimate, while $\lambda = 0.8$ is within 20% of the true estimate after 10 time steps, while $\lambda = 0.6$ is within 20% after only 3 time steps, resulting in a significantly faster response. The variation of λ resulted in an interesting set of vehicle behaviors that can be seen in Figure 7-13. For $\lambda = 1$, the estimate converges too slowly, resulting in slow convergence to the optimal policy. The convergence is so slow that both vehicles crash (vehicle 1 at time step 9, and vehicle 2 at time step 12), because the estimator was not capable of detecting the change in the value of p_{nom} quickly, and these vehicle were still operating under an optimistic value of $p_{nom} \approx 0.8$. Due to the physical dynamics of the fuel flow switch for this scenario, it turns out that the first vehicle will inevitably crash, since the switch occurs when the vehicle does not have sufficient fuel to return to base. However, if the estimator is responsive enough to detect the switch quickly, the updated policy can prevent the second vehicle from crashing. This does not occur when $\lambda = 1$. The benefits of the more responsive estimator are seen in the bottom figure, where by selecting $\lambda = 0.8$, the second vehicle only spends one unit of time on surveillance, and then immediately returns to base to refuel, with only 1 unit of fuel remaining. Thus, the faster estimator is able to adapt in time to prevent the second vehicle from crashing.

Fifth Scenario

The final scenario was a slightly different test of the adaptation mechanism in tracking a series of smaller step changes to p_{nom}. In the earlier flight tests, under a nominal fuel flow, $p_{nom} = 1$, the fuel transitions were always of 1 unit of fuel. Likewise, when $p_{nom} = 0$, the fuel transitions were always of 2 units of fuel. In this test, the transition probability p_{nom} was decreased in steps of 0.3, and the estimators saw both nominal and off-nominal fuel transitions in the estimator updates at each time

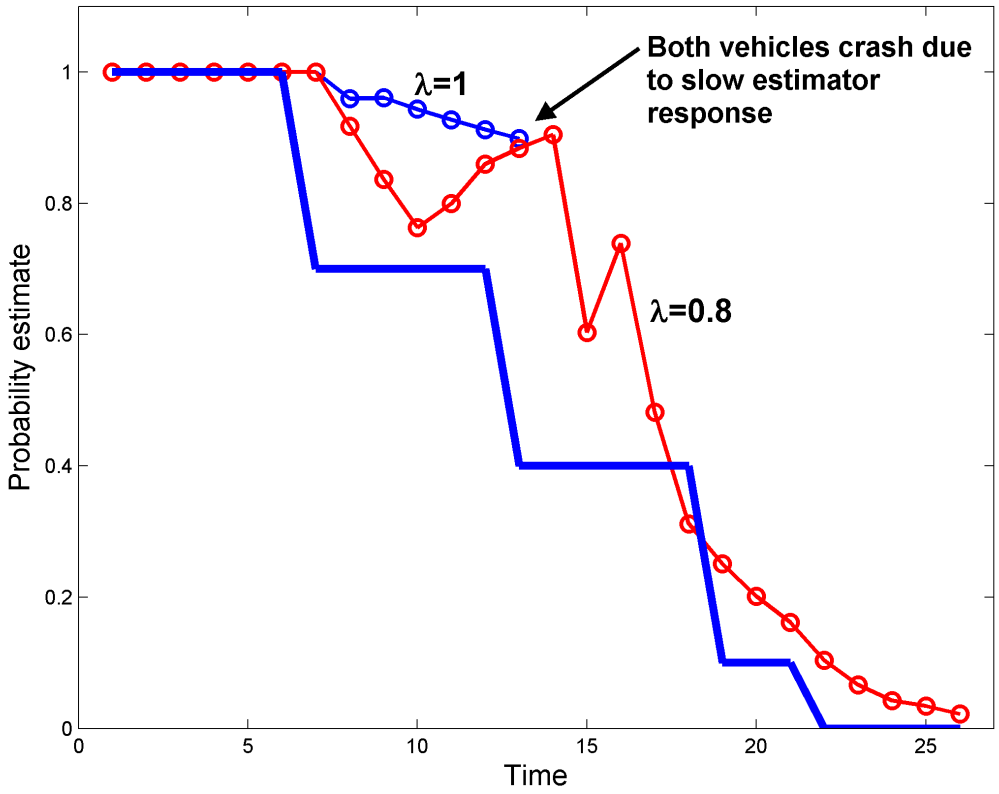
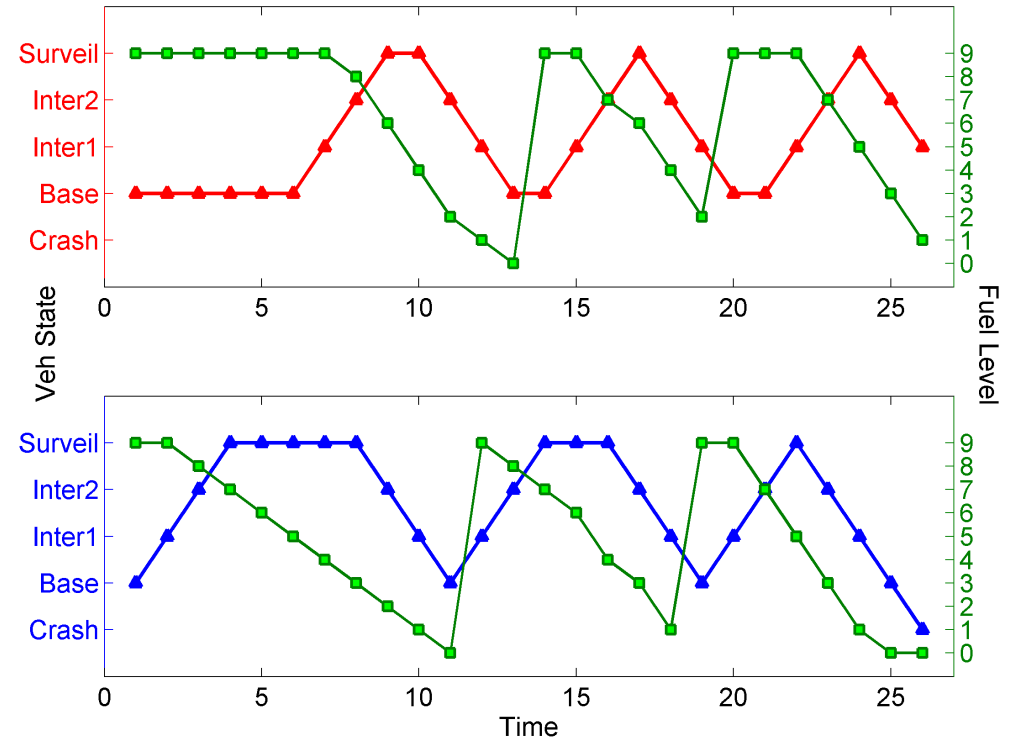


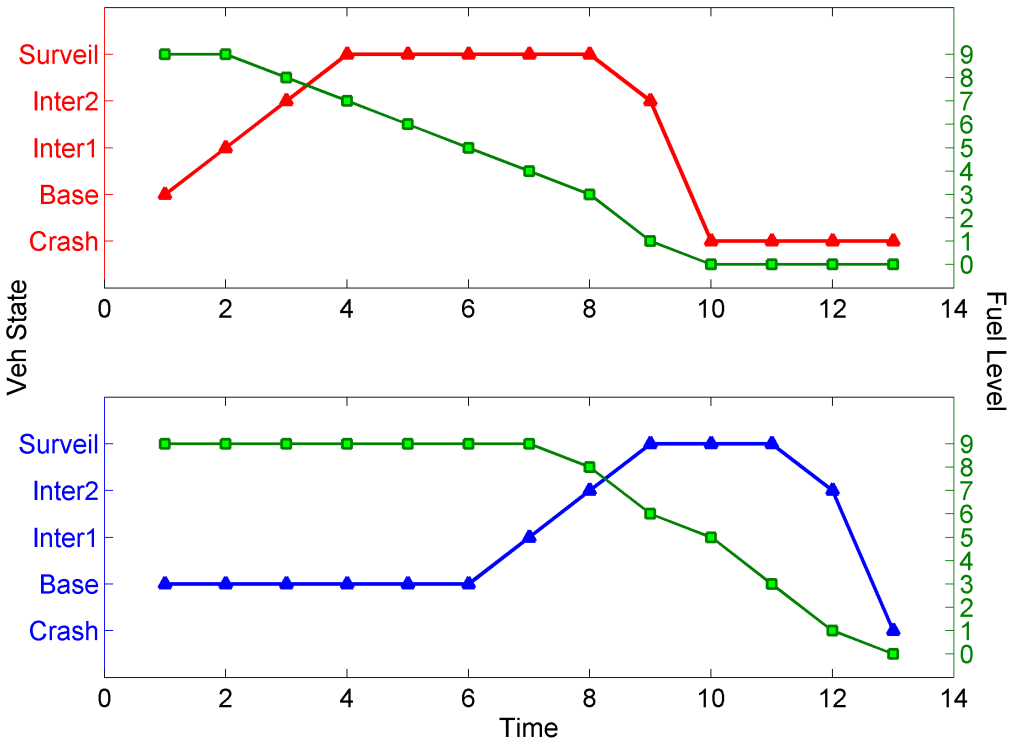
Figure 7-14: Probability estimates of p_{nom} for $\lambda = 0.8$ and $\lambda = 1$. Due to the slow response of the latter estimator, both vehicles crash by time step 13, and no further adaptation is possible. Estimator with $\lambda = 0.8$ shows faster response, and ultimately converges to the true value.

step (unlike the earlier tests where they *either* saw nominal transitions or off-nominal transitions). As a result, this test was perhaps a more realistic implementation of a gradual temporal degradation of vehicle health. Figure 7-14 is shown for two different choices of $\lambda = \{0.8, 1\}$. The first item of note is the step decreases in p_{nom}, that unlike the earlier flight results, are more subtle. Next, note that the initial response of the undiscounted estimator (blue) is extremely slow. In this flight test, the adaptation was so slow that the significant mismatch between the true and estimated system resulted in a mismatched policy that ultimately resulted in the loss of both vehicles.

Also of interest is the responsiveness of the discounted estimator (red) which shows that the variation of λ resulted in a rich set vehicle behaviors that can be seen in Figure 7-15. For $\lambda = 1$, the estimate converges too slowly, resulting in slow



(a) Adaptation with $\lambda = 0.80$



(b) Adaptation with $\lambda = 1$

Figure 7-15: The slower estimator $\lambda = 1$ (bottom) does not detect the fuel flow transition sufficiently quickly, causing both vehicles to crash. The faster estimator $\lambda = 0.8$ (top) quickly detects changes in the transition and results in good mission performance with no crashes.

convergence to the optimal policy. The convergence is so slow that both vehicles crash (vehicle 1 at time step 9, and vehicle 2 at time step 12), because the estimator was not capable of detecting the change in the value of p_{nom} quickly, and the vehicles were still operating under an optimistic value of $p_{nom} \approx 0.8$. In contrast, for the faster estimator $\lambda = 0.8$, the changes are detected quickly, and the policy smoothly decreases the vehicles' on-station surveillance times in accordance with the changing fuel dynamics. This results in the greatest possible surveillance coverage given the gradual degradation of the vehicles, while still avoiding crashes.

7.3 Extended Persistent Surveillance Problem Results

We now turn to larger, more complex missions using the extended persistent surveillance problem formulation. Due to their increased complexity, many of these problem instances were impossible to solve exactly, so BRE algorithms were used to compute approximate policies instead. However, performance comparisons between the approximate BRE policies and the optimal policy are made for several problem instances that are small enough to solve exactly.

7.3.1 Simulation Results

The parallel BRE implementation was used to compute an approximate policy for a number of extended persistent surveillance missions with communication constraints and sensor failures to illustrate the properties of the solution. Note that in these missions, only a single type of UAV was used (so that the "heterogeneous UAV team" extension described in Section 5.4.3 was not needed). The basic layout of the mission is shown in Figure 7-16. In the mission, the surveillance area is located behind a group of mountains which prevents direct communication with base. In order to establish a communications link, it is necessary to have a UAV loiter in the area marked "communications relay point" in the figure. Therefore, for this mission,

Figure 7-16: Layout of the extended persistent surveillance mission.

the communications function $comm(\mathbf{x})$ is given by

$$comm(\mathbf{x}) = \begin{cases} 1 & \text{if a UAV is present at the communications relay point in state } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}. \tag{7.2}$$

The chosen mission used a total of $n = 3$ UAVs. The surveillance location was taken as $Y_s = 3$, and the fuel capacity of each UAV was $F_{max} = 15$ (with $\Delta f = 1$). Therefore, using Eq. 5.5, the size of the state space was $28,311,552$, and the average branching factor β was 8. Finally, the parameters $p_{sensorfail}$ and p_{nom} were set at 0.02 and 0.95, respectively.

An approximate policy for this mission was computed using the parallel BRE implementation. Using a set of 300 sample states (generated by sampling trajectories from a heuristic policy, described below), the parallel BRE algorithm was run on the 24-processor computer cluster to compute an approximate policy. Starting from the initial configuration with all UAVs located at base, the approximate policy was simulated for 1,000,000 steps, and the average cost incurred per step was computed

to be 0.802.

In order to compute the exact policy for comparison purposes, a parallel implementation of value iteration was run using the same 24-processor cluster and software setup as described in the previous section. The optimal policy incurs an average cost of 0.787. Thus, for this problem, the approximate policy computed by BRE is within 1.9% of optimal, while requiring significantly less time to compute compared to the optimal policy.

Results of simulating the approximate policy for the persistent surveillance mission, for a "lucky" case where no sensor failures occur, are shown in Figure 7-17. These results are presented first to show how the policy behaves in the nominal case. Examining the figure, notice that the policy establishes a regular switching pattern, where each vehicle alternates between providing a communications link, performing surveillance, and returning to base for refueling and maintenance. Even in the nominal case without the added complexity of dealing with sensor failures, the policy exhibits a number of complex and subtle properties, including having UAVs return to base with a reserve of fuel (similar to the results presented in [30]) and precisely arranging the arrivals and departures of each UAV at the communications and surveillance locations such that the important communications link is always established, allowing unbroken communications with base even as UAVs switch roles and locations.

A set of further results is shown in Figure 7-18. This scenario is similar to the results of Figure 7-17, but in this case, a number of sensor failures did occur over the course of the mission. Depending on when in the mission a particular failure occurs, the policy responds differently to the failure. If a failure occurs when the UAV is in the surveillance area, it is sometimes immediately recalled to base to refuel and be repaired (this is case for the top UAV in the figure at time $t = 79$). In other cases, the vehicle instead moves to the communications area and begins providing communications relay service while another UAV with a functional sensor takes its place in the surveillance area (for example, this happens to the middle UAV at $t = 31$). If the sensor failure occurs while the UAV is en route to the communications area, on the other hand, the policy reassigns the UAV currently providing communications

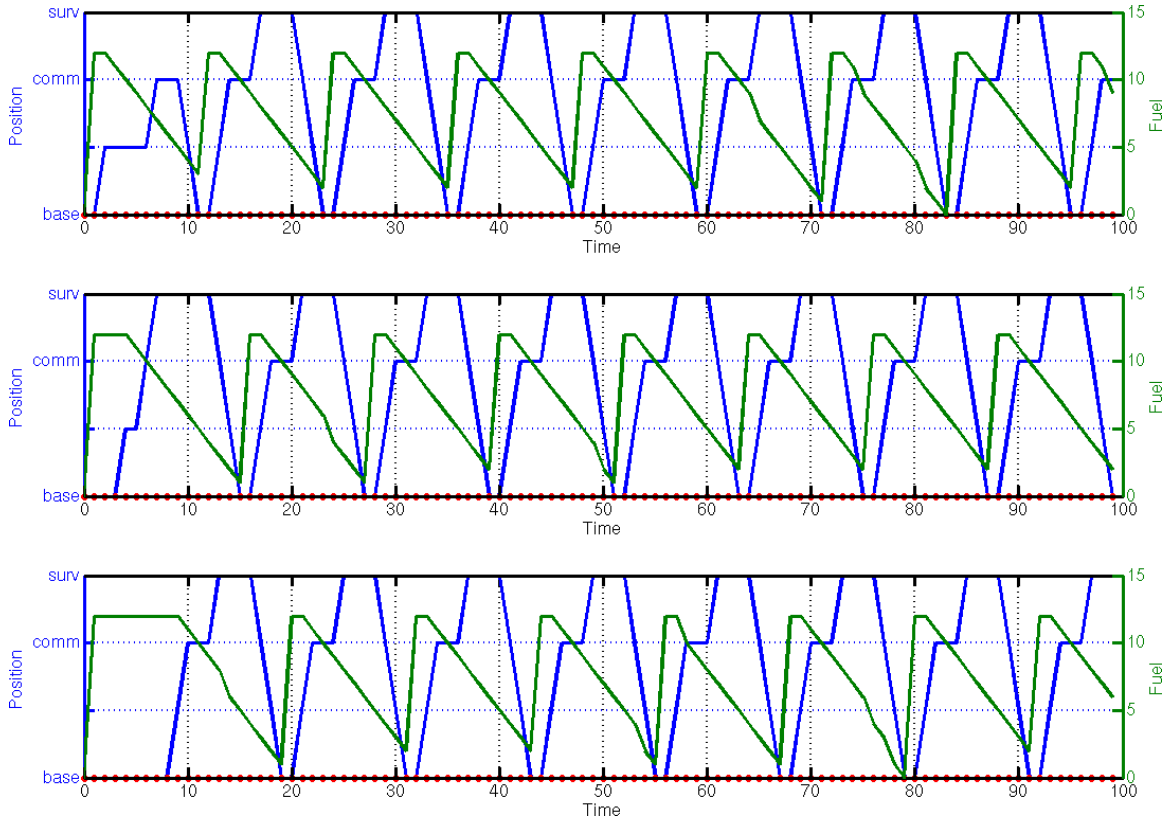


Figure 7-17: Extended persistent mission results for a nominal 3-vehicle mission with no sensor failures. Each of the 3 UAVs' states is separately depicted, where the green lines indicate fuel remaining, the blue lines indicate position, and the red circles indicate sensor state (0: sensor operational, 1: sensor failed). The approximate policy computed using BRE establishes a regular switching pattern, where each vehicle alternates between providing a communications link (labeled "comm"), performing surveillance (labeled "surv"), and returning to base for refueling and maintenance.

coverage to the surveillance area (this occurs at $t = 24$, for example). In all cases, the policy produces an appropriate response that results in maintaining surveillance and communications coverage, even in the event of the failures.

An interesting qualitative change in the behavior of the policy happens as the probability of sensor failures increases. For higher failure probabilities, the extra fuel cost of sending an additional, "backup", UAV to the surveillance area becomes outweighed by the cost of losing surveillance coverage if only a single UAV is sent and subsequently fails. For the selected mission parameters described above, the optimal policy switches to sending a backup UAV as $p_{sensorfail}$ increases from 0.02 to 0.03. Similarly, the approximate BRE policy switches to the backup strategy at the same

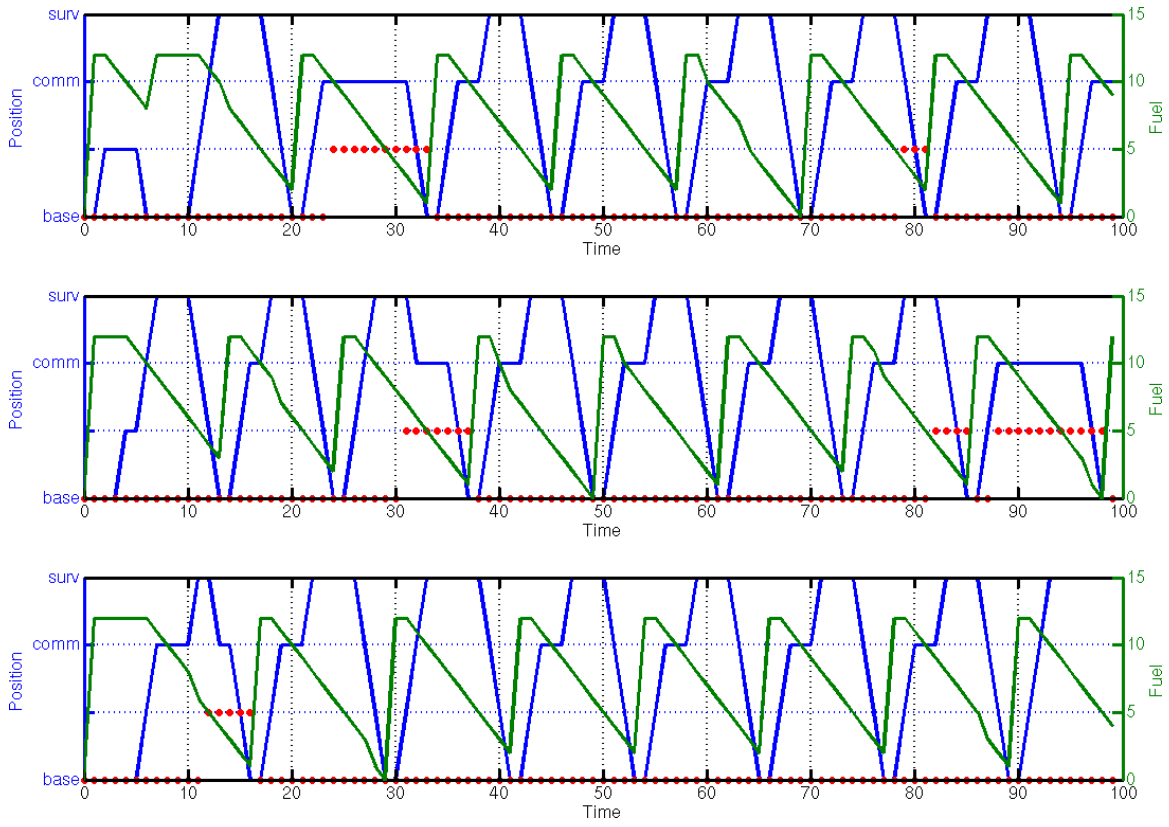


Figure 7-18: Extended persistent mission results for a 3-vehicle mission with sensor failures ($p_{sensorfail} = 0.02$). The sensor failures are indicated by the raised red dots. UAVs with failed sensors are useless for performing surveillance, but can still relay communications.

value of $p_{sensorfail}$ as the optimal policy. Results of the approximate BRE policy running for the case $p_{sensorfail} = 0.03$ are shown in Figure 7-19. In the figure, the effectiveness of the backup strategy is highlighted: several UAVs experience sensor failures while in the surveillance area, but since a backup UAV is always on station, continuous surveillance coverage is maintained.

To further explore the effect of changing the sensor failure probability on the BRE policy, a series of simulations was run. In each simulation, the BRE policy was computed for a specific value of $p_{sensorfail}$, and this policy was then simulated to determine the number of UAVs that the policy attempted to maintain at the surveillance location. Simulations were carried out for values of $p_{sensorfail}$ from 0.0 to 1.0 in steps of 0.01. The results of these simulations are shown in Figure 7-20. The figure illustrates that the policy behavior is constant over a wide range of failure proba-

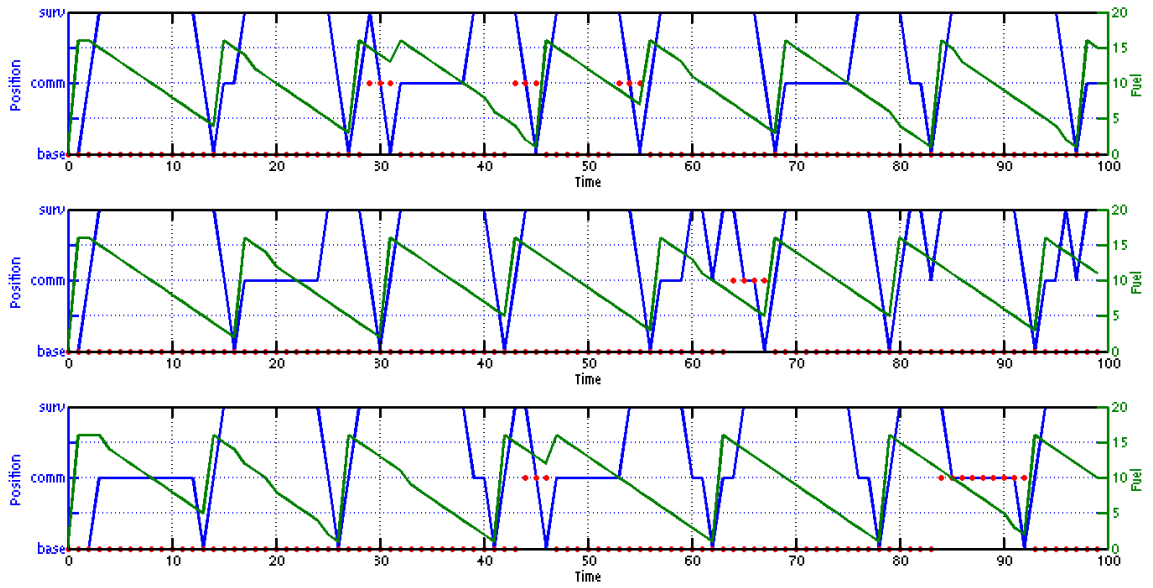


Figure 7-19: Extended persistent mission results for $p_{sensorfail} = 0.03$. For this higher value of $p_{sensorfail}$, the approximate policy computed by BRE recognizes the increased risk and sends two UAVs to the surveillance location, anticipating the possibility that one of them will fail while on station.

bilities: namely, a single surveillance UAV is maintained for all values of $p_{sensorfail}$ less than 0.03, while two surveillance UAVs are maintained for all values of $p_{sensorfail}$ from 0.03 to 0.89. Note that for values of $p_{sensorfail}$ greater than 0.89, the policy does not maintain *any* UAVs at the surveillance location, since the likelihood of achieving successful coverage is very low, and any benefit of this coverage is outweighed by the fuel costs necessary to fly the UAVs. Of course, for realistic systems, we would hope that the failure probability is not excessively large, so the data for these large values of $p_{sensorfail}$ is less interesting from a practical standpoint. Nevertheless, it is useful to observe the behavior of the policy over the entire range of the $p_{sensorfail}$ parameter.

Figure 7-20 has important implications for online adaptation in the persistent surveillance problem. Clearly, in the online adaptation architecture presented in Chapter 6, the accuracy of the parameters learned by the online parameter estimator play an important role in ultimate performance achieved by the adaptive policy. As shown in Chapter 6, using a discounted MAP estimation strategy can help to achieve fast convergence to the true parameter values, but there may still be cases where

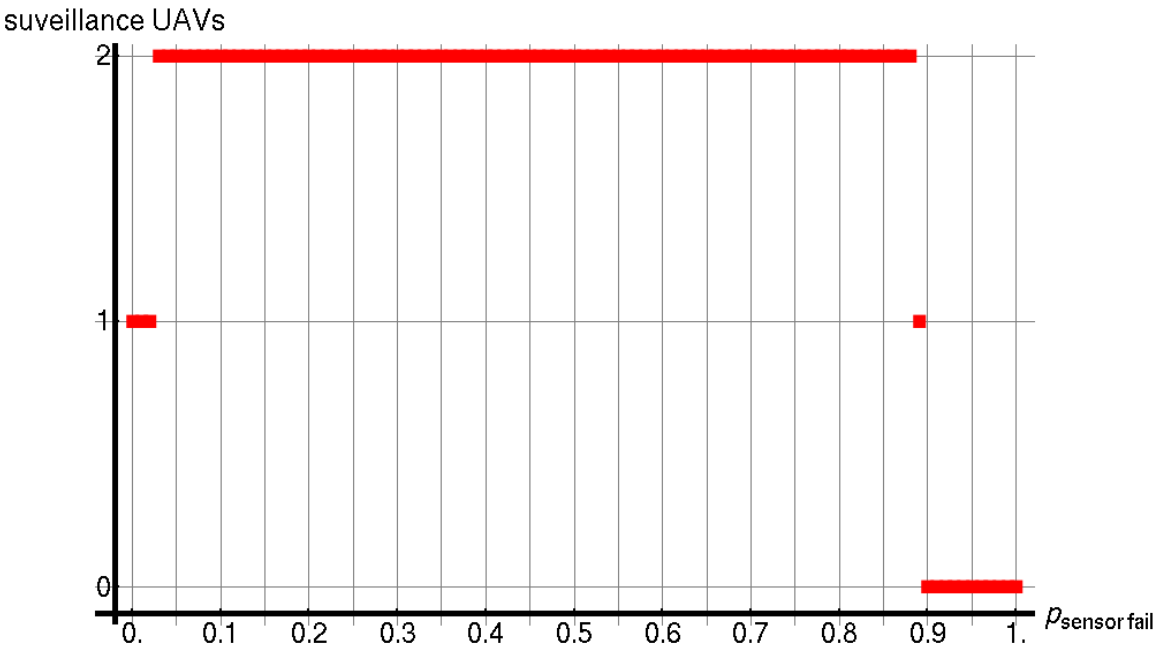


Figure 7-20: Number of UAVs maintained at the surveillance location by the BRE policy, as a function of $p_{sensorfail}$. The mission has $n = 3$ total UAVs available.

excessive noise precludes estimating the parameters exactly, or where insufficient data has been gathered to permit an accurate estimate. The data shown in Figure 7-20 indicates that the policy for the persistent surveillance problem is relatively insensitive to the value of $p_{sensorfail}$. Essentially, this means that as long as the estimator can determine that the value of $p_{sensorfail}$ lies in of the "bins" defined by the segments in Figure 7-20 where the policy remains constant, the adaptive system will execute the correct policy.

7.3.2 Comparison with Heuristic Policy

A heuristic policy was developed for the extended persistent surveillance problem in order to compare against both the optimal policy and the approximate policy computed by BRE. The heuristic is based on the Receding Horizon Task Assignment (RHTA) framework [3]. In particular, the heuristic computes the actions for each UAV by first defining tasks at the communications relay point and the surveillance location. It then uses RHTA to compute the optimal assignment of UAVs to tasks, where the goal is to minimize the total fuel usage of all UAVs (or, equivalently, to

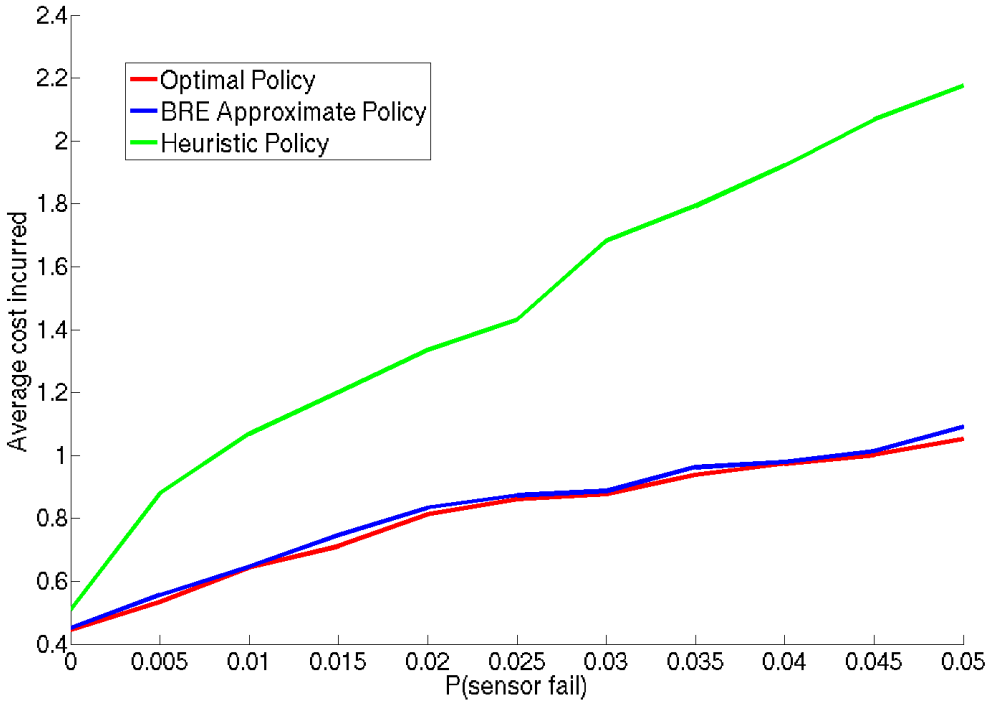


Figure 7-21: Average cost incurred in the extended persistent surveillance mission by the optimal policy, heuristic policy, and the policy computed by BRE, as a function of $p_{sensorfail}$.

minimize the total distance traveled). The optimization problem solved by RHTA includes constraints that account for the need of the UAVs to periodically return to base to refuel and the inability of UAVs with failed sensors to perform surveillance. The RHTA framework does not account for uncertainties in the system dynamics. Thus, the fuel constraint is designed to ensure the UAVs always conserve enough fuel to return to base even in the worst-case scenario where the UAV burns the maximum amount of fuel at each time step. The heuristic policy is computed on-line by constructing and solving the task assignment problem at every time step.

A comparison of the performance of the heuristic policy, BRE policy, and optimal policy is shown in Figure 7-21. The figure plots the average cost incurred by each policy (over a simulation run of 1,000,000 steps) as a function of $p_{sensorfail}$. For values of $p_{sensorfail}$ near zero, both the heuristic policy and the BRE policy yield near-optimal performance: the heuristic policy's cost is within 10% of optimal, and

the BRE policy is within 2%. This indicates that, for nearly deterministic systems where failures are very unlikely to occur, the heuristic policy is a good approach to solving the persistent surveillance problem. However, as $p_{sensorfail}$ increases, the performance of the heuristic policy degrades while the BRE policy remains very close to optimal performance. This is due to the fact that the heuristic policy does not account for the stochastic dynamics of the problem. Therefore, the heuristic can *react* to failures as they occur, but it cannot *anticipate* their occurrence and take steps to mitigate their consequences. In particular, notice the large increase in cost that the heuristic policy incurs around $p_{sensorfail} = 0.025$. Recall that this is the point at which the optimal and BRE policies recognize the need to send out a backup vehicle to the surveillance area, because the likelihood of a failure becomes more important to consider. Since the heuristic policy cannot anticipate failures, it fails to send out a backup vehicle and incurs a large cost increase as a result.

Results shown in Figure 7-22 further illustrate the advantage of the proactive policy that is generated by the BRE solution approach. The figure plots the total accumulated cost incurred by the BRE policy and the heuristic policy over the course of a simulated mission. In particular, the figure plots the function

$$G(t) = \sum_{t'=0}^{t} g(\mathbf{x}(t'), \mathbf{u}(t')) \tag{7.3}$$

as a function of elapsed time t, for each policy. In order to make the comparisons fair, the missions for both policies were simulated in the same way: namely, a number of failures were pre-scheduled to occur at the same time for both missions. $p_{sensorfail}$ was set to 0.03, the length of each mission was 100 time steps, and the sensor failures occurred at $t = 10$, 43, and 76. Therefore, both policies had to deal with the same type and number of failures, so differences in performance between the two policies are due only to the way they handle the failures. Examining Figure 7-22, notice first that the slope of the accumulated cost function $G(t)$ is slightly higher for the BRE policy than for the heuristic policy in the nominal case when no failures occur, indicating that when no failures occur, the BRE policy actually incurs slightly more

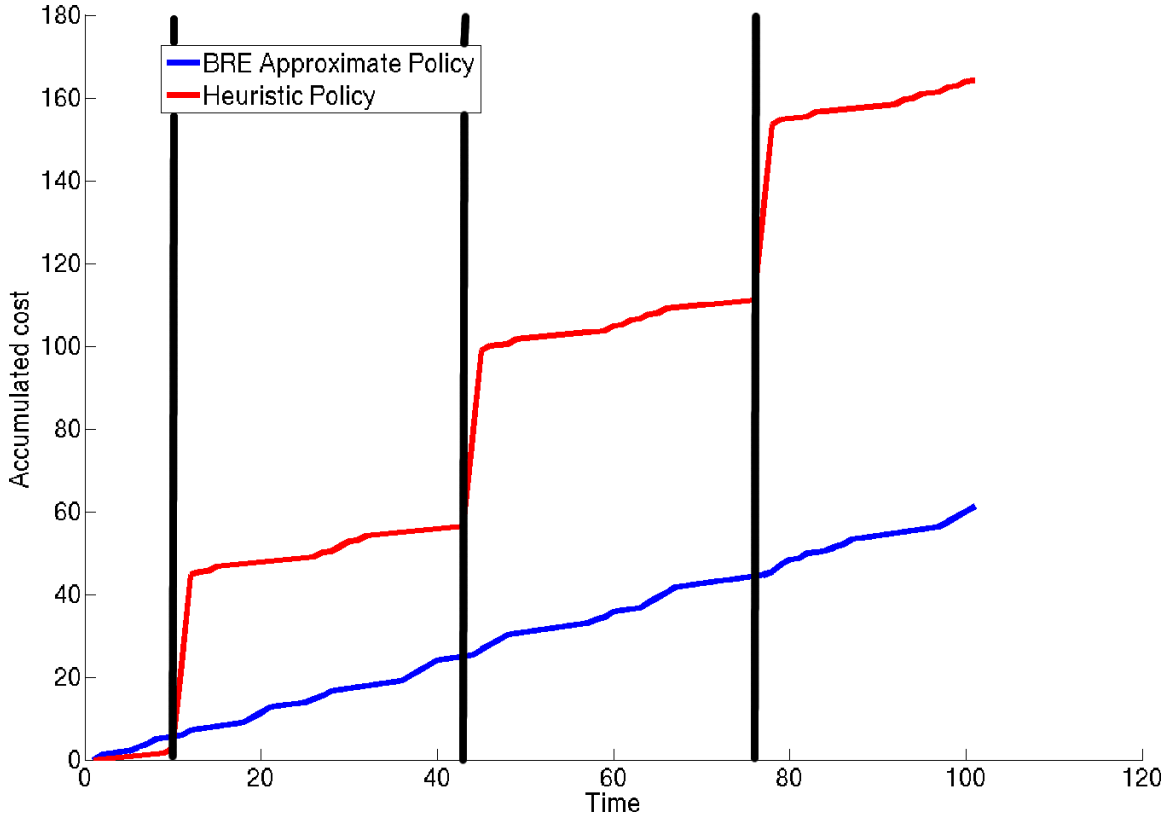


Figure 7-22: Total accumulated cost $G(t) = \sum_{t'=0}^{t} g(\mathbf{x}(t'), \mathbf{u}(t'))$, as a function of time t, for the BRE policy and the heuristic policy. The total length of the simulated mission is 100, and sensor failure events are depicted by the vertical lines.

cost than the heuristic over time. This effect is noticeable at the start of the mission up until the first sensor failure occurs at $t = 10$. The effect is to be expected, since the backup UAV that the BRE policy flies out to the surveillance area burns extra fuel, which is penalized by the cost function. However, when a failure occurs, the BRE policy avoids a surveillance gap (and the resulting large cost penalty) since the backup UAV is able to immediately take over. In contrast, heuristic policy suffers a surveillance gap, and incurs a large penalty, when a failure occurs. Over time, the effect of these intermittent, large costs incurred by the heuristic easily outweighs the extra fuel costs incurred by the BRE policy. As a result, by the end of the mission, the BRE policy has incurred significantly less total cost than the heuristic policy.

7.3.3 Flight Results with Heterogeneous UxV Team

A further series of flight experiments utilized a heterogeneous team of UxVs engaged in a complex search and track mission. The mission therefore required use of the "heterogeneous team" extension to the persistent surveillance mission discussed in Section 5.4.3. In these experiments, there were three different classes of UxVs available to the planner: a quadrotor UAV class, a car UGV class, and a unique three-winged UAV class. Furthermore, there were four possible types of tasks that needed to be performed, depending on the phase of the mission: search for targets of interest, track any discovered ground-based targets, track any discovered flying targets, and "perch" to stealthily observe any discovered targets. These task types were generated as appropriate by the mission planner as the mission progressed (a detailed discussion of the mission evolution is below).

To fit this general description of the UxV and task types into the persistent mission framework, it is necessary to specify the capabilities matrix M given by Eq. 5.8. We adopt the convention that quadrotors, cars, and three-wings have indices 1, 2, and 3, respectively, in the capabilities matrix. Furthermore, the search, track ground targets, track air targets, and perch tasks have indices 1, 2, 3, and 4, respectively. With these conventions, the capabilities matrix is given by

$$M = \begin{pmatrix} 1.0 & 0.8 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}.$$

(In other words, the quadrotors can do all tasks except perch, cars are best at tracking ground targets but can also perch, and three-wings can only perch.) The chosen mission used a total of three quadrotor UAVs, two car UGVs, and one triwing UAV. The surveillance location was taken as $Y_s = 2$, and the fuel capacity of each UxV was $F_{max} = 15$ (with $\Delta f = 1$). The same communications function as in the previous experiments [Eq. (7.2)] was utilized. With these parameters, the total size of the state space was 4.251×10^{12}. The parameters $p_{sensorfail}$ and p_{nom} were set at 0.02 and

0.95, respectively. To carry out the flight experiments, an approximate policy for the problem formulation was computed using BRE.

A sequence of still images of a single flight experiment is shown in Figure 7-23. The mission for this experiment was to use a team of three quadrotor UAVs, one three-winged UAV, and two ground-based UGVs to perform a persistent search and track mission. This "blue team" is under the control of the multi-UAV planning architecture and is initially located in a group at the near end of the flight volume (frame 1). Unknown to the blue team at the beginning of the mission, the search area (located at the far end up the flight volume) contains an enemy "red team" quadrotor (initially landed) and car UGV; these red team vehicles must be found and tracked. At the start of the mission, the mission planner requests a single vehicle to perform a search task in the surveillance area. As a result, two quadrotors launch to (1) search the area and (2) provide a communications link back to base (frame 2). Shortly after the search begins, the red team quadrotor is discovered by the searching blue team UAV, and the blue UAV drops down to inspect it (frame 3). At the same time, the mission planner requests the three-winged UAV to "perch" near the red quadrotor, and the persistency planner launches the blue three-wing and commands it to the search area (frame 4). Meanwhile, the blue quadrotor continues searching and discovers the red team UGV. After dropping down to inspect the red team UGV (frame 5), the mission planner requests a blue team UGV in order to monitor the red team UGV, and a blue UGV is commanded to the search area in response (frame 6). Shortly after the blue UGV arrives on station and begins monitoring the red UGV, the red UGV begins moving, and the blue UGV begins tracking it (frame 7). After several minutes, the blue quadrotor providing the communications link to base is recalled for refueling, and the third blue quadrotor launches to take its place (frame 8). Immediately following the swap, the red quadrotor takes off and begins maneuvering around the flight volume. The perching blue three-wing observes this event and notifies the searching blue quadrotor to begin tracking the red quadrotor (frame 9). Simultaneously, the mission planner requests a second blue quadrotor in the search area so that the searching task can continue in parallel with tracking the

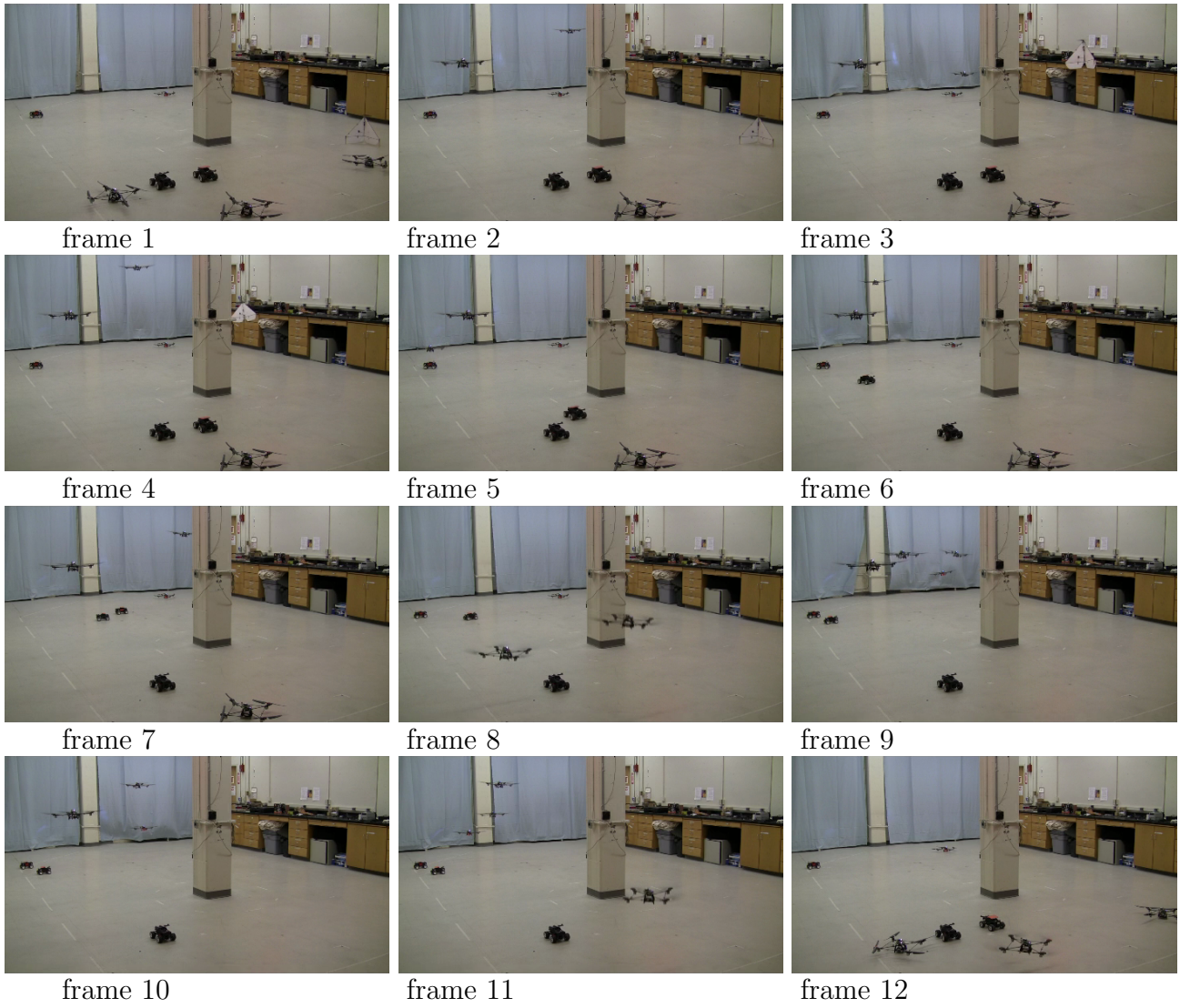


Figure 7-23: Persistent surveillance flight experiment in RAVEN, using a heterogeneous team of UxVs. The base location for the "blue team" is located at the near end of the flight volume, while the surveillance area is at the far end.

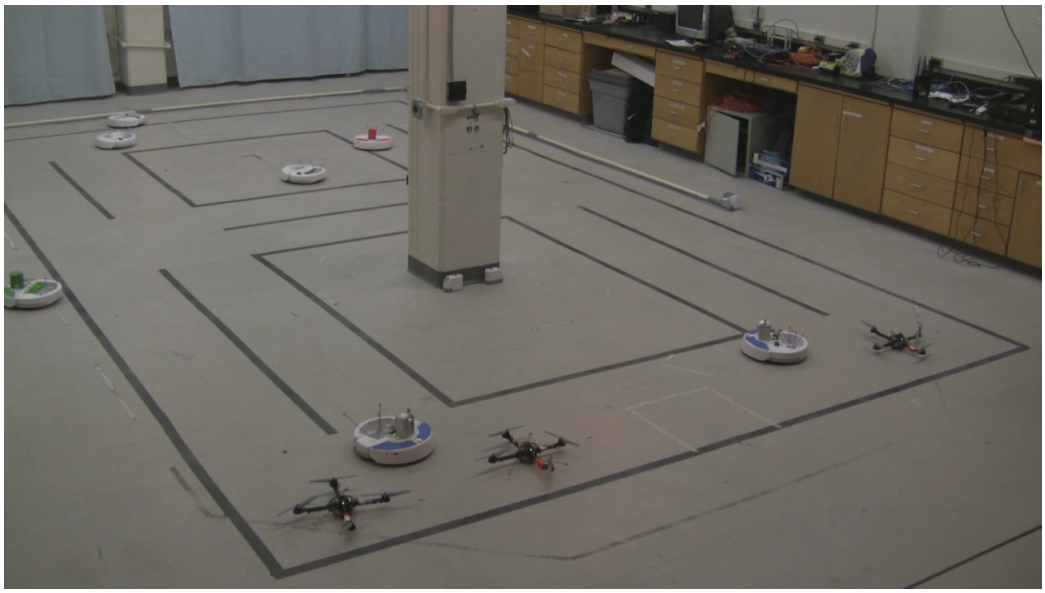


Figure 7-24: Heterogeneous search and track mission layout, at the start of the mission. The searching team consists of three quadrotor UAVs and two blue iRobot UGVs (bottom of the image). The objective is to find and track the red and green UGVs, which drive randomly in the presence of several "drone" UGVs (top of the image).

red quadrotor. In response, the now-refueled blue quadrotor at base launches and moves to the search area (frame 10). After a few more minutes, the tracking blue quadrotor must return to base for refueling, and the searching quadrotor takes its place tracking the red quadrotor (frame 11). Finally, the experiment ends and all blue vehicles return to base for landing (frame 12).

7.3.4 Adaptive Flight Results with Heterogeneous UxV Team

A further series of flight experiments was conducted using a heterogeneous team of UAVs and UGVs to carry out a vision-based search and track mission. In this mission, the targets of interest were two UGVs that moved about randomly in the presence of several other UGV "drones" that made it more difficult to track the targets. A photo of the mission layout is shown in Figure 7-24. The searching UGVs were equipped with onboard cameras, and ran vision processing software that attempted to locate the targets of interest. The UAVs in this mission were not equipped with cameras, but instead ran simulated sensor software that allowed the UAVs to "see"

the targets when they were within a small distance (in this test, 1.5 meters) of the target. In the mission, the UAVs were assumed to be primarily equipped for searching for the targets, while the UGVs were best suited for tracking the targets once they are detected. However, due to the presence of the drones, it was difficult for the UGVs to maintain perfect tracking at all times, since occasionally, a drone would move between the target and the tracking UGV, causing the tracking UGV to lose sight of the target. When this occurred, the mission manager was programmed to task a UAV with tracking the target temporarily until the UGV could regain sight of the target.

This mission specification was mapped into a persistent surveillance problem formulation as follows. Similar to the formulation presented in the previous section, we adopt the convention that UAVs and UGVs have indices 1 and 2, respectively, in the vehicle capabilities matrix. Furthermore, there are three task types: search, track using a UGV, and track using a UAV. These task types have indices 1, 2, and 3, respectively. With these conventions, the capabilities matrix is given by

$$M = \begin{pmatrix} 1.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{pmatrix}. \tag{7.4}$$

Note that the tracking tasks are split into two categories (track using a UGV, and tracking using a UAV) in order to capture the assumption that when a UGV loses sight of the target, a UAV must be called in to help. To allow the problem formulation to recognize the dependence of these tasks on each other, a small probability of task generation

$$\rho_{23}^{+} = 0.1$$

was set according to the discussion in Section 5.4.3. Thus, the planner can anticipate the possibility that the UGV will lose track of its target and require a UAV to assist. The chosen mission used a total of three quadrotor UAVs and two car UGVs. The surveillance location was taken as $Y_s = 2$, and the fuel capacity of each UxV was $F_{max} = 15$ (with $\Delta f = 1$). Again, the same communications function as in the

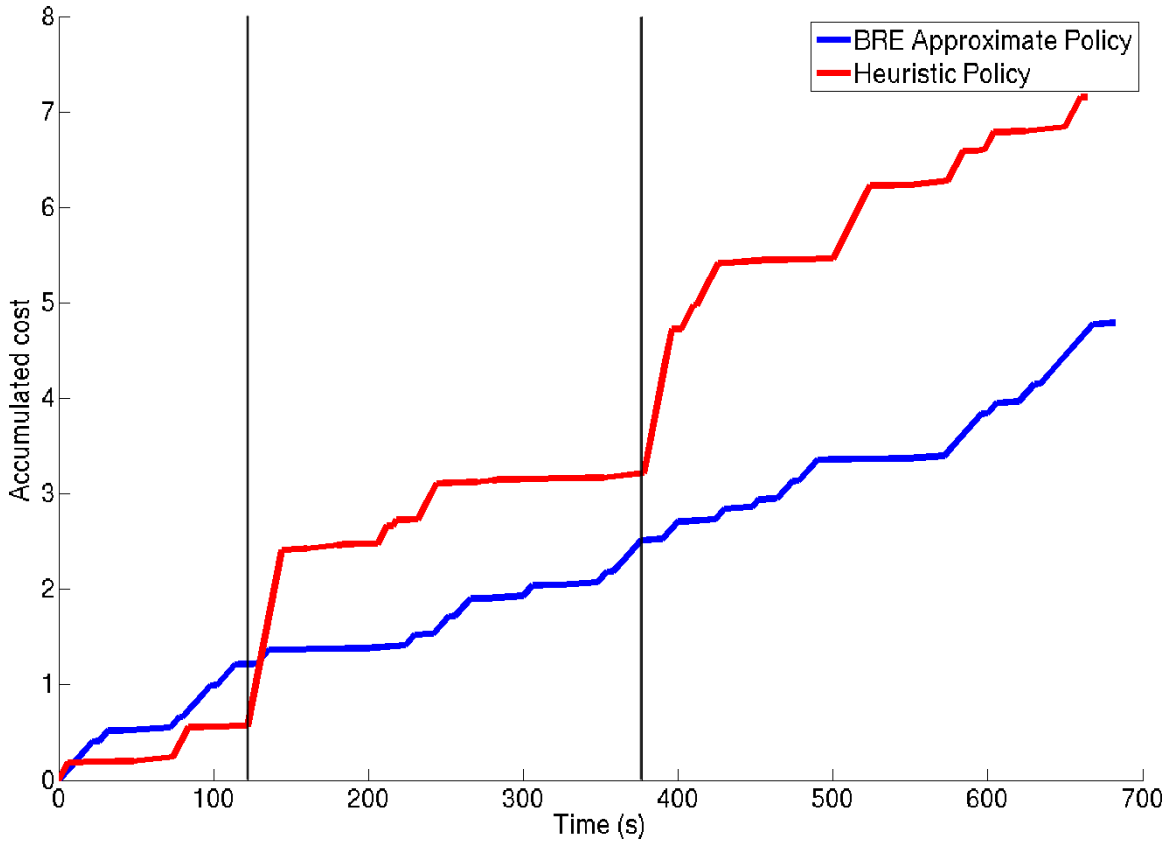


Figure 7-25: Total accumulated cost, as a function of time, for the BRE policy and the heuristic policy for a flight test with a heterogeneous UxV team. The total length of the mission is 11 minutes, and sensor failure events are depicted by the vertical lines. For this mission, the BRE policy is computed using the true value of $p_{sensorfail} = 0.030$, and no adaptation occurs.

previous experiments [Eq. (7.2)] was utilized. With these parameters, the total size of the state space was 4.724×10^{10}. The parameter p_{nom} was set at 0.95, and $p_{sensorfail}$ was varied as described below. To carry out the flight experiments, an approximate policy for the problem formulation was computed using BRE.

For the first set of flight experiments, the heterogeneous search and track mission was run twice, using the BRE policy for the first run and the heuristic policy for the second run. In order to create a fair comparison between the two policies in the presence of sensor failures, two failures were pre-programmed to occur during each run. The failures occurred two and six minutes after the start of each mission, and the total duration of each mission was 11 minutes. Therefore, since both policies experienced the same number and timing of failures, differences in the total cost

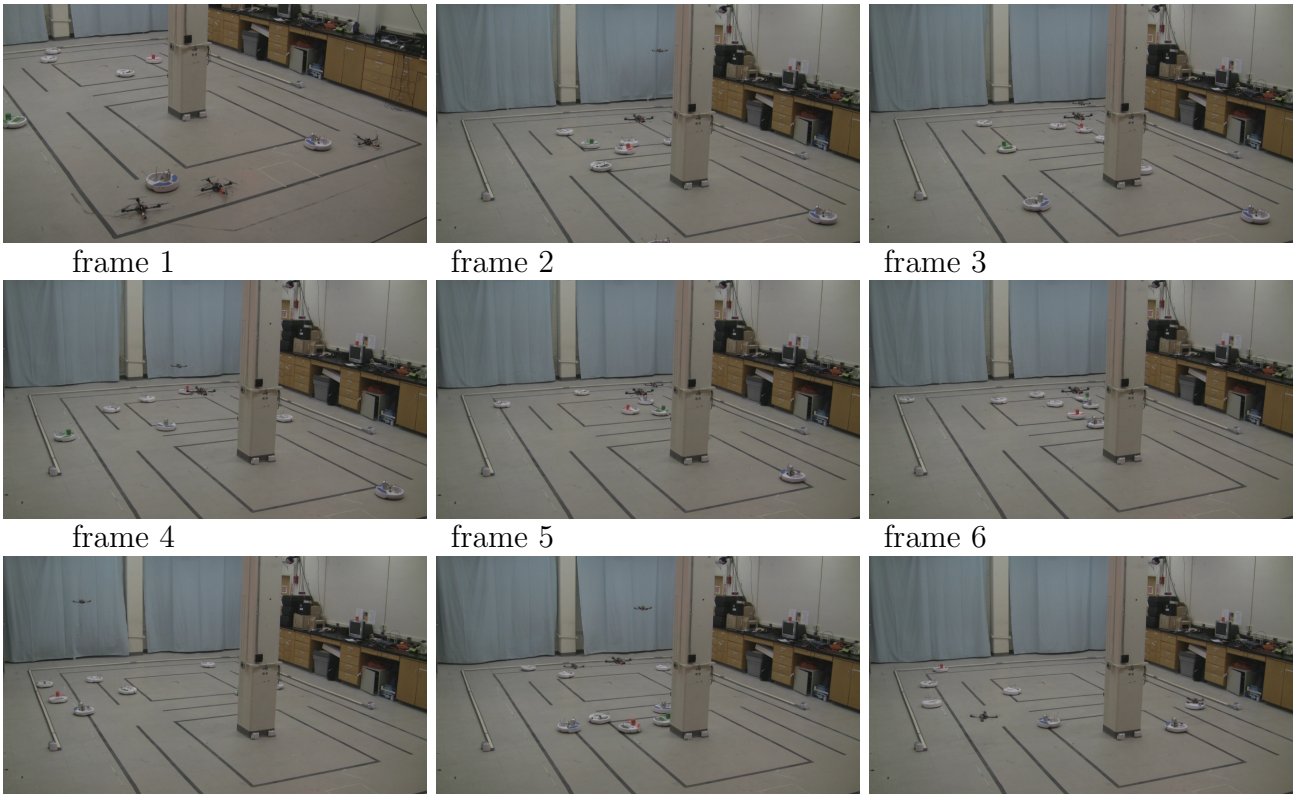


Figure 7-26: Adaptive persistent surveillance flight experiment in RAVEN, using a heterogeneous team of UxVs. The base location for the "blue team" is located at the near end of the flight volume, while the surveillance area is at the far end.

accrued over the course of the missions are due only to differences in the policies themselves. During the mission, the control policies were evaluated once every 10 seconds. Since the length of the mission was 11 minutes (660 seconds) and two sensor failures occurred during this time, the effective $p_{sensorfail}$ for these mission scenarios was $\frac{2}{660/10} = 0.030$. For the first set of flight experiments, the BRE policy was computed using the true value of $p_{sensorfail}$, and no adaptation was used (the same fixed policy was used over the entire course of the mission).

The total accumulated cost of each mission was computed using Eq. 7.3, and the results are shown in Figure 7-25. The flight data shown in Figure 7-25 confirms the results found in simulation (Figure 7-22), where it was observed that the heuristic policy incurs slightly lower cost than the BRE policy when no failures occur (due to the fact that the heuristic policy does not send a backup vehicles), but incurs large

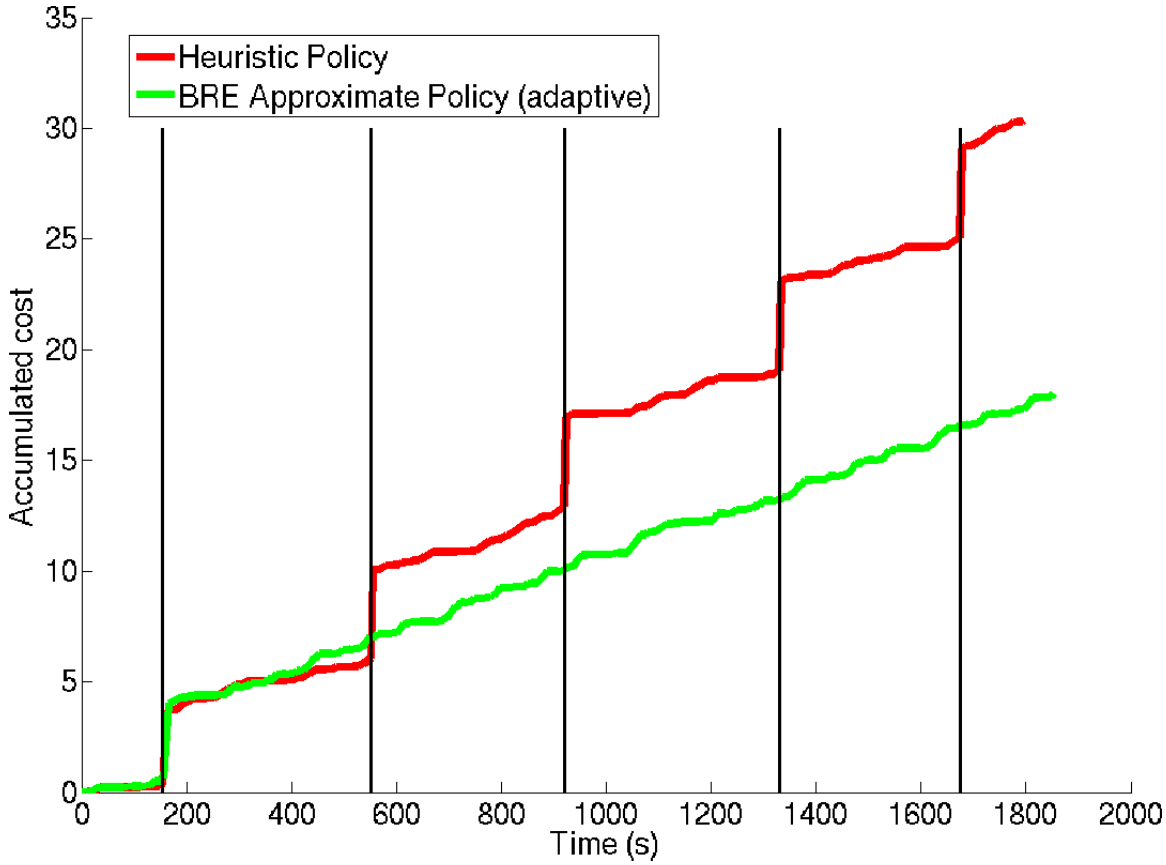


Figure 7-27: Total accumulated cost, as a function of time, for the adaptive BRE policy and the heuristic policy for a flight test with a heterogeneous UxV team. The total length of the mission is 31 minutes, and sensor failure events are depicted by the vertical lines. The BRE policy is initially computed using an incorrectly "optimistic" value of $p_{sensorfail} = 0.0$. After the first failure occurs, the BRE policy adapts its behavior, and subsequently avoids a surveillance gap when the subsequent failures occur.

cost penalties when failures occur. As a result, the BRE policy again incurs lower total cost over the course of the mission compared to the heuristic.

Another flight experiment was conducted using the BRE solver in conjunction with the adaptive MDP architecture. In this flight experiment, the initial BRE policy was computed using an incorrect estimate of $p_{sensorfail}$ (namely, $p_{sensorfail} = 0$). This resulted in an initial policy that was too "optimistic": it did not account for the possibility of failures, nor did it send out a backup vehicle. Again, the search and track mission was run, this time with the adaptation loop running. When improved estimates of $p_{sensorfail}$ arrived, the adaptation loop used the previously-computed

policy as the initial policy given to the BRE algorithm. It was observed that this bootstrapping procedure reduced the number of policy iterations needed to compute the updated policy.

Figure 7-26 shows several photo frames of the mission as it was running. At the start of the mission, all "blue team" vehicles are located at the near end of the flight volume, while all target vehicles and drones are located in the far end (frame 1). Initially, since the BRE policy is overly optimistic, only a single UAV begins searching while another provides a communications link (frame 2). After a short period of time, the searching UAV discovers the red target, and as a result one of the blue team UGVs is called out to track it (frames 3–4). The green target is discovered shortly afterward, and the second blue team UGV is called out to track it (frames 5–6). Then, the searching UAV experiences a failure, resulting in a short coverage gap (frame 7). The adaptive framework updates its estimate of $p_{sensorfail}$ and sends out two searching UAVs for the rest of the mission (frame 8). Finally, the mission ends and all vehicles return to base (frame 9).

Cost data for this flight test are shown in Figure 7-27. In the figure, notice that the adaptive BRE policy initially performs almost identically to the heuristic policy, since both policies do not employ a backup vehicle. Therefore, when the first failure occurs, both policies incur a coverage gap and a large cost penalty. However, after the failure occurs, the adaptive MDP architecture adjusts its estimate of $p_{sensorfail}$ and recomputes the BRE policy, after which point the policy recognizes the increased risk of failures occurring and sends out a backup vehicle to mitigate this risk. As a result, when the subsequent failures occur later in the mission, the BRE policy maintains continuous coverage due to the backup vehicle, while the heuristic policy again suffers coverage gaps. These results demonstrate that the BRE algorithm in conjunction with the adaptive MDP architecture allow the system to effectively and quickly adjust its behavior as better parameter estimates are generated.

7.3.5 Boeing Flight Results

In this section, we provide initial flight results from joint work with the Boeing Company [32]. This work carried out a number of flight experiments utilizing the persistent surveillance problem formulation and mission architecture depicted in Figure 7-4. The flight experiments were conducted in the Boeing Vehicle Swarm Technology Laboratory (VSTL), an environment for testing a variety of vehicles in an indoor, controlled environment [134]. VSTL has been developed in conjunction with MIT's RAVEN testbed, allowing easy interoperability between the two flight test environments. Similar to RAVEN, the primary components of the VSTL are:

1. A camera-based motion capture system for reference positions, velocities, attitudes and attitude rates;

2. A cluster of off-board computers for processing the reference data and calculating control inputs;

3. Operator interface software for providing high-level commands to individual and/or teams of agents.

The VSTL environment is shown in Figure 7-28.

The mission scenario implemented is a persistent surveillance mission with static search and dynamic track elements, similar to the mission scenarios presented earlier. The mission scenario employed a heterogeneous team of six agents, consisting of four UAVs and two UGVs, that begin at the base location Y_b and are tasked to persistently search the surveillance area. As threats are discovered via search, additional agents are called out to provide persistent tracking of the dynamic threat. Figure 7-29 shows the initial layout of the mission. Agents $1-6$ are shown in the base location Y_b, while threats $1-3$ are located in the surveillance region, $Y_s = 2$. The fuel capacity of each UxV was $F_{max} = 15$ (with $\Delta f = 1$). Similar to the previous experiment, the same communications function [Eq. (7.2)] and vehicle capabilities matrix [Eq. (7.4)] was utilized. With these parameters, the total size of the state space was 4.251×10^{12}. The parameter p_{nom} was set at 0.95, and $p_{sensorfail}$ was varied as described below.

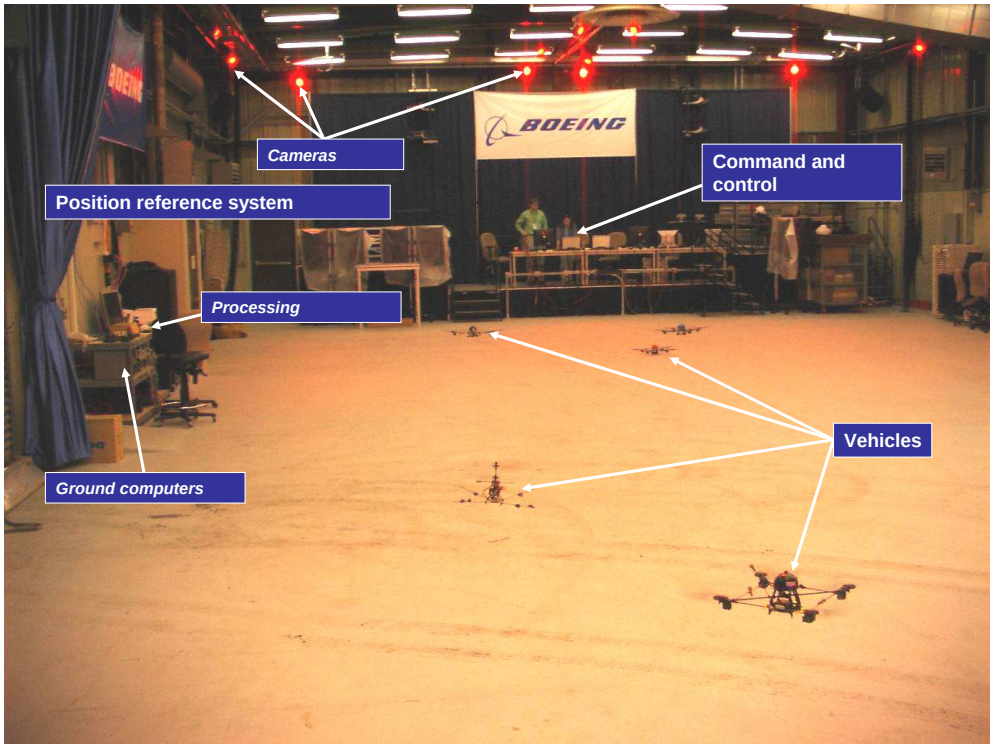


Figure 7-28: Boeing Vehicle Swarm Technology Laboratory [134]

Two different types of flight experiments were conducted. In the first experiment type, $p_{sensorfail}$ was set to zero, to establish baseline flight results in which no sensor failures occur. In the second experiment type, $p_{sensorfail}$ was set to 0.03, and several failures did occur over the course of the mission.

Under the no-failure scenario, a persistent surveillance mission is initiated with three unknown threats that must be discovered and tracked. Figure 7-30 shows the flight status of each of the agents as well as the coverage of the surveillance and communication regions (Y_s, Y_0 respectively). In this case, two of the threats were detected early (around 100 seconds into the mission), by agents 5 and 1, resulting in agents 1, 2 and 5 being called out the surveillance region simultaneously. It is important to note that, in addition to the track tasks generated by the discovery of threats, the search task is continuous and we see in the combined coverage plot of Figure 7-30 that the surveillance region is persistently surveyed over the duration of the mission. Also of note in Figure 7-30, the vertical black lines show the initiation of

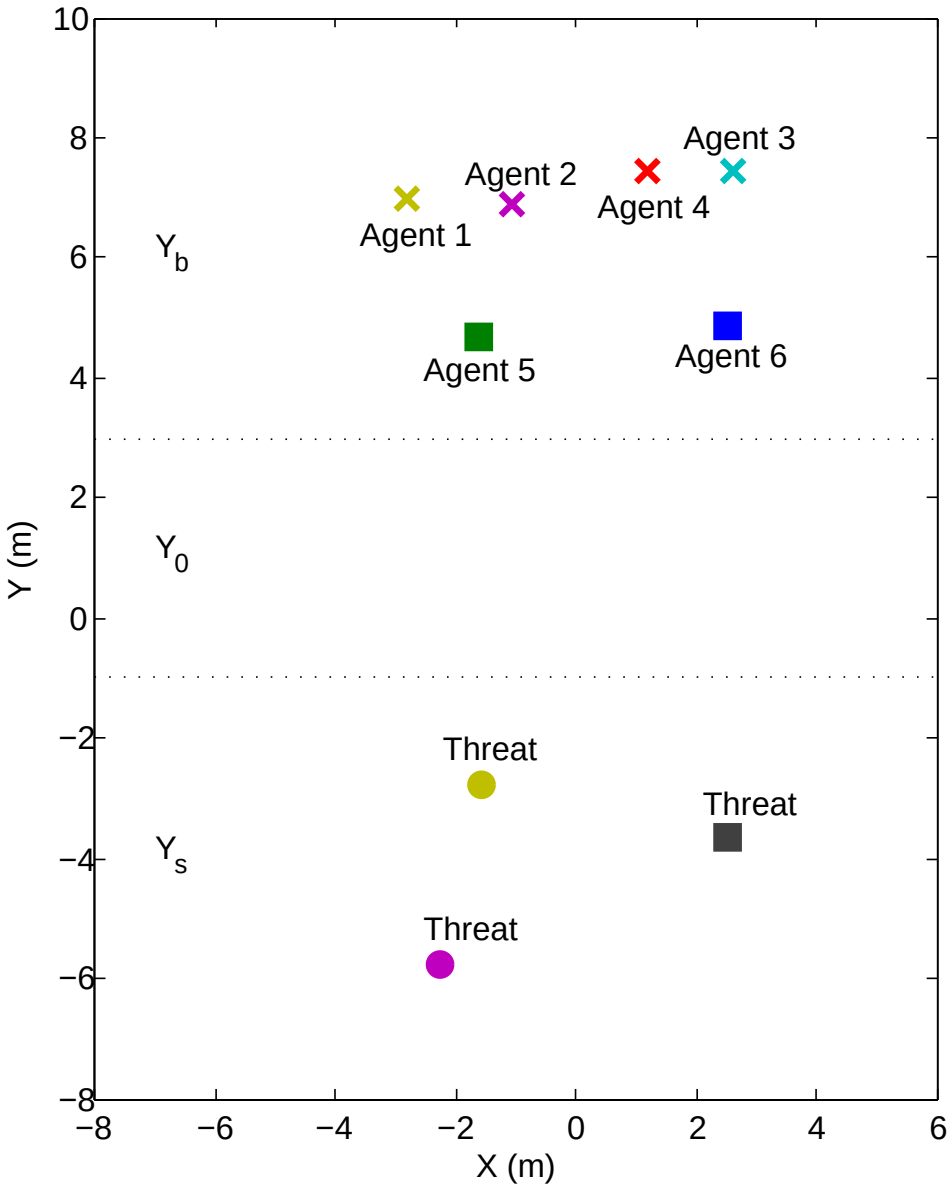


Figure 7-29: Boeing VSTL [134] mission setup with six autonomous agents and three unknown threats.

important events occurring during the flight test, such as the discovery of additional targets and commands sent from the planner.

In the second scenario, a similar mission is initiated with three unknown threats that must be discovered and tracked. However, in this case, the control policy is computed using $p_{sensorfail} = 0.03$ and therefore accounts for the possibility of vehicle failures over the course of the mission. Results for this scenario are shown in Figure 7-31, where again, the vertical black lines indicate the occurrence of important events during the mission. During the mission, Agent 2 experiences a sensor failure that is detected by its onboard health monitoring module and transmitted to the control policy, which has anticipated the possibly of a failure occurring and pre-positioned a backup vehicle in the surveillance location, allowing for continuous coverage despite the failure. Furthermore, note that during this mission, one of the agents crashed during takeoff due to a momentary loss of signal from the camera-based motion capture system. The control policy gracefully recovers from this crash and sends out another vehicle in its place, allowing the mission to continue.

7.4 Summary

This chapter has presented a number of both flight and simulation results demonstrating the use of the persistent surveillance problem formulation, the adaptive MDP architecture, and the BRE algorithms in solving a number of complex, realistic multi-UxV missions. In addition, we have compared our approach with several heuristic techniques and highlighted the performance benefits that our approach offers over these techniques. The flight results, carried out on two realistic flight testbeds (the MIT RAVEN testbed [80] and the Boeing VSTL testbed [134]), confirm that the technologies presented in this thesis can be successfully applied to challenging, real-world, multi-agent robotic applications.

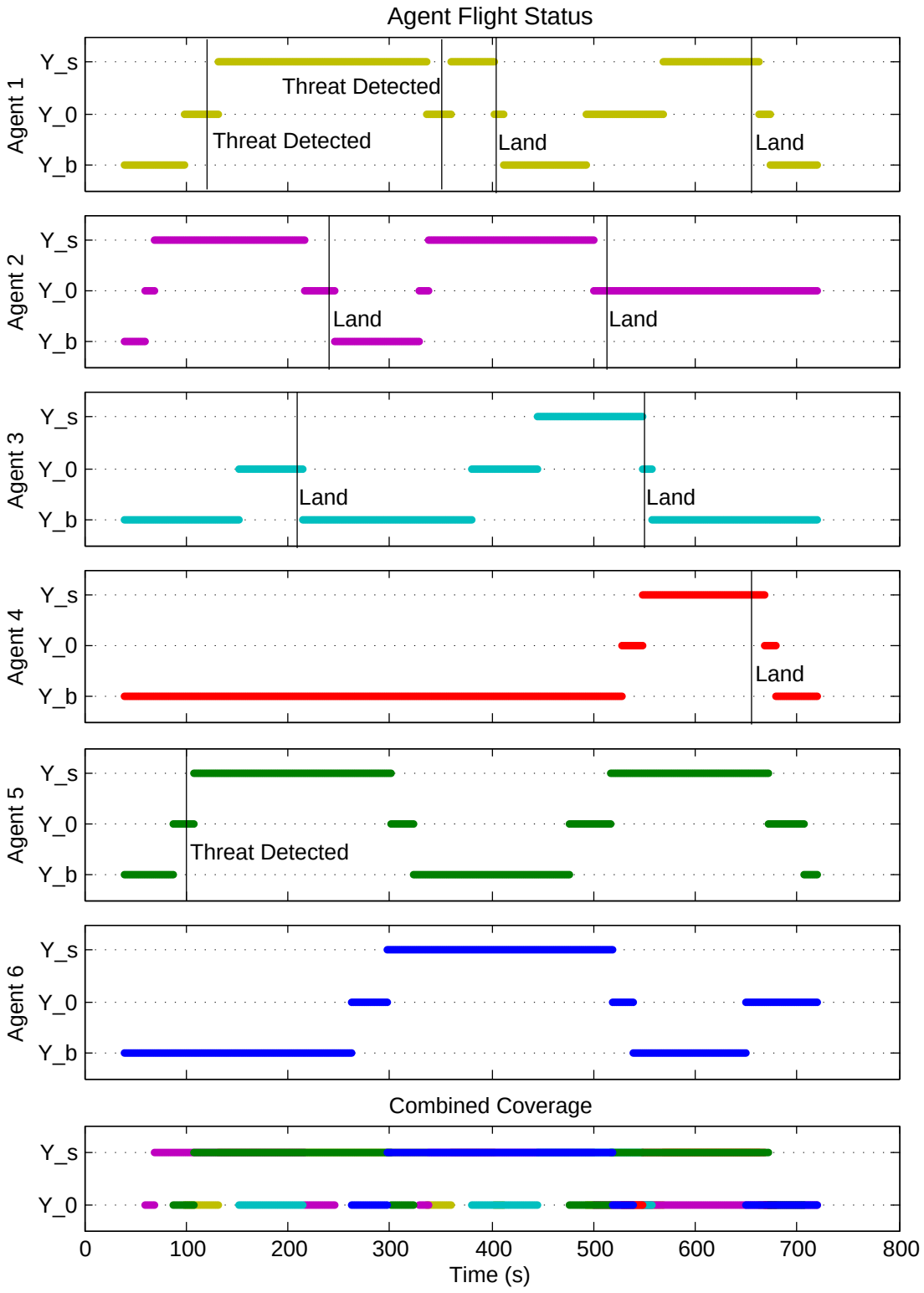


Figure 7-30: VSTL flight results with six agents, three threats and no induced failures

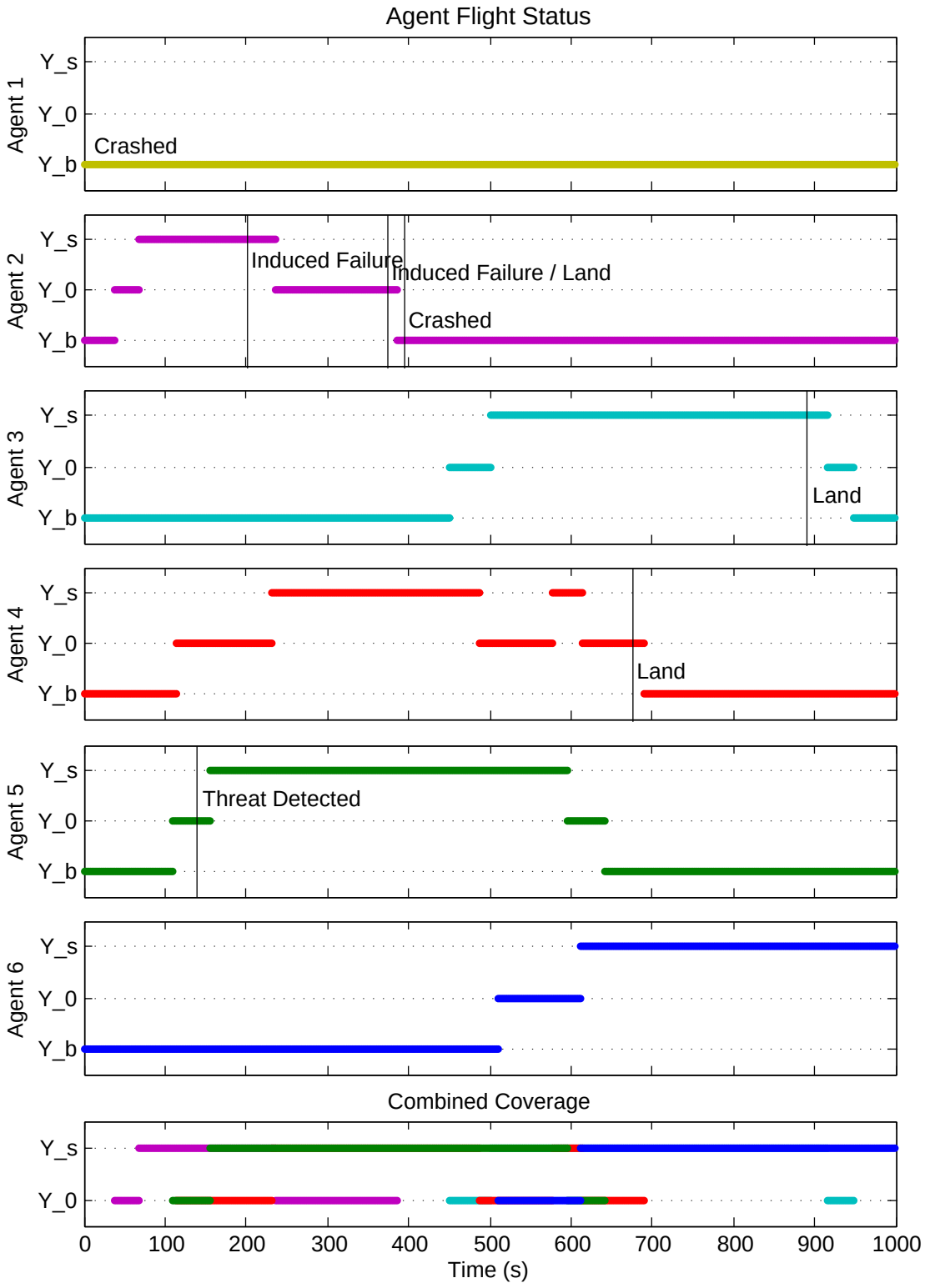


Figure 7-31: VSTL flight results with six agents, three threats and induced failures that degrade agent capability

Chapter 8

Conclusions and Future Work

This thesis has presented contributions in three main areas: kernel-based approximate dynamic programming algorithms, online adaptation to unknown and/or time-varying MDP models, and robust planning in multi-agent robotic systems that are subject to failures. In particular, in the area of kernel-based approximate dynamic programming, the thesis has:

- Developed the basic, model-based Bellman residual elimination (BRE) approach to approximate dynamic programming, and studied its theoretical properties. In particular, the thesis has demonstrated how the problem of finding a cost-to-go solution, for which the Bellman residuals are identically zero at the sample states, can be cast as a regression problem in an appropriate Reproducing Kernel Hilbert Space (RKHS). It then explained how any kernel-based regression technique can be used to solve this problem, leading to a family of BRE algorithms. These algorithms were proved to converge to the optimal policy in the limit of sampling the entire state space. Furthermore, we have shown that the BRE algorithm based on Gaussian process regression can provide error bounds on the cost-to-go solution and can automatically learn free parameters in the kernel.

- Developed a multi-stage extension of the basic BRE approach. In this extension, Bellman residuals of the form $|\widetilde{J}_\mu(i) - T_\mu^n \widetilde{J}_\mu(i)|$, where $n \geq 1$ is an integer,

are eliminated at the sample states. As a result of this extension, a kernel function arises that automatically captures local structure in the state space. In effect, this allows the multi-stage BRE algorithms to automatically discover and use a kernel that is tailored to the problem at hand. Similar to the basic BRE approach, multi-stage BRE converges to the optimal policy in the limit of sampling the entire state space.

- Developed a model-free variant of BRE, and demonstrated how the general, multi-stage BRE algorithms can be carried out when a system model is unavailable, by using simulated or actual state trajectories to approximate the data computed by the BRE algorithms. Convergence results for these model-free algorithms were proved. Furthermore, the thesis presented results comparing the performance of model-based and model-free BRE against model-based and model-free LSPI [93], and showed that both variants of BRE yield more consistent, higher performing policies than LSPI in a benchmark problem.

In the area of online MDP adaptation, the thesis has:

- Presented an architecture for online MDP adaption that allows the unknown model parameters to be estimated online; using these parameter estimates, a policy for the corresponding MDP model is then re-computed in real-time. As a result, the adaptive architecture allows the system to continuously re-tune its control policy to account for better model information obtained through observations of the actual system in operation, and react to changes in the model as they occur.

- Discussed how the adaptive architecture permits any MDP solution technique to be utilized to recompute the policy online. In particular, the architecture allows BRE-based algorithms to be used to compute approximate policies for large MDPs quickly.

- Validated the architecture through hardware flight tests carried out in the MIT RAVEN indoor flight facility [80]. These flight tests demonstrate the successful

ability of the adaptive architecture to quickly adjust the policy as new model information arrives, resulting in improved overall mission performance.

In the area of planning in multi-agent robotic systems, the thesis has:

- Formulated the basic persistent surveillance problem as an MDP, which captures the requirement of scheduling assets to periodically move back and forth between the surveillance location and the base location for refueling and maintenance. In addition, the problem formulation incorporates a number of randomly-occurring failure scenarios (such as sensor failures, unexpected fuel usage due to adverse weather conditions, etc) and constraints (such as the requirement to maintain a communications link between the base and agents in the surveillance area). We showed that the optimal policy for the persistent surveillance problem formulation not only properly manages asset scheduling, but also *anticipates* the adverse effects of failures on the mission and takes actions to mitigate their impact on mission performance.

- Highlighted the difficulties encountered in solving large instances of the persistent surveillance problem using exact methods, and demonstrated that BRE can be used to quickly compute near-optimal approximate policies.

- Presented a fully autonomous UxV mission architecture which incorporates the persistent surveillance planner into a larger framework that handles other necessary aspects including including low-level vehicle control; path planning; task generation and assignment; and online policy adaptation using the adaptive MDP architecture.

- Presented flight experiments, conducted in the MIT RAVEN indoor flight facility [80], that demonstrate the successful application of our problem formulation, BRE solution technique, and mission architecture to controlling a heterogeneous team of UxVs in a number of complex and realistic mission scenarios. Furthermore, we demonstrated performance benefits of our approach over a determin-

istic planning approach that does not account for randomness in the system dynamics model.

8.1 Future Work

There are several areas related to the topics discussed in this thesis that could be explored in future work:

Adaptive state sampling The question of how to best select the sample states used in the BRE approach is important in determining the ultimate performance of the BRE algorithms. Intuitively, a higher density of sample states in a particular region of the state space will tend to increase the accuracy of the cost-to-go function in that region. Furthermore, since the running time of the algorithms increases with the number of sample states [Eq. (3.43)], there is a tradeoff between number of samples and performance. Therefore, it is desirable to choose a set of sample states that adequately represent the important regions of the state space (where here "important" means "likely to be visited under the optimal policy"), while being of practical size for the computational resources at hand. Of course, without knowledge of the optimal policy, it may be difficult to determine which states are important a priori. This sample state selection problem is closely related to the central exploration vs. exploitation issue in reinforcement learning [15, 100]. There are several strategies that could be employed to select the sample states. In some problems, simple uniform sampling of the state space may be acceptable if the state space is small enough. If the problem is such that the important states can be inferred from prior knowledge, then this knowledge could be used to select the sample states. An interesting additional possibility, similar to active learning techniques [63, 159], is to use the posterior error covariance information computed by BRE(GP) to automatically help determine the regions of the state space where more samples are needed. In this approach, states would be adaptively re-sampled to include more states in regions of the state space where the posterior error covariance is large.

BRE with sparse kernel methods The BRE algorithms presented in this thesis employ kernel-based regression techniques such as support vector regression and Gaussian process regression in order to compute the cost-to-go function. As a result of using these techniques, the computational complexity of the BRE algorithms is cubic in n_s, the number of sample states used in the training process (see, e.g., Eqs. (3.43), (4.10), and (4.15)). It is well-known that this cubic dependence on the number of training data can be a limiting factor for kernel-based techniques as the number of training data points grows [128, Ch. 8]. Nevertheless, in some cases, it may be desirable to use many sample states when applying BRE in order to increase the performance of the algorithm. In order to allow kernel-based techniques to scale to large amounts of training data, researchers have investigated a number of sparse approximation methods. For example, [169] applied the Nyström method to approximate the dominant eigenvectors of the kernel Gram matrix, and [94] proposed a method to select and use only a representative subset of the training data. For an overview of these sparse methods, see [43], [128, Ch. 8], and [141, Sec. 10.2]. In the reinforcement learning domain, [65–67] applied sparse kernel methods to temporal difference algorithms that learn the cost-to-go function or Q-factors on-line. It would be interesting to apply similar sparse methods to the BRE algorithms developed in this thesis; such methods could allow BRE to handle more sample states in a computationally tractable way.

Adaptive MDP architecture extensions With respect to the adaptive MDP architecture presented in Chapter 6, there are a number of interesting future research areas that could be explored. First, in the flight experiments done to date, the same fuel usage and sensor failure models were assumed for all vehicles. A small but interesting modification would be to run a separate model estimator for every vehicle, allowing for the possibility that vehicles degrade at different rates, for example. Another area would be modification of the system cost function to explicitly reward exploration, where vehicles would be rewarded

for taking actions that reduce the uncertainty in the system parameters. This idea is closely related to Bayesian reinforcement learning, where a distribution over models or Q-factors is maintained by the learning agent and used to balance the exploration vs. exploitation tradeoff [56, 123, 132, 149]. One way to accomplish the goal of explicitly rewarding exploration is to embed the dynamics of the model estimator as part of the MDP, allowing the controller to estimate the effects of its actions on the uncertainty in the model. This could lead to interesting policies where the vehicles would actively and cooperatively act in ways to ensure that the system model was as accurate as possible, while simultaneously satisfying the objectives of the primary mission. An initial approach to this idea can be found in [129], but future work could expand the complexity of the models considered as well as explore the effect of embedding other types of model estimators into the exploration procedure.

Bibliography

[1] OpenMPI - a high performance message passing library. `http://www.open-mpi.org/`.

[2] STL-MPI library. `http://clustertech.com/`.

[3] M. Alighanbari. Task assignment algorithms for teams of UAVs in dynamic environments. Master's thesis, MIT, 2004.

[4] M. Alighanbari. *Robust and Decentralized Task Assignment Algorithms for UAVs*. PhD thesis, MIT, 2007.

[5] M. Alighanbari and J. How. Decentralized task assignment for unmanned aerial vehicles. In *Decision and Control, 2005 and 2005 European Control Conference*, December 2005.

[6] American Institute of Aeronautics and Astronautics. Worldwide UAV roundup. `http://www.aiaa.org/images/PDF/WilsonChart.pdf`, 2007.

[7] K. Andersson, W. Hall, S. Atkins, and E. Feron. Optimization-based analysis of collaborative airport arrival planning. *Transportation Science*, 37(4):422–433, November 2003.

[8] D. Andre, N. Friedman, and R. Parr. Generalized prioritized sweeping. In M. Jordan, M. Kearns, and S. Solla, editors, *NIPS*. The MIT Press, 1997.

[9] A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

[10] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

[11] C. Ashcraft, S. Eisenstat, J. Liu, and A. Sherman. Comparisons of three distributed algorithms for sparse Cholesky factorization. In J. Dongarra, P. Messina, D. Sorensen, and R. Voigt, editors, *PPSC*, pages 55–56. SIAM, 1989.

[12] K. Astrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[13] L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *ICML*, pages 30–37, 1995.

[14] C. Barnhart, F. Lu, and R. Shenoi. Integrated airline scheduling. In G. Yu, editor, *Operations Research in the Airline Industry*, volume 9 of *International Series in Operations Research and Management Science*, pages 384–422. Kluwer Academic Publishers, 1998.

[15] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1993.

[16] M. Belkin and P. Niyogi. Semi-supervised learning on Riemannian manifolds. *Machine Learning*, 56(1-3):209–239, 2004.

[17] M. Belkin and P. Niyogi. Towards a theoretical foundation for Laplacian-based manifold methods. In P. Auer and R. Meir, editors, *COLT*, volume 3559 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 2005.

[18] R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60:503–515, 1954.

[19] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[20] K. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2(2), 2000.

[21] C. Berg, J. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer-Verlag, New York, 1984.

[22] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2007.

[23] D. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. `http://web.mit.edu/people/dimitrib/Tempdif.pdf`, 1996.

[24] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[25] D. Bertsimas and S. S. Patterson. The air traffic flow management problem with enroute capacities. *Operations Research*, 46(3):406–422, May-June 1998.

[26] L. Bertuccelli and J. How. Estimation of non-stationary Markov chain transition models. In *Proceedings of the 2008 IEEE Conference on Decision and Control*, 2008.

[27] L. Bertuccelli and J. How. Robust decision-making for uncertain Markov decision processes using sigma point sampling. *IEEE American Controls Conference*, 2008.

[28] B. Bethke and J. How. Approximate dynamic programming using Bellman residual elimination and Gaussian process regression. In *Proceedings of the American Control Conference*, St. Louis, MO, 2009.

[29] B. Bethke, J. How, and A. Ozdaglar. Approximate dynamic programming using support vector regression. In *Proceedings of the 2008 IEEE Conference on Decision and Control*, Cancun, Mexico, 2008.

[30] B. Bethke, J. How, and J. Vian. Group health management of UAV teams with applications to persistent surveillance. In *Proceedings of the American Control Conference*, 2008.

[31] B. Bethke, J. How, and J. Vian. Multi-UAV persistent surveillance with communication constraints and health management. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Chicago, IL, August 2009.

[32] B. Bethke, J. Redding, J. How, M. Vavrina, and J. Vian. Agent capability in persistent mission planning using approximate dynamic programming. In *Proceedings of the American Control Conference (submitted)*, 2010.

[33] B. Bethke, M. Valenti, and J. How. Cooperative vision based estimation and tracking using multiple UAVs. In *Proceedings of the Conference on Cooperative Control and Optimization*, Gainesville, FL, January 2007.

[34] C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[35] S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and C. Whaley. ScaLAPACK: A linear algebra library for message-passing computers. In *PPSC*. SIAM, 1997.

[36] V. Borkar. A convex analytic approach to Markov decision processes. *Probability Theory and Related Fields*, 78(4):583–602, 1988.

[37] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learing Theory*, pages 144–152, 1992.

[38] L. Bottou. *Large scale kernel machines*. MIT Press, Cambridge, MA, 2007.

[39] J. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2-3):233–246, 2002.

[40] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

[41] C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, number 2, 1998.

[42] C. Schumacher and P. R. Chandler and S. J. Rasmussen. Task allocation for wide area search munitions via iterative network flow. In *Proceedings of the 2002 AIAA Guidance, Navigation and Control Conference*, Monterrey, CA, August 2002.

[43] J. Candela and C. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

[44] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[45] H. L. Choi, L. Brunet, and J. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4), 2009.

[46] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and C. Whaley. ScaLAPACK: A portable linear algebra library

for distributed memory computers - design issues and performance. In J. Dongarra, K. Madsen, and J. Wasniewski, editors, *PARA*, volume 1041 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 1995.

[47] N. Christianini and J. Shawe-Taylor. *Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[48] L. Clarke, E. Johnson, G. Nemhauser, and Z. Zhu. The aircraft rotation problem. *Annals of Operations Research*, 69:33–46, 1997.

[49] S. Coraluppi. *Optimal Control of Markov Decision Processes for Performance and Robustness*. PhD thesis, Univ of Maryland, 1997.

[50] F. Cucker and S. Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39(1), 2002.

[51] D. Dale. Automated ground maintenance and health management for autonomous unmanned aerial vehicles. Master's thesis, MIT, June 2007.

[52] D. P. de Farias. *The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application*. PhD thesis, Stanford University, June 2002.

[53] D.P. de Farias and B. Van Roy. Approximate linear programming for average-cost dynamic programming. *Advances in Neural Information Processing Systems*, 15:1587–1594, 2003.

[54] D.P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[55] D.P. de Farias and B. Van Roy. A cost-shaping lp for Bellman error minimization with performance guarantees. *Advances in Neural Information Processing Systems 17: Proceedings Of The 2004 Conference*, 2005.

[56] R. Dearden, N. Friedman, and D. Andre. Model based bayesian exploration. In K. Laskey and H. Prade, editors, *UAI*, pages 150–159. Morgan Kaufmann, 1999.

[57] M. Deisenroth, J. Peters, and C. Rasmussen. Approximate dynamic programming with Gaussian processes. In *Proceedings of the American Control Conference*, 2008.

[58] E.V. Denardo. On linear programming in a Markov decision problem. *Management Science*, 16(5):282–288, 1970.

[59] T. Dietterich and X. Wang. Batch value function approximation via support vectors. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 1491–1498. MIT Press, 2001.

[60] C. Dixon and E. Frew. Decentralized extremum-seeking control of nonholonomic vehicles to form a communication chain. *Lecture notes in Control and Information Sciences*, 369:311, 2007.

[61] C. Dixon and E.W. Frew. Maintaining optimal communication chains in robotic sensor networks using mobility control. In *Proceedings of the 1st international conference on Robot communication and coordination*. IEEE Press Piscataway, NJ, USA, 2007.

[62] J. Dongarra, F. Gustavson, and A. Karp. Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Review*, 26:91, 1984.

[63] F. Doshi, J. Pineau, and N. Roy. Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs. *International Conference on Machine Learning*, 2008.

[64] F. Doshi and N. Roy. Efficient model learning for dialog management. *Proc of Human Robot Interaction*, 2007.

[65] Y. Engel. *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, 2005.

[66] Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In T. Fawcett and N. Mishra, editors, *ICML*, pages 154–161. AAAI Press, 2003.

[67] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In L. Raedt and S. Wrobel, editors, *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 201–208. ACM, 2005.

[68] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th*

European PVM/MPI Users' Group Meeting, pages 97–104, Budapest, Hungary, September 2004.

[69] P. Gaudiano, B. Shargel, E. Bonabeau, and B. Clough. Control of UAV swarms: What the bugs can teach us. In *Proceedings of the 2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations Aerospace Conference*, San Diego, CA, September 2003.

[70] Gentoo Foundation, Inc. Gentoo Linux. `http://www.gentoo.org/`, 2007.

[71] F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

[72] R. Gopalan and K. Talluri. The aircraft maintenance routing problem. *Operations Research*, 46(2):260–271, April 1998.

[73] G.H. Hardy. *Ramanujan: Twelve Lectures on Subjects Suggested by His Life and Work, 3rd ed.* Chelsea, New York, 1999.

[74] J. C. Hartman and J. Ban. The series-parallel replacement problem. *Robotics and Computer Integrated Manufacturing*, 18:215–221, 2002.

[75] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillan, New York, 1994.

[76] J. Higle and A. Johnson. Flight schedule planning with maintenance considerations. Submitted to OMEGA, The International Journal of Management Science, 2005.

[77] M. J. Hirsch, P. M. Pardalos, R. Murphey, and D. Grundel, editors. *Advances in Cooperative Control and Optimization. Proceedings of the 7th International Conference on Cooperative Control and Optimization*, volume 369 of *Lecture Notes in Control and Information Sciences.* Springer, Nov 2007.

[78] A. Hordijk and L. Kallenberg. Linear programming and Markov decision chains. *Management Science*, 25(4):352–362, 1979.

[79] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[80] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, April 2008.

[81] Intel Corporation. Intel Math Kernel Library. `http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm`.

[82] G. Iyengar. Robust dynamic programming. *Math. Oper. Res.*, 30(2):257–280, 2005.

[83] T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 345–352. MIT Press, 1995.

[84] R. Jaulmes, J. Pineau, and D. Precup. Active learning in partially observable Markov decision processes. *European Conference on Machine Learning (ECML)*, 2005.

[85] R. Jaulmes, J. Pineau, and D. Precup. Learning in non-stationary partially observable Markov decision processes. *ECML Workshop on Reinforcement Learning in Non-Stationary Environments*, 2005.

[86] Y. Jin, A. Minai, and M. Polycarpou. Cooperative real-time search and task allocation in UAV teams. In *Proceedings of the IEEE Conference on Decision and Control*, 2003.

[87] L. P. Kaelbing, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[88] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[89] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. `http://citeseer.ist.psu.edu/244558.html`, 1999.

[90] M. Kochenderfer. *Adaptive Modelling and Planning for Learning Intelligent Behaviour*. PhD thesis, University of Edinburgh, 2006.

[91] D. Koller and R. Parr. Policy iteration for factored MDPs. In C. Boutilier and M. Goldszmidt, editors, *UAI*, pages 326–334. Morgan Kaufmann, 2000.

[92] A. Kott. *Advanced Technology Concepts for Command and Control*. Xlibris Corporation, 2004.

[93] M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[94] N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 609–616. MIT Press, 2002.

[95] C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: a string kernel for protein classification. In *Pacific symposium on biocomputing*, 2002.

[96] C. Leslie and R. Kuang. Fast kernels for inexact string matching. In B. Schölkopf and M. Warmuth, editors, *COLT*, volume 2777 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2003.

[97] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, (2):419–444, 2002.

[98] M. Valenti, B. Bethke, J. How, D. Pucci de Farias, J. Vian. Embedding health management into mission tasking for UAV teams. In *American Controls Conference*, New York, NY, June 2007.

[99] M. Valenti, D. Dale, J. How, and J. Vian. Mission health management for 24/7 persistent surveillance operations. In *AIAA Guidance, Control and Navigation Conference*, Myrtle Beach, SC, August 2007.

[100] W. G. Macready and D. H. Wolpert. Bandit problems and the exploration/exploitatin tradeoff. *IEEE Transactions on Evolutionary Computation*, 2:2–22, 1998.

[101] M. Maggioni and S. Mahadevan. Fast direct policy evaluation using multiscale analysis of Markov diffusion processes. In W. Cohen and A. Moore, editors, *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 601–608. ACM, 2006.

[102] S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *International Conference on Machine Learning*, 2005.

[103] S. Mahadevan and M. Maggioni. Value function approximation with diffusion wavelets and Laplacian eigenfunctions. In *NIPS*, 2005.

[104] A. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, 1960.

[105] J. McGrew. Real-time maneuvering decisions for autonomous air combat. Master's thesis, MIT, June 2008.

[106] B. Michini. Modeling and adaptive control of indoor unmanned aerial vehicles. Master's thesis, MIT, August 2009.

[107] B. Michini and J. How. L1 adaptive control for indoor autonomous vehicles: Design process and flight testing. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, August 2009.

[108] G. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, January 1982.

[109] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[110] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. In *IEEE Neural Networks*, number 12(2), pages 181–201, 2001.

[111] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49(2/3):291–323, November/December 2002.

[112] R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 1:815–857, 2008.

[113] R. Murray. Recent research in cooperative control of multi-vehicle systems. *ASME Journal of Dynamic Systems, Measurement, and Control*, 2007.

[114] A. Nedic and D. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

[115] A. Nilim and L. El Ghaoui. Robust solutions to Markov decision problems with uncertain transition matrices. *Operations Research*, 53(5), 2005.

[116] Office of the Secretary of Defense. Unmanned aircraft systems roadmap. Technical report, OSD, 2005.

[117] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161–178, 2002.

[118] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pages 276–285, Sep 1997.

[119] H. Paruanak, S. Brueckner, and J. Odell. Swarming coordination of multiple UAVs for collaborative sensing. In *Proceedings of the 2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations Aerospace Conference*, San Diego, CA, September 2003.

[120] J. Peng and R. J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1:437–454, 1993.

[121] J. Platt. *Fast training of support vector machines using sequential minimal optimization in Advances in Kernel Methods — Support Vector Learning.* MIT Press, 1999.

[122] J. Platt. Using sparseness and analytic QP to speed training of support vector machines. In *Advances in Neural Information Processing Systems*, pages 557–563, 1999.

[123] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In W. Cohen and A. Moore, editors, *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 697–704. ACM, 2006.

[124] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

[125] S. Qiu and T. Lane. The RNA string kernel for siRNA efficacy prediction. In *BIBE*, pages 307–314. IEEE, 2007.

[126] S. Qiu, T. Lane, and L. Buturovic. A randomized string kernel and its application to RNA interference. In *AAAI*, pages 627–632. AAAI Press, 2007.

[127] C. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems*, 16:751–759, 2004.

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

[129] J. Redding, B. Bethke, L. Bertuccelli, and J. How. Experimental demonstration of exploration toward model learning under an adaptive MDP-based planner. In *Proceedings of the AIAA Infotech Conference*, 2009.

[130] J. Reisinger, P. Stone, and R. Miikkulainen. Online kernel selection for Bayesian reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[131] J. Rosenberger, E. Johnson, and G. Nemhauser. Rerouting aircraft for airline recovery. *Transportation Science*, 37(4):408–421, November 2003.

[132] S. Ross, B. Chaib-draa, and J. Pineau. Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *ICRA*, pages 2845–2851. IEEE, 2008.

[133] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, 2nd edition, 2003.

[134] E. Saad, J. Vian, G. Clark, and S. Bieniawski. Vehicle swarm rapid prototyping testbed. In *AIAA Infotech@Aerospace*, Seattle, WA, 2009.

[135] S. Saitoh. *Theory of Reproducing Kernels and its Applications*. Longman Scientific and Technical, Harlow, England, 1988.

[136] A. Samuel. Some studies in machine learning using the game of checkers. *IBM J. of Research and Development*, 3:210–229, 1959.

[137] E. Santos and P. Chu. Efficient and optimal parallel algorithms for Cholesky decomposition. *J. Math. Model. Algorithms*, 2(3):217–234, 2003.

[138] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In J. Shavlik, editor, *ICML*, pages 515–521. Morgan Kaufmann, 1998.

[139] B. Schölkopf. Support vector learning. `http://www.kyb.tuebingen.mpg.de/~bs`, 1997.

[140] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In *First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, 1995. AAAI Press.

[141] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

[142] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

[143] P. Schweitzer and A. Seidman. Generalized polynomial approximation in Markovian decision processes. *Journal of mathematical analysis and applications*, 110:568–582, 1985.

[144] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[145] J. Si, A. Barto, W. Powell, and D. Wunsch. *Learning and Approximate Dynamic Programming*. IEEE Press, NY, 2004.

[146] W. Smart. Explicit manifold representations for value-function approximation in reinforcement learning. In *AMAI*, 2004.

[147] A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[148] E. Solak, R. Murray-Smith, W. Leithead, D. Leith, and C. Rasmussen. Derivative observations in Gaussian process models of dynamic systems. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 1033–1040. MIT Press, 2002.

[149] M. Strens. A Bayesian framework for reinforcement learning. In P. Langley, editor, *ICML*, pages 943–950. Morgan Kaufmann, 2000.

[150] M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar. Geodesic gaussian kernels for value function approximation. In *Workshop on Information-Based Induction Sciences*, 2006.

[151] M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar. Value function approximation on non-linear manifolds for robot motor control. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2007.

[152] R. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1984.

[153] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[154] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[155] I. Szita and A. Lörincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, 2006.

[156] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[157] G. Tesauro. Temporal difference learning and TD-Gammon. *Commun. ACM*, 38(3):58–68, 1995.

[158] J. Tobias and P. Daniel. Least squares SVM for least squares TD learning. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI*, pages 499–503. IOS Press, 2006.

[159] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

[160] M. Trick and S. Zin. Spline approximations to value functions. *Macroeconomic Dynamics*, 1:255–277, January 1997.

[161] M. Valenti. *Approximate Dynamic Programming with Applications in Multi-Agent Systems*. PhD thesis, Massachusetts Institute of Technology, 2007.

[162] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[163] V. Vapnik. An overview of statistical learning theory. *Neural Networks, IEEE Transactions on*, 10(5):988–999, 1999.

[164] J. Vert. A tree kernel to analyse phylogenetic profiles. *Bioinformatics*, 18(S1):S276–284, 2002.

[165] Vicon. Vicon MX Systems. `http://www.vicon.com/products/viconmx.html`, July 2006.

[166] S. V. N. Vishwanathan and A. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Processing Systems 15*, pages 569–576. MIT Press, 2003.

[167] G. Wahba. Spline models for observational data. In *Proceedings of CBMS-NSF Regional Conference Series in Applied Mathematics*, SIAM, Philidelphia, 1990.

[168] M. A. Wiering. *Explorations in Efficient Reinforcement Learning.* PhD thesis, University of Amsterdam / IDSIA, February 1999.

[169] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *NIPS*, pages 682–688. MIT Press, 2000.

[170] D. Wingate and K. Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research*, 6:851–881, 2005.

[171] J. Xiong Dong, A. Krzyzak, and C. Suen. Fast SVM training algorithm with decomposition on very large data sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(4):603–618, 2005.