Approximate Dynamic Programming with Applications in Multi-Agent Systems

by

Mario J. Valenti

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2007

Author .....
Department of Electrical Engineering and Computer Science
May 4, 2007

Certified by .....
Daniela Pucci de Farias
Assistant Professor of Mechanical Engineering
Thesis Supervisor

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

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Approximate Dynamic Programming with Applications in Multi-Agent Systems

by

Mario J. Valenti

Submitted to the Department of Electrical Engineering and Computer Science
on May 4, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering

Abstract

This thesis presents the development and implementation of approximate dynamic programming methods used to manage multi-agent systems. The purpose of this thesis is to develop an architectural framework and theoretical methods that enable an autonomous mission system to manage real-time multi-agent operations. To meet this goal, we begin by discussing aspects of the real-time multi-agent mission problem. Next, we formulate this problem as a Markov Decision Process (MDP) and present a system architecture designed to improve mission-level functional reliability through system self-awareness and adaptive mission planning. Since most multi-agent mission problems are computationally difficult to solve in real-time, approximation techniques are needed to find policies for these large-scale problems. Thus, we have developed theoretical methods used to find feasible solutions to large-scale optimization problems. More specifically, we investigate methods designed to automatically generate an approximation to the cost-to-go function using basis functions for a given MDP. Next, these these techniques are used by an autonomous mission system to manage multi-agent mission scenarios. Simulation results using these methods are provided for a large-scale mission problem. In addition, this thesis presents the implementation of techniques used to manage autonomous unmanned aerial vehicles (UAVs) performing persistent surveillance operations. We present an indoor multi-vehicle testbed called RAVEN (Real-time indoor Autonomous Vehicle test ENvironment) that was developed to study long-duration missions in a controlled environment. The RAVEN's design allows researchers to focus on high-level tasks by autonomously managing the platform's realistic air and ground vehicles during multi-vehicle operations, thus promoting the rapid prototyping of UAV technologies by flight testing new vehicle configurations and algorithms without redesigning vehicle hardware. Finally, using the RAVEN, we present flight test results from autonomous, extended mission tests using the technologies developed in this thesis. Flight results from a 24 hr, fully-autonomous air vehicle flight-recharge test and an autonomous, multi-vehicle extended mission test using small, electric-powered air vehicles are provided.

Thesis Supervisor: Daniela Pucci de Farias
Title: Assistant Professor of Mechanical Engineering

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

Acknowledgments

First and foremost, I would like to express gratitude to my thesis supervisors. First, to Prof. Daniela Pucci de Farias for her assistance, guidance, enthusiasm and patience with my research work.

Second, to Prof. Jonathan How for his guidance, passion, and positive attitude toward this research. I appreciated the flexibility he provided me in managing this research project and I enjoyed the cooperative research environment.

Third, to my committee members Prof. Dimitri Bertsekas, Prof. Daniela Rus, and Prof. George Verghese for their valuable insight, comments and suggestions.

Fourth, to Dr. John Vian and The Boeing Company for funding this research under the "Vehicle and Vehicle System Health Management" contract.

Fifth, to the professors at Temple University and MIT that assisted me during my academic studies. Specifically, to Prof. Brian Butz, Prof. Thomas Ward, Prof. Robert Yantorno, Prof. John Essignmann and Prof. Eric Feron for their advice during my collegiate studies.

Sixth, to all of my colleagues and friends at MIT. Specifically, to Brett Bethke, Keith and Liz Bonawitz, Daniel Dale, and Adrian Frank for fostering a great (and fun) research environment.

Seventh, to the students and staff of Simmons Hall and members of the Eastgate Bible Study for their support and friendship.

Eighth, to my family, especially my brother Phillip, who has faithfully served our country during my stay at MIT, my brother Justin, who always has creative ideas and a way with words, and my mother, who has comforting thoughts in the best and worst of times. And, most importantly, to my father, the voice of reason in challenging situations and a great role model as a father, a leader and an engineer. He is one of the best engineers I have ever met.

Ninth, to my son Ian for coming to the lab and always brightening up the day for all of us. I hope his passion for exploring and experiencing the world around him is as strong today as it was when I wrote this.

Finally, I would like to express my deepest gratitude and thanks to my Lord and savior Jesus Christ, by whom all things are possible. *Phil. 3:12-14*

Dedicated to my loving wife Tricia ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned vehicles have been in use for many years. For example, some of the first unmanned (or pilotless) aircraft were developed after World War I to be used as "aerial torpedoes" (which are now called cruise missiles) [35]. This work later led to the development of the remotely-piloted target "drones" to train anti-aircraft gunnery operators in Great Britain and the United States in the 1930s [35]. Today, unmanned aerial vehicles (UAVs) are becoming vital warfare and homeland security platforms because they significantly reduce costs and the risk to human life while amplifying warfighter and first-responder capabilities. These vehicles have been used in Iraq and during Hurricane Katrina rescue efforts with success. In fact, the U. S. Department of Defense has increased UAV development funding from about $3 billion in the 1990s to over $12 billion for 2004 through 2009 as they continue to find new roles and missions for UAVs in combat and surveillance operations [72]. However, achieving the vision of multiple UAVs operating cooperatively with other manned and unmanned vehicles in the national airspace and beyond remains a formidable barrier. Indeed, many unmanned vehicles do not exhibit the level of performance and flexibility needed to complete an entire mission autonomously. For example, most UAV guidance and mission planning systems do not possess the capability to recognize and react to unexpected changes in the operating conditions. The complexity of this problem increases as multiple agents are introduced. For example, if another autonomous agent is added to the mission scenario, then both vehicles must resolve information regarding the impending actions of the other vehicle. Similarly, if a manned agent is added to the original scenario, the autonomous vehicle must also possess the capability to effectively communicate and coordinate its actions with the manned vehicle.

Despite the aforementioned challenges, these multi-agent teams can provide valu-

able information that can be used to make mission critical decisions in real-time. Multi-agent teams offer a promising alternative to a number of high-risk manned mission scenarios. These vehicle groups can be used to perform round-the-clock missions in dangerous and unknown environments without considerable risk to the human operators. However, operational costs remain a key concern for mission planners. For example, although the procurement costs for UAVs are much lower than that of many manned aircraft, UAVs – thought at one time to be low-cost tactical solutions for high-risk situations – have become increasingly expensive. In 2003, the estimated procurement costs for 22 MQ-1 Predators was $154.1 million [71]. As a result, UAVs are no longer thought of as "disposable" aircraft because of their cost [93].

Numerous researchers are investigating future systems that use autonomous agents to cooperatively execute these missions [16, 31, 73, 74]. However, little has been said to date about how to perform multi-day autonomous system operations. Autonomous mission systems must balance issues related to vehicle capability, reliability, and robustness with task and mission goals when creating an effective strategy. In addition, these systems have the added responsibility of interacting with numerous human operators while managing both high-level mission goals and agents conducting individual tasks. As a result, multi-agent health management techniques are being developed to increase the reliability of autonomous systems during mission operation.

In prior work, the term *health management* is used to define systems that actively monitor and manage vehicle components (for example actuators, flight control, engine, avionics and fuel management hardware) in the event of failures [30]. Prognostic and health management techniques are being developed for new military aircraft systems to reduce future operating, maintenance, and repair costs [5]. In the context of multiple vehicle operations, this definition can be extended to autonomous multi-agent teams. In this case, teams involved in a mission serve as a vehicle system. Each multi-agent team involved in the mission is a subsystem of the larger mission team, and each vehicle is a subsystem of each multi-agent team.

As with mission-critical systems for a single agent, multi-agent task allocation and mission management systems must account for vehicle- and system-level health-related issues to ensure that these systems are cost effective to operate. For example, despite the on-going development of health management techniques for flight-critical systems, most UAVs are controlled by a team of operators from a remote location [78]. Here, we define an operator as a human being that monitors or issues commands to a UAV during a mission. Although recent advances in mission systems have reduced the number of operators per vehicle, the vehicle to operator ratio for most UAV mission

platforms remain less than or equal to one. This ratio is small for many reasons. First, most UAVs are piloted by humans during take-off, landing and other complex flight tasks [29, 32, 33]. This means that for every UAV, there is at least one UAV "pilot" operator. In addition, most UAVs have remote ground stations to monitor their flight critical systems (e.g., communications link, flight control, guidance/navigation systems) and mission data. Since it is difficult for a "pilot" operator to monitor all mission critical information, most UAVs have more than one operator during a mission [68]. Therefore, in a rapidly changing environment an operator may find themselves "overloaded" with information from multiple task teams.

As a result, there are major problems – which are not solely vehicle allocation issues – that need to be addressed in the current autonomous multi-agent problem. For example, questions related to decision making and the division of work between man and machine are not well-defined. This problem leaves us with the following question:

> *During a mission, how should teams of autonomous agents be managed to meet scenario objectives while minimizing the total cost of the operation?*

This question directly relates to the health of the mission system. Since each multi-agent team may perform mission tasks over an extended period of time, issues relating to task coordination, maintenance, operator support, and asset (both operator and vehicle) shift changes will arise. In fact, in citing recent reports on UAV activities, [45] notes that "... system reliability may be emerging as a greater threat to UAVs than it currently is to conventional aircraft. This trend may serve to increase the criticality of maintenance...." in UAV systems [45]. As a result, a mission system and its operators must collaboratively manage the health of several multi-agent teams in order to meet mission requirements. This problem may be formulated as a very large mathematical programming problem; however, this approach is likely to be computationally intractable for any computer or human operator to solve in real-time (even for a problem of reasonable size).

1.1 Literature Review

In principle, many of these questions are very similar to questions arising in manufacturing. For example, the problem of scheduling machine maintenance in between production runs is a common scheduling problem found in the manufacturing world. A simple machine repair example problem can be found in [7] where using Dynamic

Programming an optimal solution could be found. In [42], the authors examine the multiple machine replacement problem for parallel and serial production lines. Each problem is formulated using an integer programming approach to yield a policy for how each asset will be utilized over a finite horizon.

Likewise, scheduling and maintenance problems have been explored with respect to air transportation [3, 4, 10, 13, 18, 36, 43, 80]. For example, in [10] the authors use an integer programming formulation of the problem to address the deterministic, multi-airport Air Traffic Flow Management Problem with route capacity constraints. The solution to this problem provides the departure times and speed adjustments of each aircraft flying between a network of airports in capacitated airspace to ensure that the vehicles arrive and depart from each destination in a timely fashion, thus reducing the operating costs of the overall system. In addition, they show that the Traffic Flow Management Problem (TFMP) with capacity equal to one is an NP-Hard problem. They also show that their problem formulation can handle variations on the TFMP such as dependencies between arrival and departure times, groups of multiple aircraft from a carrier arriving and departing from a location, and aircraft re-routing. Papers, such as [3], have used a similar method to generate sequences of arrival times for incoming aircraft to reduce operating costs. Other papers, such as [13, 80], have also addressed airline scheduling problems to reduce delay time and operating costs due to external disruptions in flight plans (i.e., weather conditions, mechanical failures).

On the other hand, [4, 18, 36, 43] incorporate maintenance considerations into the vehicle routing problem to lower routing and operating costs while meeting scheduled maintenance constraints. For example, in [18] the authors use asymmetric travelling salesman model with side constraints and solve the problem using Lagrangian relaxation techniques and subgradient optimization. In [36], the authors start with sets of Lines of Flights (LOFs) which aircraft fly regularly and form a graph that is adjusted to meet three-day maintenance routing constraints.

Though many of the issues presented in these papers apply to problems related to scheduling concerns, some of the challenges specific to persistent operations include (but are not limited to): several multi-agent teams may be operating simultaneously that may or may not coordinate tasks and information about the current task, vehicle assets may be lost during the course of a mission, and little or no information about the vehicles may be directly available to the operator during the mission. For example, a vehicle failure may become known only after the vehicle has failed to show up for refuelling past a given deadline.

A number or researchers have explored persistent UAV operations [28, 29, 84]. For example, in [29], researchers demonstrated video surveillance over an area using two fixed-wing UAVs, while [84] describes techniques for using multiple UAVs to explore regions for ground objects of interest. Similar missions have been studied as part of the DARPA Sponsored MICA (Mixed Initiative Control of Automa-Teams) project which focused on the use of multiple vehicle teams to accomplish discrete mission tasks [6, 51, 105]. These papers focus on asset allocation and mission management during flight operations. More recently, researchers are beginning to examine fuel constraints in UAV tasking operations [14, 15, 86]. However, in each case these papers do not address the fuel / health monitoring problem explicitly as part of the tasking problem (in terms of flight time limitations, etc). In [86], the authors assume that the vehicles can reach all of the targets and / or complete the tasks given to the vehicles as specified. In [14, 15], although the authors say that their decentralized multiple-UAV approach for monitoring the perimeter of a forest fires includes a feature to "...systematically add and remove UAVs from the team (important for refueling)" the authors later say in the conclusion of [14] that "... numerous technical issues remain to be resolved including determination of the initial rendezvous time, dealing with fuel contingencies and refueling, implementation with irregular and growing fire shapes, and determining factors that allow the perimeter length to be updated frequently enough." Refs. [98, 99] specifically include health and fuel models in the mission and tasking architecture. These models were used to monitor vehicle capabilities during flight tests with real hardware in real-time. These models are described in detail as part of this thesis work.

Once the problem is defined, the next step is formulating the problem in a matter by which it can be solved. One of the major obstacles faced when an approximate dynamic programming method is used to solve a large-scale problem is the calculation or generation of the cost-to-go structure for the problem. For example, using an Approximate Linear Programming (ALP) formulation of the original problem requires the selection of appropriate basis vectors and state relevance weights. In many cases, selecting the appropriate parameters used to find basis functions is based on experience.

A variety of methods for value function approximation have been proposed in the literature and are summarized in [9]. The history of parametric function approximations dates back to Samuel in 1959 when he used a parametric approximation to allow a computer to automatically generate policies for playing checkers [65, 81]. Currently, there are some papers in the literature on procedures for selecting param-

eters for basis functions, however there is a need for more research in this area. First, Refs. [75, 76] propose a scheme for generating a set of basis functions from an initial basis, its current domain, and weight vector. In this formulation, they use the dual problem to determine the best candidate for the new basis function. Similarly, [60] generates future basis functions by optimizing a class of parametric basis function models. [61] proposes a scheme called "Least Square Policy Iteration" that uses a linear architecture where the basis functions are given by $\phi(x) = [1\ s\ ...\ s^K]$ where s represents the state number. [95] proposes a scheme that approximates the value functions using low-dimensional cubic-spline. [34, 87] model the state space using a manifolds representation based on reachable sets used to partition the state space in to regions (which, as the author describes, is much like partitioning an "atlas" in to overlapping "charts").

At the University of Massachusetts-Amherst, Mahadevan *et al.* are developing model-free methods to generate basis functions from the underlying problem formulation. In [65, 66], the authors propose a method to build proto-value functions by developing basis functions using the underlying state space geometry using the low-order eigenfunctions of the graph Laplacian. Refs. [64, 67] propose using a methods based on diffusion wavelets to generate basis functions. Refs. [106, 107] propose multi-grid methods to approximate the cost-to-go function for an MDP, where the basis functions are calculated recursively using a set of inter-level operators that are computed prior to starting the optimization for different resolution. However, the authors noted in their conclusion that finding a method for computing the inter-level operators remained an issue for the algorithm's application.

Ref. [69] proposes a method that adapts the (non-linear) basis function parameters, while optimizing the basis function weights. This paper assumes that these basis functions have some pre-determined parametric form (in this case, radial basis functions) and uses a gradient-based approach and a cross entropy method to perform the adaptation, while using a weighted 2-norm scoring function of the approximate Bellman error as an optimization criterion. Finally, [55] proposes a method that uses neighborhood component analysis (NCA) to select new features to be added to the feature matrix. In this research sampled trajectories are used to generate the approximate Bellman error for each state. Using neighborhood component analysis, a transformation is learned that maps states with similar Bellman errors together, which is then used to select features for the new basis functions to be used in the next iteration.

Although many of these techniques have promising results, many of these algo-

rithms cannot be employed efficiently for large-scale problems. The main difference between the research presented in this thesis and previous work is that the basis function generation algorithm presented here is designed to be implemented for large-scale problems. In addition, the methods presented in this thesis use a simple closed-form approximation structure for computing basis functions *implicitly* using sampled trajectories. Also, to the best of our knowledge, this basis function generation method is the only method in the literature that has been used to manage an hardware-based autonomous system in real-time.

For testing these research ideas, a variety of test platforms have been developed to study advanced theories and approaches in the development of innovative multi-vehicle and UAV concepts [2, 19, 26, 27, 33, 46, 47, 49, 52, 50, 53, 54, 56, 58, 70, 85, 90, 103, 104]. For example, the BErkeley AeRobot (BEAR) project features a fleet of commercially available rotary-wing and fixed-wing UAVs that have been retrofitted with special electronics. These vehicles have been used in applications such as autonomous exploration in unknown urban environments and probabilistic pursuit-evasion games [85, 103]. In the Aerospace Controls Laboratory at MIT, an outdoor testbed consisting of a fleet of eight fixed-wing autonomous unmanned UAVs was designed as a platform for evaluating coordination and control algorithms [49, 56]. Similarly, researchers in the Multiple AGent Intelligent Coordination and Control (MAGICC) Lab at Brigham Young University have built and flown a group of small fixed-wing UAVs to perform multi-vehicle experiments outdoors [70]. These planes are launched by hand and track waypoints autonomously.

Likewise, the DragonFly project at Stanford University's Hybrid Systems Laboratory uses a heavily-modified fixed-wing model aircraft with a 10-foot wingspan for experiments, and two additional aircraft are under development [26, 90]. The objective of this platform is to provide an inexpensive capability for conducting UAV experimental research, ranging from low-level flight control to high-level multiple aircraft coordination. Similarly, to demonstrate new concepts in multi-agent control on a real world platform, the Hybrid Systems Lab developed the Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC). STARMAC is a multi-vehicle testbed consisting of two quadrotor UAVs that autonomously track a given waypoint trajectory [46]. Quadrotors are used in this platform based on their convenient handling characteristics, low cost, and simplicity.

In addition, indoor multi-vehicle testbeds have been constructed to study multi-agent activities. For example, the HOvercraft Testbed for DEcentralized Control (HOTDEC) Platform at the University of Illinois Urbana-Champagne is a ground

vehicle testbed used for multi-vehicle control and networking research [104]. The ground-based vehicles in this test platform have a simple battery monitor built into the vehicles to denote when they are low on power, require recharging, and can dock with a power station to fully recharge the batteries in 2.5 hours. Researchers at Vanderbilt University have built the Vanderbilt Embedded Computing Platform for Autonomous Vehicles, which has been used to fly two vehicles in an indoor environment [58]. Also, the UltraSwarm Project at the University of Essex is designed to use indoor aerial vehicles to examine questions related to flocking and wireless cluster computing [47]. Likewise, researchers that Oklahoma State University use a ground vehicle testbed called COMET (COoperative MultivehiclE Testbed) to implement and test cooperative control techniques both indoors and outdoors [19].

As mentioned earlier, most of these testbeds do not focus on vehicle health and maintenance monitoring in regard to UAV technologies. Past research into battery monitoring and state estimation has focused on using direct, invasive measurements of current flow and voltage level to calculate a battery's state of charge (SOC). Most of this research focuses on calculating SOC using complex analytical models of internal battery dynamics [77, 79]. However, these approaches require significant knowledge of battery properties and internal dynamics. Some recent research has sought to simplify the construction of battery SOC models by using machine learning techniques using voltage and current measurements from the battery [41]. A learning approach can be advantageous because it does not require knowledge of internal battery chemistry and can be easily extended to multiple battery chemistries.

Even as electric-powered autonomous vehicles and their support systems become smarter, they are fundamentally limited by the capacity of the batteries that power these vehicles. As described in the section above, autonomous health management hardware and software allow vehicles to determine the battery's current status and decide when a vehicle must land to replace or recharge itself before continuing its mission. Ground platforms have been developed to allow robots to recharge during operations. For example, at the University of Tsukuba, researchers constructed an autonomous ground vehicle and recharge system in order to facilitate autonomous ground vehicle navigation and control experiments [40]. The system was tested by running an autonomous vehicle nonstop for one week. During the week-long experiment, over one thousand recharge dockings were successfully accomplished. However, to the best of our knowledge, Ref. [97] reported the first instance (in the literature) of an autonomous docking and recharge using an electric-powered air vehicle.

1.2 Thesis Outline

The main goal of this thesis is to develop, implement, and test methodologies that can be used in real-world applications to manage autonomous multi-agent systems in extended mission operations. To meet this goal, this thesis focuses on two primary objectives:

- Explore and develop methods to find feasible solutions for large-scale optimization problems relating to multi-agent tasking problems
- Design and implement a system architecture which incorporates these methods to manage multi-agent teams in a real-time environment

To meet these objectives, this thesis begins by presenting the real-time, multi-agent mission problem. As part of this problem, different aspects (e.g., mission lengths, agent reliability, communication and control issues, computational concerns) of this problem are discussed to understand how each component plays a role in the performance of the mission system. Next, this thesis presents a mission system architecture used to manage long-duration autonomous missions. A key component of this architecture (which distinguishes it from previous mission system architectures) is the inclusion of system and component health information.

Following this discussion on the mission system, we focus on a method that allows an automated vehicle tasking system to formulate and solve an multi-agent tasking problem with health management considerations in real-time. This method is designed to automatically generate an approximate cost structure that can be used to find policies for a given Markov Decision Process (MDP) automatically. Using this method, an autonomous system can automatically compute a set of basis functions for a given MDP, so that the problem can be formulated and solved in real-time. This is a fundamental question related to use of approximate dynamic programming in real-time systems.

Next, we return our attention to the real-time, multi-agent mission problem. Using the basis function generation algorithm formulated in the previous chapter, we address the multi-agent mission management problem for long-duration missions. This discussion focuses on the practical implementation of this technique for the centralized multi-agent mission problem using a real-time simulation.

Next, to fully investigate questions related to the implementation of such algorithms in real-time, an indoor multi-vehicle testbed was created to study long-duration mission and to develop health management systems for autonomous multi-

agent mission platforms. A description of this indoor multi-vehicle testbed called RAVEN (Real-time indoor Autonomous Vehicle test ENvironment) is provided. Normally, demonstrations of multi-vehicle coordination and control technologies require that multiple human operators simultaneously manage flight hardware, navigation, control, and vehicle tasking. However, RAVEN simplifies all of these issues. Since the system autonomously manages the navigation, control, and tasking of realistic air vehicles during multi-vehicle operations, researchers can focus on high-level tasks. This characteristic promotes the *rapid prototyping* of UAV technologies by means of flight testing new vehicle configurations and algorithms without redesigning vehicle hardware. This discussion includes a description of the components and architecture of RAVEN and presents recent flight test results. In addition, this section discusses how a mission manager can be integrated with other tasking systems to effectively manage long-duration, multi-vehicle operations.

Finally, using RAVEN we present the development and implementation of all of the above techniques to manage autonomous unmanned aerial vehicles (UAVs) performing persistent surveillance operations. This section presents mission health monitors aimed at identifying and improving mission system performance to avoid down time, increase mission system efficiency and reduce operator loading. In addition, we discuss the additional hardware infrastructure needed to execute an autonomous persistent surveillance operation. Finally, we present results from a fully-autonomous, extended mission test using the technology and monitors developed in this thesis to improve system performance in real-time.

1.3 Thesis Contributions

Through this work, this thesis makes contributions in the four major research areas:

- First, this thesis presents a method designed to generate a set of basis functions used to form the approximate cost-to-go function. This method enables an autonomous system to take an Markov Decision Process (MDP), formulate it as an approximate linear program (ALP), and solve it in real-time automatically. Our approach uses sampled trajectories generated via simulation to estimate future cost-to-go can improve cost-to-go estimates and provide results in policies. In addition, by using a trajectory-based approximation method in calculating the future cost-to-go for each state, we have developed a simple closed-form approximation structure for computing basis functions *implicitly* using sampled trajectories by storing only multipliers r_0, r_1, ... r_N. Therefore, our approach

does not require that large amounts of information be saved to generate the basis functions. As a result, this basis function generation method is designed to allow users to distribute computations over networked resources, thereby resulting is a viable algorithm for use in real-time. A numerical complexity result, a proof showing the convergence of the algorithm without trajectory sampling, and an error bound comparing the optimal solution to the approximate solution using the basis function generation algorithm for a single update using sampled trajectories are given.

- Second, this thesis presents a formulation of the mission management problem that accounts for vehicle failures and system health concerns. In this problem, mission assets are commanded by to carry out tasks (e.g., search an area, classify objects, locate and attack targets) issued via from sophisticated operator stations miles from their areas of operations. The problem formulation developed in this thesis enables an autonomous system to implement and solve this multi-agent mission planning problem in real-time using real hardware. Since this problem can be formulated as a MDP with many states, we demonstrate how policies can be generated for this problem using approximate dynamic programming methods. More specifically, by using the basis function generation algorithm to generate the cost-to-go function in real-time, we demonstrate how this mission management problem formulation allows an autonomous system to make adjustments on-the-fly based on system health feedback information and manage mission operations in real-time.

- Third, this thesis presents the architecture and setup of an indoor multi-vehicle testbed called the RAVEN (Real-time indoor Autonomous Vehicle test ENvironment) to study long-duration missions in a controlled environment. The RAVEN is designed to allow researchers to *rapidly prototype* UAV technologies by means of flight testing new vehicle configurations and algorithms without redesigning vehicle hardware. The RAVEN's mission system autonomously manages the navigation, control, and tasking of realistic air vehicles during multi-vehicle operations, researchers can focus on high-level tasks. In addition, RAVEN enables users to examine and develop health management techniques to improve the mission system's operational capabilities. For example, by adjusting parameters in the testbed's setup, the RAVEN allows researchers to examine a mission planning algorithm's response to a variety of failure cases, environmental conditions, and other real-world scenarios. Using information

from these tests, researchers have been able to create and improve techniques that promote proactive mission planning strategies (while reducing operational costs) using system health feedback. Through this research, the RAVEN is being used to enable many first-time tests for UAV and other unmanned system technologies implemented in real-time.

- Finally, this thesis presents the development and implementation of techniques used to manage autonomous unmanned aerial vehicles (UAVs) performing persistent surveillance operations. Although few papers have suggested the means by which such a test could be performed, this thesis is the first to develop and implement the technologies necessary to achieve this capability. These results represent a large step in the development and implementation of autonomous UAV mission technologies for long-duration missions. These technologies include, but are not limited to, battery monitoring of electric vehicles, automatic recharging of UAVs, precision landing and takeoff, mission- and vehicle-level health monitoring, and an integrated communication protocol allowing all components to communicate and make decisions cooperatively.

As a result, this thesis presents the development, implementation, and testing of methodologies that can be used in real-world applications to manage teams of autonomous vehicle systems in extended mission operations.

Chapter 2

The Real-Time Multi-Agent Mission System Problem

To begin this thesis discussion, we will start by presenting the issues surrounding the real-time multi-agent mission problem. In real-life, mission activities can be defined in many ways. For example, an urban combat mission may involve both ground and air facilities; however, the needs and activities of these resources in a surveillance mission may greatly differ from that of a rescue activity. Therefore, the requirements from one mission may vastly differ from another. As shown in Figure 2-1, a single mission system may be required to coordinate the actions of multi-agent mission teams in a variety of scenarios. Note that these scenarios and their requirements will change over time.

Designing a flexible, yet robust architecture framework is a large obstacle for autonomous mission system designers. Since different aspects (e.g., mission lengths, agent reliability, communication and control issues, computational concerns) of the multi-agent mission problem pose different demands on the system, it is difficult to find a unifying framework that works for every mission scenario. However, there are commonalities in all mission types. In fact, a simplified description for a wide variety of mission scenarios can be made as follows:

M Mission Assets

A mission system has a set of mission assets (for example vehicles, groups of agents, swarm teams, etc.) that can be allocated to a variety of mission tasks. These assets can be represented by a unique description (location, orientation, velocity, capability, etc.) that identifies it from its counterparts. These mission

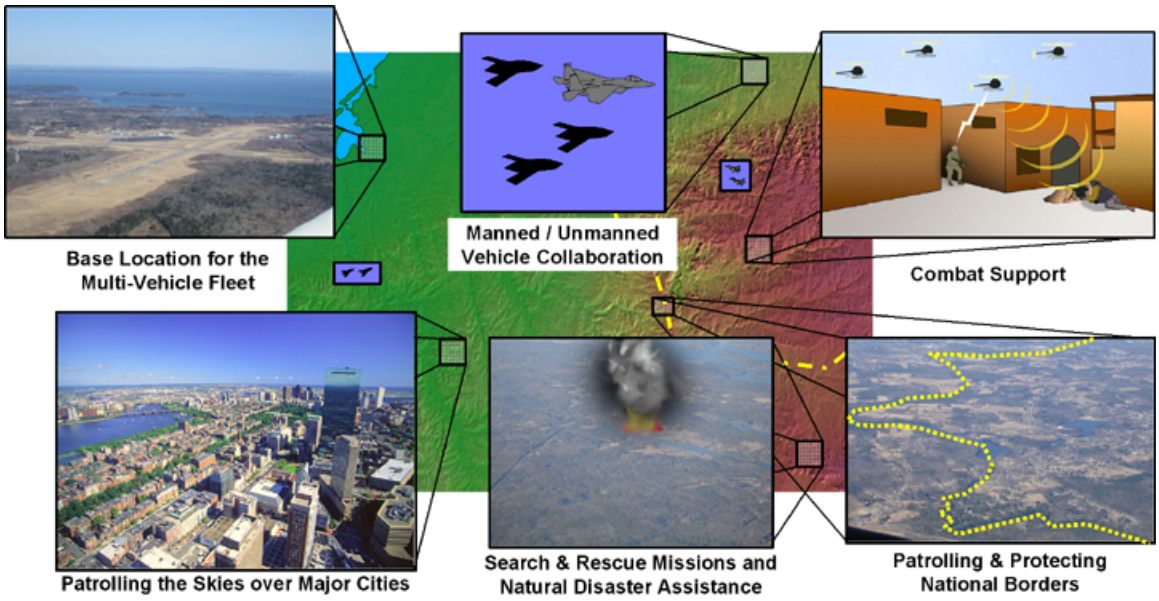


Figure 2-1: Multi-Agent Mission System Problem Scenario

assets each may have capabilities (tools, weaponry, sensors, skills, etc.) that can be used in a variety of mission scenarios. In addition, as each vehicle is used, the condition and capability of each asset can change over time.

N Designated Mission Sites with K Possible Scenarios

A mission system will also be provided with a series of mission sites and scenarios with different requirements. Note that most mission tasks are specific to their location, duration, and mission success criteria. In addition, many scenarios have task-specific requirements (e.g., vehicle coordination, persistent presence of mission assets, time constraints) and relevant information (that may change over time) that impact and govern the approach mission assets must use when approaching the problem.

Here, the main objective of the mission system is to coordinate the actions of its mission assets to achieve the system's mission goals while meeting task requirements. Note that even though this description abstracts away most of the mission-specific details, this problem remains difficult to solve in a real-time framework as a single optimization for large M, N, and K.

However, using this simplified description as a guide, this problem can be broken down into a series of steps that help to reduce (but not completely eliminate) the complexity of this problem. In trying to decide what mission assets are allocated to each mission site and scenario, there are three fundamental questions that arise as part of this allocation process:

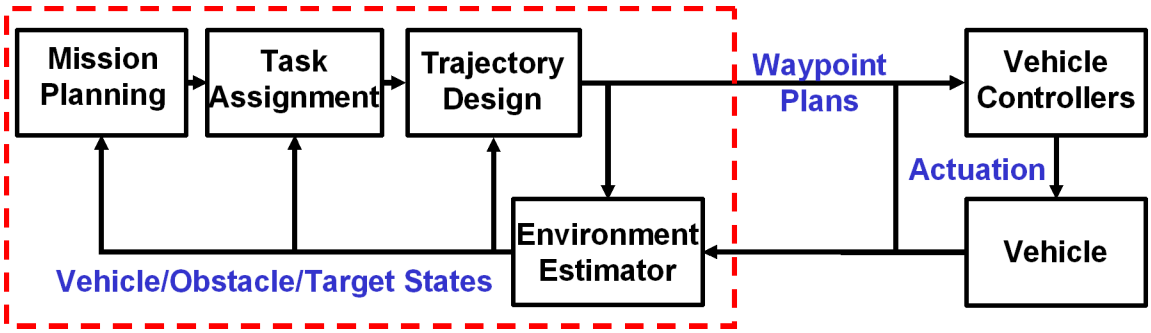


Figure 2-2: Multi-Agent Mission System Architecture using Hierarchical Decomposition

1. How many assets should be assigned to a mission scenario?

2. What tasks will each member of a mission group perform? (Here, a mission group is defined as a conglomerate of mission assets assigned to a mission task)

3. How will each mission asset carry out the activities it has been assigned?

Note that even if we assume that no mission asset can be a member of two mission groups, these three questions invoke the following actions:

1. Based on the number of incoming mission scenarios, the system must decide how many assets need to be assigned to each mission task. For the purpose of this thesis, this problem will be defined as the multi-agent mission management problem.

2. For each mission scenario, the system must decide how to assign tasks to each mission asset and decide how and when each task request must be performed. For this purpose of the thesis, this problem will be defined as the multi-agent task assignment problem.

3. As task requests are issued, each mission asset must formulate and implement a strategy that will ensure the completion of each task, while meeting task requirements. For the purpose of this thesis, these problems will be defined as the task scheduling and trajectory planning problems.

Using this multi-tiered decomposition, each aspect of the multi-agent mission problem can be addressed using different computational resources, which leads to a computationally tractable problem formulation that can be implemented in real-time. Figure 2-2 shows a representation of a multi-agent system architecture that uses

this decomposition to address the multi-agent mission problem for real-time mission scenarios. Here, each mission asset is a vehicle that performs a specified operation, as defined by a task request. Similar architectures have been proposed and implemented in the past with success [56, 82, 100].

2.1 Health Management in Real-Time Mission Systems

In using the hierarchical architecture as shown in Figure 2-2, a number of implementation concerns arise that can reduce performance and possibly jeopardize the success of a mission in a real-time environment. Some of these real-time challenges include:

- Communication Issues (Bandwidth and range constraints, dropped packets, incomplete messages)
- Computational Concerns (Memory and resource allocation, distributed processing, "hard" deadlines)
- Control Issues (Centralized/decentralized, multi-agent operations, autonomous capabilities)
- Human/Operator Interaction
- Safety Concerns (SWARM and individual vehicle)
- Software Integration Issues
- System Health Concerns (failures, loss of system assets, low mission efficiency)

In addition to these issues, external and environmental conditions will affect system performance. In most real-life situations these conditions are not controlled and the operational environment is only partially known. Therefore, as explained in [100], real-time implementations of mission system technology must be designed to be robust to external and unexpected conditions. For example, a vehicle's trajectory planning system must plan paths that are robust to environmental conditions (i.e., pop-up obstacles, no-fly zones, changes to the mission plan, wind conditions). Likewise, if information between two mission system components is not received, either system should re-transmit its data. Without feedback on vehicle and system component performance, many times architecture components become reactionary to problems

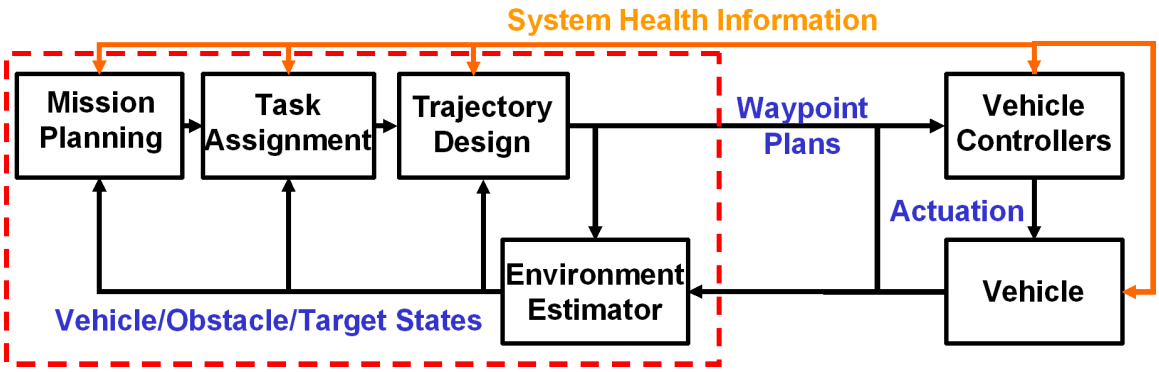


Figure 2-3: Multi-Agent Mission System Architecture using Hierarchical Decomposition with System Health Management Information

that occur in a mission system, which in many cases causes a reduction in system performance. However, most systems do not possess health monitoring components to evaluate subsystem performance.

For this reason, health management feedback and monitoring in a mission system is essential in maintaining a proactive approach to mission planning. Adding system health monitors and feedback to a simple hierarchical design (as shown in Figure 2-3) can improve a system's run-time capabilities by ensuring that it maintains a basic, functional capability during a mission in accordance with system goals.

As shown in Figure 2-4, health management systems are tailored to aid each level in the architecture. At the vehicle level, many aspects related to a vehicle's current performance and capabilities can be considered as part of this state. For example, battery charge, sensor capabilities, and motor/engine wear play a large role in an electric vehicle's capability to perform a mission. Since vehicle-level failures may impact high-level decision making and performance, mission- and task-level systems must also incorporate vehicle and task group health information into their decision making processes to ensure that these systems are cost effective to operate over the duration of a mission. Thus, this health feedback loop ensures that high-level systems in the architecture account for vehicle health management issues (e.g., refuelling, vehicle failures) in mission planning.

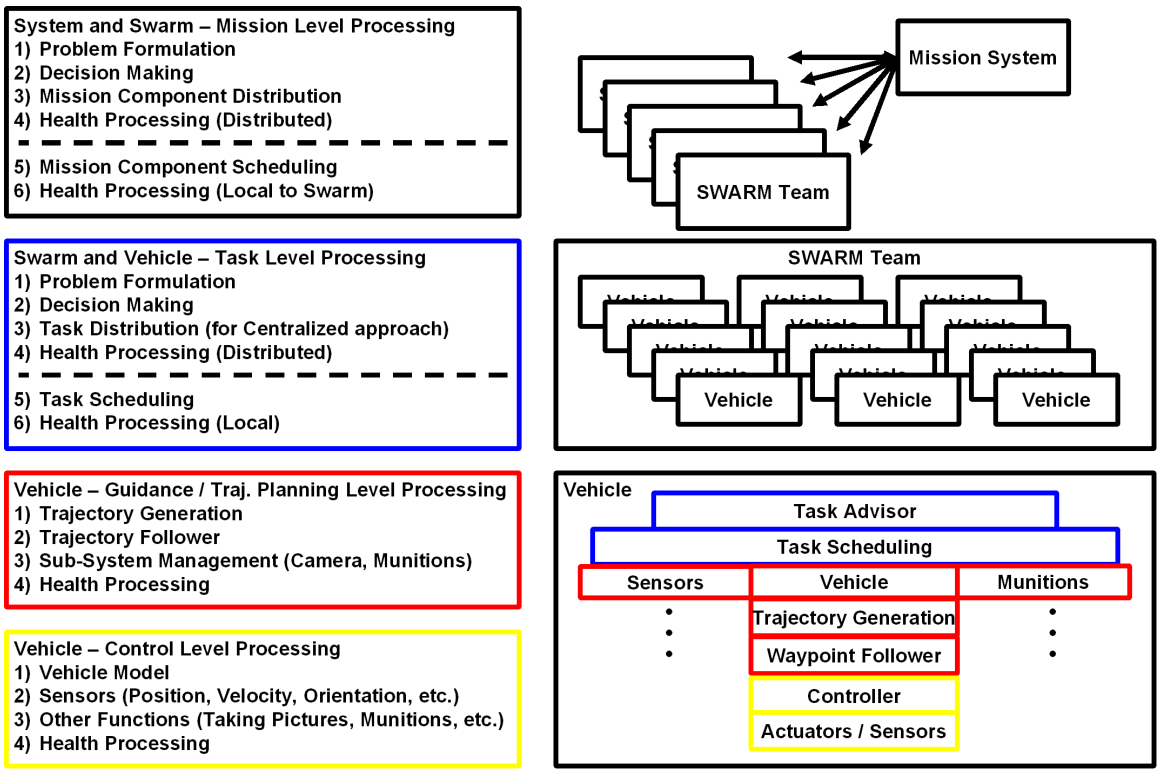


Figure 2-4: Multi-Agent Mission System Architecture

2.2 The UAV SWARM Mission Management Problem

Using these architecture ideas, we can now focus on the real-time UAV SWARM Mission Management Problem. In this problem, UAVs are commanded to perform tasks from sophisticated operator stations miles from their areas of operations. Since each multi-agent team may perform mission tasks over an extended period of time, issues relating to task coordination, maintenance, and asset shift changes will arise. As such, an autonomous mission planning system must manage the health of several multi-agent teams to meet mission requirements. Once again, this problem may be formulated as a very large mathematical programming problem; however, this approach is likely to be computationally intractable for any computer or human operator to solve in real-time (even for a problem of reasonable size).

In principle, though similar questions arise in manufacturing [7, 42] and air transportation [3, 4, 10, 13, 18, 36, 43, 80] systems, some of the challenges specific to the multi-agent tasking problem include several multi-agent teams are operating simultaneously (which may or may not coordinate tasks), team scope and composition of a task team may change throughout a mission, and little or no information about the

vehicles may be directly available to the mission system (e.g., a vehicle failure may become known only after the vehicle has failed to show up for refuelling past a given deadline).

2.3 Problem Formulation

Therefore, we consider a formulation similar to that applied in the web server farm management problem, considered in [21, 62]. Consider a collection of M agents of the same capability and N designated mission sites. Distributed amongst these mission sites, consider K distinct types of agent task requests. To simplify this discussion, let $L_k(t)$ represent the aggregate arrival rates of task requests (mainly because task requests are handled by a task allocation schemes at a lower level in the mission system architecture) [97]. As shown in the web server farm management problem formulation [21, 62], using aggregate arrival rates provides with a simplified resource management problem for which we can find a tractable solution for the problems of task assignment and agent task scheduling problems under some simplified assumptions. Next, assume that tasks are assigned according to costs based on their importance (assume that there is a metric for defining such costs for now) and that these costs may differ between different mission tasks.

Now, there are three types of decisions we must make as part of this process: the number of agents assigned to a mission task, the agent's task assignment as a member of a mission group, and the agent task scheduling required to complete this task. For now, we will assume that no agent can be a member of two mission groups.

Because of the problem's size, a sub-optimal approach is used to separate the agent allocation problem from the agent task assignment / scheduling problems (which are normally managed by sub-systems in the mission architecture). In this formulation, a simplified task assignment and scheduling problem is introduced and formulated for the multi-agent mission allocation problem. This model is adapted from web server farm example found in [21], which is an adaptation of the model found in [62]. For this reason, we state the problem and the main points of the approach in this section, referring the reader to [21] and [62] for details. In addition, we use the notation provided in [21] for continuity in this discussion.

In the task allocation and scheduling problems, we will assume that agent mission allocation and task requests rates remain fixed during the interval of observation (which are length $\frac{T}{2}$ for now). In each interval, we will use the symbol I to indicate the allocation of each agent in the task space, and the symbol L denote a vector of

aggregated task requests L_k. Therefore, the solution to this Task Assignment and Scheduling Problem will be a function of the pair (I, L). Now, using the fluid model approach as shown in [21] and [62], the task assignment and scheduling decision variables reduce to $\lambda_{i,k}$, which describes the arrival of a class k task request to agent i, and $\psi_{i,k}$, which describes the fraction of time agent i will be assigned to task request k in each mission group. Note that the task requests across any mission group for any series of task requests must be less than or equal to the aggregate task arrival rate L_k for the mission group. To compute L_k for a given class of task requests, we can assume based on prior experience / observations that tasks of different types will require at minimum a certain number of agents to be actively completed. Therefore, we can derive an appropriate value for L_k for any given task.

Next, since we assume that each task will provide a benefit B_k, and that the expected time agent i spends completing a task request of type k over each time interval $\frac{T}{2}$ is given by $\frac{\lambda_{i,k}}{\mu_k}$, then the expected profit guaranteed by completing a task request of type k using agent i is

$$B_k \frac{\lambda_{i,k}}{\mu_k}$$

From this we can generate the following optimization problem:

$$\begin{array}{lll} \max_{\lambda_{i,k},\psi_{i,k}} & \sum_{i=1}^{M}\sum_{k=1}^{K} B_k \frac{\lambda_{i,k}}{\mu_k} & \\ \text{subj. to} & \sum_{i=1}^{M} \lambda_{i,k} \leq L_k & k \in \mathcal{K} \\ & \lambda_{i,k} = 0, & \text{if } I(i,k) = 0, i \in \mathcal{V}, k \in \mathcal{K} \\ & \lambda_{i,k} \geq 0, & \text{if } I(i,k) = 1, i \in \mathcal{V}, k \in \mathcal{K} \\ & \sum_{k=1}^{K} \psi_{i,k} \leq 1 & i \in \mathcal{V} \\ & \psi_{i,k} = 0, & \text{if } I(i,k) = 0, i \in \mathcal{V}, k \in \mathcal{K} \\ & \psi_{i,k} \geq 0, & \text{if } I(i,k) = 1, i \in \mathcal{V}, k \in \mathcal{K} \end{array} \tag{2.1}$$

where $\mathcal{V} = \{1, ..., M\}$ and $\mathcal{K} = \{1, ..., K\}$. In this problem note that $I(i,k)$ indicates whether an agent is allocated to a mission task group associated with task request of class k. Now, the reason this formulation is valid for this application is that each agent in a task group may provide assistance on a variety of tasks – depending on the needs of the mission group. In addition, multiple agents may be required to perform a task, and as a result the formulation accounts for the fact that an agent may be required to switch its duties if a task is completed in this interval.

As shown in [21] and [62], Eq. (2.1) can be solved analytically, and the optimal policy is greedy. In other words, the policy must seek to serve all requests where the

most benefit to the system is derived first before serving other classes (which provide a lower benefit to the system). In addition, we know that we will have a near-optimal policy to the task assignment / scheduling problem by assigning agents (according to the greedy policy) to maximize the ratio $\frac{B_k}{\mu_k}$ under the assumptions above. Therefore, we can use the optimal value for any pair (I, L) and calculate the single-stage cost using the above, which we will denote as $g(I, L)$.

2.4 MDP Model for the UAV SWARM Resource Management Problem

Given the above information, we can extend this problem formulation to account for vehicle- and system-level health-related issues at the mission planning level and defined the MDP model for the UAV SWARM Resource Management Problem. First, define the state as the vector (I, H, L) where I indicates the allocation of each agent in the task space, H indicates the maintenance / health state of each agent in the task space, and the symbol L denote a vector of aggregated task requests. Next, define the actions A on the same space as I, where A indicates the new agent-mission group configuration. Note that only agents which have been "deallocated" from a task group can be re-allocated to a new task group. Here, de-allocated means that an agent is either (i) waiting at a ready location, (ii) available after maintenance, or (iii) removed from a task group which no longer needs its services or is not beneficial to the system. Next, a state (I, H, L) will transition under action A to state $(\bar{A}, \bar{H}, \bar{L})$ with probability $P_A((I, H, L), (\bar{A}, \bar{H}, \bar{L}))$. Finally, as stated above, consider the single-stage cost for being in state (I, H, L) by using $g(I, L)$. Here, we are assuming that L remains constant over each time step.

With the MDP model defined, the problem must be formulated and solved in a manor that promotes a feasible real-time implementation. As a result, the next four chapters of this thesis will explain our approach to formulating and solving large-scale problems (such as this) for real-time use. In Chapter 3, we consider the general problem of large-scale MDPs. Leveraging the approximate linear programming method, we propose an algorithm for automatic selection of approximation architectures. Next, in Chapter 4 the methods provided in Chapter 3 are used to implement and manage multi-vehicle mission problems. Finally, in Chapters 5 and 6 an indoor multi-vehicle test environment is used to implement and test these methods over extended periods time using a fully-autonomous system with little operator interaction.

Chapter 3

Basis Function Generation for Approximate Dynamic Programming Methods

Dynamic Programming can be used to find decision-making policies for problems where the goal is to minimize the total cost of decisions made over a sequence of stages [9]. From each state, the dynamic programming algorithm can be used to determine the best action that will minimize the cost-to-go function [7]. For a Markov Decision Process (MDP) with state space S, action space A, probability transition matrix $P_u \in \mathbb{R}^{|S|\times|S|}$ for policy $u : S \mapsto A$, local cost $g : S \times A \mapsto \mathbb{R}$ and discount factor $\alpha \in (0, 1)$, the cost-to-go function $J_u : S \mapsto \mathbb{R}$ associated with policy u can be expressed by

$$J_u(x) \quad = \quad E\left[\sum_{k=0}^{\infty} \alpha^k g(x_k, u(x_k))|x_0 = x\right]$$

and the optimal cost-to-go J^* from each state $x \in S$ is defined by

$$J^*(x) \quad = \quad \min_u J_u(x)$$

Here, the main goal in solving this optimization problem is to find the optimal policy u^* such that

$$J^*(x) \quad = \quad J_{u^*}(x)$$

for all $x \in \mathcal{S}$. Here, using the Principle of Optimality [7], we can write Bellman's Equation as

$$J(x) \quad = \quad \min_{a \in A(x)} E\left[g_a(x) + \alpha \sum_{y \in S} P_a(x,y) J(y) \right].$$

Note that $P_a(x,y)$ defines the probability that the system transitions from state $x \in S$ to state $y \in S$ when using action $a \in A_x$. Also, we will define $g_a(x)$ which is the local cost applied at state $x \in S$ when using action $a \in A(x)$. In addition, Bellmans Equation can be re-written as

$$(TJ)(x) \quad = \quad \min_{a \in \mathcal{A}_x} \left[g_a(x) + \alpha \sum_{y \in S} P_a(x,y) J(y) \right] \quad \forall \ x \in S$$

where T is the dynamic programming operator, such that for finite $|S|$,

$$TJ \quad = \quad \min_u \left[g_u + \alpha P_u J \right]$$

where the optimal control policy u at each state can be found using:

$$u^*(x) \quad = \quad \arg \min_{a \in A(x)} E\left[g_a(x) + \alpha \sum_{y \in S} P_a(x,y) J^*(y) \right]$$

As stated in [9], the state and action space for most real-world problems of interest are too large for the problem to be solved in real-time using modest computational resources. Therefore, approximation methods that require less computational resources and can be solved in shorter periods of time are being used to find a near-optimal action to use from a given state. In this thesis, we consider methods that approximate the optimal cost-to-go-function J^* by a linear combination of basis functions: $J^* \approx \Phi r$.

The purpose of this chapter is to present a method designed to automatically find a set of basis functions for a given MDP so that the problem can be formulated and solved in real-time. This is a fundamental question related to use of approximate dynamic programming in real-time systems. In this chapter, we formulate the basis function function generation problem. Next, we present an algorithm designed to iteratively generate basis functions in real-time. Then, an error bound comparing the optimal solution to the original dynamic program to the approximate solution using the basis functions is presented. Experimental results using the algorithm for the

UAV mission management problem are presented in Chapter 4.

3.1 Approximate Linear Programming Problem Formulation

As mentioned above, most real world problems of interest may be formulated as a very large dynamic programming problem; however, these problems are likely to be computationally intractable for most computer systems to solve in real-time (even for a problem of reasonable size). One such option is to formulate and solve the problem using an Approximate Linear Programming [21, 83]. To use this approach, the original Dynamic Programming problem of interest must be reformulated into a Linear Programming problem. Since

$$J \leq TJ \leq T^2J \leq ... \leq T^nJ$$

the dynamic programming problem can be rewritten in the following form:

$$\begin{aligned} \max \quad & c^TJ \\ \text{subject to} \quad & TJ \geq J \end{aligned} \tag{3.1}$$

where c is the state relevance weights for the problem. To ensure that there is a unique optimal solution to this problem, we must ensure that $c > 0$. Problem (3.1) is equivalent to:

$$\begin{aligned} \max \quad & c^TJ \\ \text{subject to} \quad & g_(x,a) + \alpha \textstyle\sum_{y \in S} P_a(x,y)J(y) \geq J(x) \quad \forall x \in S, a \in A_x \end{aligned}$$

This is called the exact Linear Programming problem. Next, using the basis functions $\phi_1, ...\phi_K$, we can restrict J to be of the form Φr. Here, $\phi_i \in \mathbb{R}^{|S|\times 1}$ for $i \in \{1, ..., K\}$ represents a column of the matrix $\Phi \in \mathbb{R}^{|S|\times K}$ and $\Phi(x) \in \mathbb{R}^{1\times K}$ represents the row of the matrix Φ associated with state $x \in S$. In addition, $r \in \mathbb{R}^{K\times 1}$ is the column right multiplying the matrix Φ so that $J \approx \Phi r$. Using this notation, we can re-write Eq. (3.1) as:

$$\begin{aligned} \max \quad & c^T\Phi r \\ \text{subject to} \quad & T\Phi r \geq \Phi r \end{aligned}$$

or in other words:

$$\begin{array}{rl} \max & c^T \Phi r \\ \text{subject to} & g_(x,a) + \alpha \sum_{y \in S} P_a(x,y)\Phi(y)r \geq \Phi(x)r \quad \forall x \in S, a \in A_x \end{array} \tag{3.2}$$

This is the Approximate Linear Programming (ALP) formulation of the original Dynamic Programming problem.

3.2 Issue: Selecting Basis Functions

One of the major obstacles faced when using Approximate Linear Programming (ALP) is the selection of appropriate basis vectors and state relevance weights. In many cases, selecting the appropriate parameters used to find Φ is based on experience. A variety of methods for value function approximation have been proposed in the literature (as discussed in Chapter 1) and many are summarized in [9]. The history of parametric function approximations dates back to Samuel in 1959 when he used a parametric approximation to allow a computer to automatically generate policies for playing checkers [65, 81]. Currently, there are some papers in the literature on procedures for selecting parameters for basis functions, however there is a need for more research in this area. As stated earlier, although many of these techniques in the current literature have promising results, many of these algorithms cannot be employed efficiently for large-scale problems.

Our goal is to improve the cost-to-go approximation by automatically generating basis functions. Our approach uses the Bellman update to determine the best direction for cost improvement. Using a weighted 1-norm, our goal is to select a basis vector which minimizes

$$\|T\Phi r - \Phi r\|_{1,c}$$

where we define $u : S \mapsto A$ as a stationary policy and

$$\begin{aligned} \|J\|_{1,c} &= \sum_{x \in S} c(x)|J(x)| \\ \sum_{x \in S} c(x) &= 1 \end{aligned}$$

From the proof of Theorem 1 in [22] we know that for an arbitrary $J \leq J^*$

$$\begin{aligned} \|J_{u_J} - J^*\|_{1,c} &\leq \frac{1}{1-\alpha}\mu_{u_J,c}^T(TJ - J) \\ &= \|TJ - J\|_{1,\mu_{u_J,c}} \end{aligned}$$

where u_J is the greedy policy with respect to J and

$$\mu_{u,c} = (1-\alpha)c^T(I - \alpha P_u)^{-1}$$

where $\mu_{u,c}$ is a measure of the relative importance of each state under policy u [21]. If we let $J = \Phi r$, then

$$\|J_{u_J} - J^*\|_{1,c} \leq \frac{1}{1-\alpha}\|T\Phi r - \Phi r\|_{1,\mu_{u_J,c}}$$

Therefore, by using $\|T\Phi r - \Phi r\|_{1,\mu_{u,c}}$ to generate our basis functions, we take into account their impact on the performance of the policy that is generated. In addition, by using this weighted norm, the basis function which is selected will also place more emphasis on the weighted states (based on their relevance weights) – thereby emphasizing states of importance to the problem setup.

Using this relationship, our goal is to select the best vector d to improve the cost-to-go approximation at each iteration. By starting with a pre-generated set of basis functions, an LP can be formulated which selects the vector d that best reduces the Bellman Error using an online approach that does not assume a preselected form each basis function. One way that we could do this is to use a technique like Singular Value Decomposition, however this technique requires $\mathcal{O}(n^3)$ multiplications for a square matrix. Another possibility is to use the properties of P_u and g_u (generated from the vector r found by solving the ALP) to decide what d should be.

Using such an approach, we have the following result:

Lemma 3.1. *Given r, the vector $d = J_u - \Phi r$ minimizes $\|T_u(\Phi r + d) - (\Phi r + d)\|_{1,\mu}$.*

Proof:
First, note that if $x = \overline{0}$, then $\|x\|_{1,\mu} = 0$. Let $\Delta = T_u(\Phi r + d) - (\Phi r + d)$. So, if

$$\begin{aligned} T_u(\Phi r + d) - (\Phi r + d) &= 0 \\ g_u + \alpha P_u(\Phi r + d) - (\Phi r + d) &= 0 \\ g_u - (I - \alpha P_u)\Phi r &= (I - \alpha P_u)d \end{aligned}$$

So,

$$\begin{aligned} d &= (I - \alpha P_u)^{-1} g_u - \Phi r \\ &= \sum_{j=0}^{\infty} (\alpha P_u)^j g_u - \Phi r \\ &= J_u - \Phi r \end{aligned}$$

□

The main implementation issue with this result is that since $n = |\mathcal{S}|$ may be large, it may be difficult to calculate $J_u = (I - \alpha P_u)^{-1} g_u$, which requires at least $\mathcal{O}(n^3)$ operations, in a reasonable period of time. Notice that we can make an approximation of d by using $d = \sum_{j=0}^{M} (\alpha P_u)^j g_u - \Phi_k r_k$ which is $\mathcal{O}(Mn^2)$. Also note that we could use the second eigenvalue of the matrix P_u to decide on what M should be, however the number of calculations to make this determination for each policy is at least $\mathcal{O}(n^2)$.

3.3 Basis Function Generation Algorithm

Given an MDP (S, A, P, g) with policies of the form $u : S \mapsto A$, discount factor α, state relevance weights c, and basis functions Φ, we can find an approximation of the cost-to-go function $J^* = \sum_{j=0}^{\infty} (\alpha P_{u^*})^j g_{u^*}$ under policy u^* of the form Φr using:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g_a(x) + \alpha \sum_{y \in S} P_a(x, y) \Phi(y) r \geq \Phi(x) r \quad \forall (x, a) \in D \end{array} \tag{3.3}$$

where $D \subseteq S \times A$. Here, let $\Phi \tilde{r}$ represent the approximate cost found using Eq. (3.3). To reduce the number of calculations needed to make a "good" approximation of J^* with guarantees on the error, one can sample both the constraints *and* the future states considered in the ALP. In doing so, we do not have to evaluate $\Phi(y)$ every future state $y \in S$ to have a "good" approximation of the future cost-to-go from state $x \in S$. As a result, the revised problem statement from Eq. (3.3) becomes:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g_a(x) + \frac{\alpha}{M} \sum_{i=1}^{M} \Phi(y_i) r \geq \Phi(x) r \quad \forall (x, a) \in C \end{array} \tag{3.4}$$

where $C \subset D$ defines the set of constraints being used and for each state $x \in S$, a set of M future states $\{y_1, y_2, ..., y_M\}$ are used to approximate the quantity

$$\sum_{y \in S} P_a(x,y)\Phi(y) \quad \approx \quad \frac{1}{M}\sum_{i=1}^{M} \Phi(y_i) \tag{3.5}$$

Let $\Phi\hat{r}$ represent the approximate cost found using Eq. (3.4). This solution gives us a computationally efficient approximation of J^* using $\Phi\hat{r}$.

By solving the above ALP problem iteratively, the following process can be used to generate new basis functions:

1. Start with $\Phi_0 = \Phi$
2. Calculate the best r_k, according to

$$\begin{aligned} \max_{r_k} \quad & c^T \Phi_k r_k \\ \text{subj. to} \quad & g_a(x) + \tfrac{\alpha}{M}\textstyle\sum_{i=1}^{M} \Phi(y_i) r \geq \Phi(x) r \quad \forall (x,a) \in C \end{aligned}$$

3. Using updated r_k, determine the new basis function ϕ_{k+1} and augment the matrix Φ_k so that $\Phi_{k+1} = [\Phi_k \quad \phi_{k+1}]$
4. Repeat steps 2 and 3 until $\|T\Phi_k r_k - \Phi_k r_k\|_{1,c} \leq SC$ where SC is the predetermined stopping constant

The main part of this process of interest here is Step 3. First, to gain intuition into this method, we will start by assuming that using Eq. (3.4), r_k is calculated given Φ_k. Using the policy $u : S \mapsto A$, we can calculate each additional basis function using the update provided by Bellman's equation where

$$\phi_{k+1} \quad = \quad g_u + \alpha P_u \Phi_k r_k$$

or in other words, for each state $x \in S$ where $a = u(x)$

$$\phi_{k+1}(x) \quad = \quad g_a(x) + \alpha \sum_{y \in S} P_a(x,y)\Phi_k(y) r_k$$

As mentioned above, the main problem encountered with the above is the computational complexity of this process. Since $|S|$ may be very large, an approximation must be used to generate $\alpha \sum_{y \in S} P_a(x,y)\Phi_k(y) r_k$ for each state. Here, we will use the approximation similar to the one shown in Eq. (3.5), such that the approximate

cost-to-go for the sampled set of next states x_{k+1} from the current state $x_k = x$ will be generated using sampled trajectories $\{x_0, x_1, x_2, ..., x_{k+1}\}$. As shown in [8], using sampled trajectories generated via simulation to estimate future cost-to-go can improve cost-to-go estimates and provide results in policies.

Using this trajectory-based approximation method for the future cost-to-go from each state allows us to develop a simple closed-form approximation structure for computing basis functions *implicitly* using sampled trajectories by storing only multipliers r_0, r_1, ... r_N. For example, ϕ_1 is approximated by:

$$\phi_1(x) \approx \frac{1}{M}\sum_{k=1}^{M}\left(g_{a_0}(x_0^k) + \alpha\Phi_0(x_1^k)r_0\right)$$

In this case, we are averaging the basis functions computed using M trajectories of the form $\{x_0^k, x_1^k\}$ for $k \in \{1, ..., M\}$. Likewise, the $(N+1)$th basis function is generated using M trajectories of the form $\{x_0^k, x_1^k, x_2^k, ..., x_{N+1}^k\}$ for $k \in \{1, ..., M\}$ such that:

$$\phi_{(N+1)}(x_0) \approx \frac{1}{M}\sum_{k=0}^{M}\sum_{i=0}^{N}\left[g_{a_i^k}(x_i^k) \quad \Phi_0(x_{i+1}^k)\right] m_{Ni}$$

where:

$$m_{Ni} = \begin{cases} \alpha\sum_{j=(i-1)}^{N-1} m_{j(i-1)}r_{N(j+1)} & \text{when } i > 0 \\ [1 \quad \alpha r_{N0}]^T & \text{when } i = 0 \end{cases}$$

and $r_{00} = r_0 \in \mathbb{R}^{1\times K}$, $r_N = [r_{N0}\, r_{N1}\, ...\, r_{NN}]^T$ such that $r_{N0} \in \mathbb{R}^{1\times K}$ and $r_{Ni} \in \mathbb{R}$ for $i \in \{1, ..., N\}$.

As a result, to generate the $(N+1)$th basis function, only $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$ values of m_{ij} must be saved. Since $|m_{ij}| = |r_{00}| + 1$, we must store $\frac{N(N+1)}{2}(|r_{00}| + 1)$ values to generate each $\phi_1(x_0)$,...,$\phi_{N+1}(x_0)$ given $g_{a_0}(x_0)$,...,$g_{a_N}(x_N)$ and $\phi_0(x_1)$,...,$\phi_0(x_{N+1})$. A description expanding on the development of this formulation can be found in the appendix.

Based on this derivation, the basis function generation algorithm can be defined as shown in Table (3.1). There are many computational advantages to this approach. First, due to the fact that a set of multipliers are used to generate each basis function at a given state, the basis functions are implicitly defined by the multipliers, resulting in an algorithm that does not require a large amount of memory to generate future basis functions and that scales well with problem size. Thus, the algorithm offers users the opportunity to distribute the problem formulation across many different

computation resources so that it can be solved faster (making it ideal for real-time use). Finally, since the algorithm is providing basis function updates designed to decrease the Bellman error, the approximated cost-to-go function will improve as basis functions are added to the problem.

In summary, using this algorithm one does not have to have prior experience with the problem to be solved since the basis function generation algorithm will automatically calculate basis functions that improves the approximated cost-to-go function based on the MDP provided.

3.3.1 Numerical Complexity of the Basis Function Generation Algorithm

Now, before we can calculate the values of the basis functions, we must calculate each m_{Ni} in the $(N+1)$th step. Here, we have:

$$\begin{aligned}
&\text{if } i = 0 \;\Rightarrow\; |r_{00}| \text{ multiplications (by } \alpha) \\
&\text{if } i \neq 0 \;\Rightarrow\; \alpha \sum_{j=(i-1)}^{N-1} m_{j(i-1)} r_{N(j+1)} \\
&\qquad\Rightarrow\; (N-i+1)(|r_{00}|+1) \text{ multiplications (by each } r_{N(j+1)}) \\
&\qquad\quad\; (N-i)(|r_{00}|+1) \text{ additions (adding up all of the } m_{j(i-1)} r_{N(j+1)}) \\
&\qquad\quad\; (|r_{00}|+1) \text{ multiplications (by } \alpha)
\end{aligned}$$

Therefore, at the $(N+1)$th step, the number of calculations needed to find all of the m_{Ni}'s are:

$$\begin{aligned}
|r_{00}| + \sum_{i=1}^{N}(N-i+2)(|r_{00}|+1) &= (|r_{00}|+1)\sum_{i=1}^{N}(N-i+2) \\
&= (|r_{00}|+1)\left(N^2 - \frac{N(N+1)}{2} + 2N\right) + |r_{00}| \\
&= (|r_{00}|+1)\frac{N(N+3)}{2} + |r_{00}| \text{ multiplications} \\
\text{and} \quad \sum_{i=1}^{N}(N-i)(|r_{00}|+1) &= (|r_{00}|+1)\left(N^2 - \frac{N(N+1)}{2}\right) \\
&= (|r_{00}|+1)\frac{N(N-1)}{2} \text{ additions}
\end{aligned}$$

Step 0: Start with $\Phi_0 = \bar{1}$ and calculate r_0 using

$$\begin{aligned} \max_{r_0} \quad & c^T \Phi_0 r_0 \\ \text{subj. to} \quad & g_a(x) + \tfrac{\alpha}{M} \textstyle\sum_{i=1}^{M} \Phi_0(y_i) r \geq \Phi_0(x) r_0 \quad \forall (x,a) \in C \end{aligned}$$

Using this r_0, find $m_{00} = [1 \;\; \alpha r_0]^T$ and let $k = 1$.

Step 1: Generate the constraints for the ALP in the following way:
For each state-action pair $(x, a) \in C$, generate M trajectories of the form $\{x_0^j, x_1^j, x_2^j, ..., x_{k+1}^j\}$ where $x_0^j = x$ for $j \in \{1, ..., M\}$. Using these trajectories, generate each constraint in the ALP by generating the basis functions online using:

$$\frac{1}{M} \sum_{j=1}^{M} \left(\Phi_k(x_0^j) - \alpha \Phi_k(x_1^j) \right) r \geq g_a(x) \quad \forall (x,a) \in C$$

where $\Phi_k(x) = [\Phi_0(x) \;\; \phi_1(x) \;\; ... \;\; \phi_k(x)]$ such that each basis function $\phi_{i+1}(x_0^j)$ for $i \in \{0, ..., k-1\}$ is calculated using

$$\phi_{i+1}(x_0^j) \;\; = \;\; \sum_{l=0}^{i} \left[g_{a_l^j}(x_l^j) \;\; \Phi_0(x_{l+1}^j) \right] m_{il}$$

Step 2: Calculate the best r_k, according to

$$\begin{aligned} \max_{r_k} \quad & c^T \Phi_k r_k \\ \text{subj. to} \quad & \tfrac{1}{M} \textstyle\sum_{j=1}^{M} \left(\Phi_k(x_0^j) - \alpha \Phi_k(x_1^j) \right) r_k \geq g_a(x) \quad \forall (x,a) \in C \end{aligned}$$

Step 3: Using updated r_k, determine the multipliers

$$m_{ki} \;\; = \;\; \begin{cases} \alpha \sum_{j=(i-1)}^{k-1} m_{j(i-1)} r_{k(j+1)} & \text{when } \; i > 0 \\ [1 \;\; \alpha r_{k0}]^T & \text{when } \; i = 0 \end{cases}$$

where $r_{00} = r_0 \in \mathbb{R}^{1 \times K}$, $r_k = [r_{k0} \, r_{k1} \, ... \, r_{kk}]^T$ such that $r_{k0} \in \mathbb{R}^{1 \times K}$ and $r_{ki} \in \mathbb{R}$ for $i \in \{1, ..., k\}$.

Table 3.1: The Basis Function Generation Algorithm

Step 4: Repeat Steps 1 through 3 (letting $k = k + 1$) until $\|\hat{T}\Phi_k r_k - \Phi_k r_k\|_{1,c} \leq SC$ where SC is the predetermined stopping constant and $\hat{T}\Phi_k r_k = g_u + \alpha \hat{P}_u \Phi_k r_k$ or $k = \bar{N}$ where $\bar{N}$ is the pre-determined number of basis functions to be calculated for the problem.

Therefore, using the multipliers generated in Step 3, each basis function ϕ_{k+1} can be computed by using M trajectories of the form $\{x_0^j, x_1^j, x_2^j, ..., x_{k+1}^j\}$ for $j \in \{1, ..., M\}$ such that:

$$\phi_{(k+1)}(x_0) = \frac{1}{M} \sum_{j=0}^{M} \sum_{i=0}^{k} \left[g_{a_i^j}(x_i^j) \quad \Phi_0(x_{i+1}^j) \right] m_{ki}$$

Table 3.2: The Basis Function Generation Algorithm (continued)

This means that the computational complexity of calculating the multipliers m_{N1},...,m_{NN} at the $(N+1)$th step assuming you have all of the m_{ij} where $i < N$ is

$$(|r_{00}| + 1)(N^2 + N) + |r_{00}| \text{ total operations}$$

Now, given that we have all of the m_{ij}'s calculated for all $i \leq N$, then the computational complexity in calculating $\phi_i(x_0)$ is:

$$i(|r_{00}| + 1) \text{ multiplications}$$
$$(i-1)(|r_{00}| + 1) \text{ additions}$$

Therefore, to generate all of the basis functions $\phi_1(x_0)$,...,$\phi_{N+1}(x_0)$ using the m_{ij}'s, we must perform:

$$\sum_{i=1}^{N+1} i(|r_{00}| + 1) = (|r_{00}| + 1)\frac{(N+1)(N+2)}{2} \text{ multiplications}$$

$$\sum_{i=1}^{N+1} (i-1)(|r_{00}| + 1) = (|r_{00}| + 1)\frac{N(N+1)}{2} \text{ additions}$$

$$\Rightarrow (|r_{00}| + 1)(N+1)^2 \text{ operations for each trajectory}$$

Therefore, if we sample M trajectories to generate our basis functions and perform this operation for K states, we must perform

$$MK(N+1)^2(|r_{00}|+1)+(N^2+N)(|r_{00}|+1)+|r_{00}|$$

total operations at each iteration of the algorithm to calculate all of the basis functions needed for the constraints in the LP at each stage. Note that this number does not include the calculations needed to generate each of the trajectories at each stage.

3.4 Error Bound for a Single Policy, Single Iteration Update

Before we discuss an error bound for a single policy, it is useful to discuss the structure of the basis functions. Since each basis function is calculated using the equation $\phi_{k+1} = g + \alpha P \Phi_k r_k$, there is a relationship between the convergence of the basis function generation algorithm and value iteration.

Lemma 3.2. *Given an MDP (S, A, P, g) with a single policy $u : S \mapsto A$, discount factor α, state relevance weights c, and basis functions Φ_0. Let $\Phi_k = [\Phi_0 \ \phi_1 \ ... \ \phi_k]$ such that $\phi_j = g + \alpha P \Phi_{j-1} r_{j-1}$ for $j \in \{1, ..., k\}$. Then, as $k \to \infty$, the cost-to-go function $J_k = \Phi_k r_k$ will converge to the optimal cost-to-go function J^* at least as fast as value iteration.*

Proof:

Recall that $\phi_k = T\Phi_{k-1} r_{k-1}$. Since $T\Phi_{k-1} r_{k-1} \geq \Phi_{k-1} r_{k-1}$, by monotonicity of T we have $T\phi_k \geq \phi_k$ and ϕ_k is a feasible solution to the ALP with basis function $[\Phi_{k-1}\phi_k]$.

This means that the value iteration update is a special case of the basis function generation update, which means that as $k \to \infty$, the cost-to-go function $J_k = \Phi_k r_k$ will converge to the optimal cost-to-go function J^* at least as fast as value iteration.

□

Next, it is very important to understand how the cost-to-go approximation evolves as for the single policy case. The main reason for studying the single policy case is that a base policy is used to generated the trajectories used to approximate the future cost-to-go. Therefore, an error bound for the single policy case is very relevant in understanding how the sampled trajectories from this policy affect the cost-to-go

approximations. As described earlier, given an MDP (S, A, P, g) with a single policy $u : S \mapsto A$, discount factor α, state relevance weights c, and basis functions Φ, we can find an approximation of the optimal cost $J^* = \sum_{j=0}^{\infty}(\alpha P)^j g$ under this policy of the form Φr using:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g + \alpha P \Phi r \geq \Phi r \end{array} \tag{3.6}$$

which yields $\tilde{r}$. Next, if we just sample the states used to find the future cost at each state $x \in S$, the revised problem statement from Eq. (3.6) becomes:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g(x) + \frac{\alpha}{M} \sum_{i=1}^{M} \Phi(y_i) r \geq \Phi(x) r \quad \forall x \in S \end{array} \tag{3.7}$$

which we can write as:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g + \alpha \hat{P} \Phi r \geq \Phi r \end{array} \tag{3.8}$$

which yields $\hat{r}$ where $\hat{P}$ is an approximation to P induced by sampling.

Similarly, we can bound the error resulting from the sampling approximation for a single policy using a method called cost shaping. Here, we have a similar problem formulation as above, except for the addition of $\eta > 0$ and $\rho : S \mapsto [1, \infty)$. Therefore, we can find an approximation of the optimal cost $J^* = \sum_{j=0}^{\infty}(\alpha P)^j g$ of the form Φr using:

$$\begin{array}{ll} \max_{s \geq 0, r} & c^T \Phi r - \eta s \\ \text{subj. to} & g + \alpha P \Phi r + \rho s \geq \Phi r \end{array} \tag{3.9}$$

which yields $(\tilde{r}, \tilde{s})$ where $\eta > 0$ and $\rho : S \mapsto [1, \infty)$. Next, if we just sample the states used to find the future cost at each state $x \in S$, the revised problem statement from Eq. (3.9) becomes:

$$\begin{array}{ll} \max_{s \geq 0, r} & c^T \Phi r - \eta s \\ \text{subj. to} & g(x) + \frac{\alpha}{M} \sum_{i=1}^{M} \phi^T(y_i) r - \rho s \geq \phi^T(x) r \quad \forall x \in S \end{array} \tag{3.10}$$

which we can write as:

$$\begin{array}{ll} \max_{s\geq 0,r} & c^T\Phi r - \eta s \\ \text{subj. to} & g + \alpha\hat{P}\Phi r - \rho s \geq \Phi r \end{array} \tag{3.11}$$

which yields $(\hat{r}, \hat{s})$ where $\hat{P}$ is an approximation to P induced by sampling.

In addition, we can generate a new basis function using:

$$\phi_1 = g + \alpha P\Phi\tilde{r}_0 \tag{3.12}$$

which can be approximated using the basis function generation algorithm by:

$$\hat{\phi}_1 = g + \alpha\bar{P}\Phi\hat{r}_0 \tag{3.13}$$

where $\bar{P}$ is an approximation to P induced by sampling. Then, to understand how our approximation affects the solution, we can relate the solutions to Equations (3.9) and (3.11) when using $\Phi_1 = [\Phi_0\, \phi_1]$ in Equation (3.9) and $\hat{\Phi}_1 = [\Phi_0\, \hat{\phi}_1]$ in Equation (3.11).

Therefore, let:

$$\begin{array}{ll} \max_{r} & c^T\Phi_1 r \\ \text{subj. to} & g + \alpha P\Phi_1 r \geq \Phi_1 r \end{array} \tag{3.14}$$

which yields $\tilde{r}$ and let:

$$\begin{array}{ll} \max_{s\geq 0,r} & c^T\hat{\Phi}_1 r - \eta s \\ \text{subj. to} & g + \alpha\hat{P}\hat{\Phi}_1 r - \rho s \geq \hat{\Phi}_1 r \end{array} \tag{3.15}$$

which yields $(\hat{r}, \hat{s})$.

Before stating the lemma, we will define the following terms:

$$J^* = (I - \alpha P)^{-1} g \tag{3.16}$$

$$\hat{\mu} = (1-\alpha)c^T(I - \alpha\hat{P})^{-1} \tag{3.17}$$

Define the jth column Φ_0^j in Φ_0 where $j \in \{1, ..., K\}$, we have that $\Phi_0^j(x) \in [a^j(x), b^j(x)]$.

Next, let

$$\Phi_1 = [\Phi_0 \; \phi_1] \tag{3.18}$$

$$\hat{\Phi}_1 = [\Phi_0 \; \hat{\phi}_1] \tag{3.19}$$

Then, we have the following lemma:

Lemma 3.3. *Given an MDP (S, A, P, g) with a single policy $u : S \mapsto A$, discount factor α, state relevance weights c, and basis functions Φ_1 and $\hat{\Phi}_1$, define the approximations of P induced by sampling to be $\hat{P}$ and $\bar{P}$. Then, Eq. (3.15) can be used to approximate Eq. (3.14) where*

$$\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} \;=\; \|J^* - \Phi_1\tilde{r}\|_{1,c} + \frac{2\alpha}{1-\alpha}\|(P-\hat{P})J^*\|_{1,\hat{\mu}} + \|\left[0 \mid (P-\bar{P})\Phi \; \delta r_0\right]\tilde{r}\|_{1,c} + \eta\bar{s}$$

for $\rho : S \mapsto [1, \infty)$ where $\eta = 2c^T(I - \alpha\hat{P})^{-1}\rho$, $\delta r_0 = \bar{1}\|\hat{r}_0 - \tilde{r}_0\|_\infty \in \mathbb{R}^K$, and

$$\bar{s} \;=\; \max_{x \in S}\left[\frac{1}{\rho(x)}\left(\alpha(\hat{P}-P)\Phi_1\tilde{r} + \hat{\varepsilon}(\tilde{r})\right)(x), 0\right]$$

where $\hat{\varepsilon}(\tilde{r}) = (\alpha\hat{P} - I)\left[(\bar{P}-P)\Phi(\hat{r}_0 - \tilde{r}_0)\right]\tilde{r}$.

Proof:

First, assume Eq. (3.14) yields $\tilde{r}$ and Eq. (3.15) yields $(\hat{r}, \hat{s})$. Using the constraint in Eq. (3.15):

$$\begin{aligned} g + \alpha\hat{P}\hat{\Phi}_1 r + \rho s &\geq \hat{\Phi}_1 r \\ (I - \alpha\hat{P})^{-1}g + (I - \alpha\hat{P})^{-1}\rho s &\geq \hat{\Phi}_1 r \\ J_\varepsilon + (I - \alpha\hat{P})^{-1}\rho\hat{s} &\geq \hat{\Phi}_1\hat{r} \\ c^T J_\varepsilon + c^T(I - \alpha\hat{P})^{-1}\rho\hat{s} &\geq c^T\hat{\Phi}_1\hat{r} \end{aligned}$$

which means that if $\eta \geq c^T(I - \alpha\hat{P})^{-1}\rho$, then $c^T\hat{\Phi}_1\hat{r} - \eta\hat{s}$ will be upper bounded by $c^T J_\varepsilon$.

Next,

$$g + \alpha\hat{P}\hat{\Phi}_1\tilde{r} - \hat{\Phi}_1\tilde{r} \;=\; g + \alpha\hat{P}\Phi_1\tilde{r} + \hat{\varepsilon}(\tilde{r}) - \Phi_1\tilde{r} \tag{3.20}$$

such that

$$\begin{aligned}\hat{\varepsilon}(\tilde{r}) &= (\alpha\hat{P} - I)(\hat{\Phi}_1 - \Phi_1)\tilde{r} \\ &= (\alpha\hat{P} - I)\left[0 \mid (\bar{P} - P)\Phi(\hat{r}_0 - \tilde{r}_0)\right]\tilde{r}\end{aligned}$$

where $\tilde{r}_0$ and $\hat{r}_0$ are from the solutions to Eq. (3.6) and Eq. (3.8) respectively and $\bar{P}$ is an approximation to P induced by sampling in Eq. (3.13). Therefore, we can re-write Eq. (3.20) as

$$g + \alpha\hat{P}\Phi_1\tilde{r} + \hat{\varepsilon}(\tilde{r}) - \Phi_1\tilde{r} = g + \alpha P\Phi_1\tilde{r} - \Phi_1\tilde{r} + \hat{\varepsilon}(\tilde{r}) + \alpha(\hat{P} - P)\Phi_1\tilde{r} \quad (3.21)$$

Then, if we let

$$\bar{s} = \max_{x\in S}\left[\frac{1}{\rho(x)}\left(\alpha(\hat{P} - P)\Phi_1\tilde{r} + \hat{\varepsilon}(\tilde{r})\right)(x),\ 0\right]$$

then $\rho\bar{s} \geq \hat{\varepsilon}(\tilde{r}) + \alpha(\hat{P} - P)\Phi_1\tilde{r}$, which means that

$$g + \alpha\hat{P}\hat{\Phi}_1\tilde{r} + \rho\bar{s} \geq \hat{\Phi}_1\tilde{r}$$

where $(\tilde{r}, \bar{s})$ is a feasible solution Eq. (3.15). Therefore,

$$c^T\hat{\Phi}_1\hat{r} - \eta\hat{s} \geq c^T\hat{\Phi}_1\tilde{r} - \eta\bar{s} \quad (3.22)$$

Therefore:

$$\begin{aligned}\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} &= c^T|J^* - \hat{\Phi}_1\hat{r}| \\ &= c^T|J^* - J_\varepsilon + J_\varepsilon + (I - \alpha\hat{P})^{-1}\rho\hat{s} - (I - \alpha\hat{P})^{-1}\rho\hat{s} - \hat{\Phi}_1\hat{r}|\end{aligned}$$

Since $J_\varepsilon + (I - \alpha\hat{P})^{-1}\rho\hat{s} - \hat{\Phi}_1\hat{r} \geq 0$ and $\hat{s} \geq 0$, then

$$\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} \leq c^T|J^* - J_\varepsilon| + c^T(J_\varepsilon + (I - \alpha\hat{P})^{-1}\rho\hat{s} - \hat{\Phi}_1\hat{r}) + c^T(I - \alpha\hat{P})^{-1}\rho\hat{s}$$

Next, let $\eta = 2c^T(I - \alpha\hat{P})^{-1}\rho$, then

$$\begin{aligned}\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} &\leq c^T|J^* - J_\varepsilon| + c^T(J_\varepsilon - \hat{\Phi}_1\hat{r}) + \eta\hat{s} \\ &= c^T|J^* - J_\varepsilon| + c^T(J_\varepsilon - J^* + J^* - \hat{\Phi}_1\hat{r}) + \eta\hat{s} \\ &\leq 2c^T|J^* - J_\varepsilon| + c^T(J^* - \hat{\Phi}_1\hat{r}) + \eta\hat{s}\end{aligned}$$

Then, using Eq. (3.22), we have

$$\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} \;\leq\; 2\|J^* - J_\varepsilon\|_{1,c} + c^T(J^* - \hat{\Phi}_1\tilde{r}) + \eta\bar{s}$$

Note that since $(I - \alpha\hat{P})^{-1} = \sum_{i=0}^{\infty}\alpha^i\hat{P}^i$, then every element $a_{ij} \in (I - \alpha\hat{P})^{-1}$ will satisfy $a_{ij} \geq 0$ since $\alpha \in [0,1]$ and $\hat{p}_{ij} \in \hat{P}$ such that $\hat{p}_{ij} \in [0,1]$. Therefore, since $\hat{\mu} = (1-\alpha)c^T(I - \alpha\hat{P})^{-1}$, then

$$\|J^* - J_\varepsilon\|_{1,c} \;=\; \frac{\alpha}{1-\alpha}\|(P - \hat{P})J^*\|_{1,\hat{\mu}} \tag{3.23}$$

This means that

$$\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} \;\leq\; \frac{2\alpha}{1-\alpha}\|(P - \hat{P})J^*\|_{1,\hat{\mu}} + c^T(J^* - \hat{\Phi}_1\tilde{r}) + \eta\bar{s}$$

Next, since

$$\hat{\Phi}_1\tilde{r} \;=\; \left[0 \mid (\bar{P} - P)\Phi(\hat{r}_0 - \tilde{r}_0)\right]\tilde{r} + \Phi_1\tilde{r}$$

then

$$\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} \;=\; \frac{2\alpha}{1-\alpha}\|(P - \hat{P})J^*\|_{1,\hat{\mu}} + c^T(J^* - \Phi_1\tilde{r}) + c^T\left[0 \mid (P - \bar{P})\Phi(\hat{r}_0 - \tilde{r}_0)\right]\tilde{r} + \eta\bar{s}$$

Therefore, if we let $\delta r_0 = \bar{1}\|\hat{r}_0 - \tilde{r}_0\|_\infty \in \mathbb{R}^K$, then

$$\|J^* - \hat{\Phi}_1\hat{r}\|_{1,c} \;=\; \|J^* - \Phi_1\tilde{r}\|_{1,c} + \frac{2\alpha}{1-\alpha}\|(P - \hat{P})J^*\|_{1,\hat{\mu}} + \|\left[0 \mid (P - \bar{P})\Phi\;\delta r_0\right]\tilde{r}\|_{1,c} + \eta\bar{s}$$

□

Chapter 4

The Multi-Vehicle Mission / Task Management Problem

From our discussion in Chapter 2, we showed that the UAV SWARM Resource Management Problem can be posed as an Markov Decision Process (MDP). Here, the state is defined as the vector (I, H, L) where I indicates the allocation of each agent in the task space, H indicates the maintenance / health state of each agent in the task space, and the symbol L denote a vector of aggregated task requests. The actions A are defined on the same space as I, where A indicates the new agent-mission group configuration. Note that only agents which have been "deallocated" from a task group can be re-allocated to a new task group. Here, de-allocated means that an agent is either (i) waiting at a ready location, (ii) available after maintenance, or (iii) removed from a task group which no longer needs its services or is not beneficial to the system. Next, a state (I, H, L) will transition under action A to state $(\bar{A}, \bar{H}, \bar{L})$ with probability $P_A((I, H, L), (\bar{A}, \bar{H}, \bar{L}))$. Finally, as stated above, the single-stage cost for being in state (I, H, L) is defined as $g(I, L)$ where we assume L remains constant over each time step.

In this chapter, we use the methods provided in Chapter 3 to implement and manage multi-vehicle mission problems. We present simulation results for a small-scale mission problem and compare the approximate cost-to-go function generated using these methods with the optimal cost-to-go for this problem. In addition, these techniques are used to find vehicle allocation policies for a large-scale mission problem and simulation results are provided for a variety of scenarios.

4.1 Implementation: Mission Management for a Simplified Persistent Surveillance Operation

As mentioned in Chapter 2, health monitors can be used by mission systems to improve their effectiveness over the duration of an operation. Monitors that evaluate mission performance, task group efficiency, team service rates and capability can be used to tune system parameters online, thereby providing a real-time evaluation of system capabilities. As shown in Figure 4-1, system health feedback combined with environmental data can be used to revise mission system strategies.

For example, health monitors are very important to systems managing persistent surveillance missions. Since a mission asset must be on-site for an extended period of time, cycling this asset with other system assets (for maintenance, fuel and other reasons) is critical to ensure that adequate coverage is provided over the area requested. Note that the simplified persistent surveillance mission (PSM) resource management problem can be posed as a Markov Decision Process (MDP) and formulated as an Approximate Linear Programming (ALP) problem that can be solved in real-time. In fact, the simplified PSM is a subproblem of the UAV SWARM Resource Management Problem where there is only one task location and one task type.

Here, we provide a simplified three vehicle PSM example. For this problem, the state $x \in S$ is defined as the vector $x = (z_1, z_2, z_3, h_1, h_2, h_3)$, where z_i indicates the task to which each agent is allocated, h_i indicates each agent's maintenance/health state, and S is the state space for the problem. Next, each action $a \in A_x$ is defined as the vector $a = (a_1, a_2, a_3)$ where a_i indicates the system's desired allocation for agent in the task space and A_x is the action space associated with the state $x \in S$. Each state x will transition under action a to the future state $y \in S$ with probability $P_a(x, y)$. Note that the agents in this problem can experience failures that cause them to be unavailable. In addition, agents are available for flight operations for a limited period of time (because of fuel, failure and maintenance concerns). Finally, a local cost for being in state x under action a is defined by the function $g_a(x) \in \mathbb{R}$.

Since this problem is posed as an MDP, an Approximate Linear Programming problem formulation can be used to find a feasible policy. The ALP formulation of the original Dynamic Programming problem is of the form

$$\begin{aligned} & \max_r \; c^T \Phi r \\ \text{s.t.} \quad & g_a(x) + \alpha \sum_{y \in S} P_a(x, y) \Phi(y) r \geq \Phi(x) r, \;\; \forall x \in S, \;\; a \in A_x \end{aligned} \tag{4.1}$$

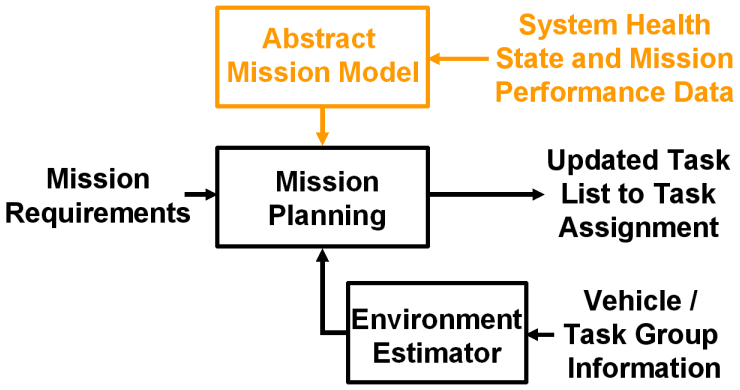


Figure 4-1: Updates to the mission planning subsystem in Figure 2-3 via health and performance data

such that a set of M basis functions $\phi_1, ...\phi_M$ where $M \ll |S|$ are used to approximate the cost-to-go function $J^* \approx \Phi r^*$. Here, the approximate solution to the original problem can be used to generate a policy that directs the vehicles to meet the overall system goals, while maintaining the overall health of the mission system.

However, as discussed in the previous chapter, one of the major obstacles faced when using an ALP formulation of the original problem is the selection of appropriate basis vectors and state relevance weights. In many cases, selecting the appropriate parameters to find Φ is based on experience. Therefore, by using the basis function generation algorithm shown in Table (3.1), the cost-to-go structure for this problem can be generated automatically for the underlying problem.

4.1.1 Results using Exact Probability Transition Matrices

As mentioned in the previous chapter, for small sized problems (problems by which the matrices can be stored) it is possible to use the exact probability transition matrices in determining the future cost-to-go for a state rather than simulated trajectories. For these problems, each successive basis function can be calculated and saved exactly using the method provided in Table (4.1). Note that in the formulation presented in Table (4.1) we are considering the entire problem, thereby not sampling future states nor sampling future trajectories, since the entire problem can be formulated and saved in system memory. Note that this method is not practical for use with problems where the size of the state space is large.

The goal of this first simulation is to compare the convergence rate of the cost-to-go approximation Φr using the basis function generation algorithm without sampling

Step 0: Start with $\Phi_0 = \bar{1}$, $k = 0$

Step 1: Calculate the best r_k, according to

$$\begin{array}{ll} \max\limits_{r_k} & c^T \Phi_k r_k \\ \text{subj. to} & g_a(x) + \alpha \sum_{y \in S} P_a(x,y)\Phi(y)r \geq \Phi(x)r \quad \forall x \in S, a \in A_x \end{array}$$

Step 2: Using the r_k, generate and save the basis function ϕ_{k+1} such that:

$$\phi_{(k+1)}(x) \quad = \quad \min_{a \in A_x} \left[g_a(x) + \alpha \sum_{y \in S} P_a(x,y)\Phi_k(y)r_k \right]$$

Step 3: After determining the new basis function ϕ_{k+1} and augment the matrix Φ_k so that $\Phi_{k+1} = [\Phi_k \quad \phi_{k+1}]$ and then let $k = k + 1$.

Step 4: Repeat Steps 1 through 4 until $\|T\Phi_k r_k - \Phi_k r_k\|_{1,c} \leq SC$ where SC is the predetermined stopping constant or $k = \bar{N}$ where $\bar{N}$ is the pre-determined number of basis functions to be calculated for the problem.

Table 4.1: Modification on the Basis Function Generation Algorithm using Exact Probability Transition Matrices (for small problems)

(shown in Table (4.1)) versus optimal cost-to-go function J^* found using policy iteration in MATLAB [91]. Figure 4-2 shows the convergence rate of the cost-to-go function to the optimal cost J^* as basis functions are generated. This problem has 1000 states and 3375 state-action pairs. Figure 4-2 (a) and (b) show the difference between the optimal cost-to-go J^* and $\Phi_k r_k$ as a basis function is computed and added at each iteration. For this test, the basis functions multipliers r_k were calculated using the Common Optimization INterface for Operations Research (COIN-OR) Open-Source Software package [63], which can be downloaded from `http://www.coin-or.org/`. Notice that the cost-to-go approximation $\Phi_k r_k$ approaches J^* very quickly after 7 basis functions are computed for this problem. In addition, the difference between J^* and the cost-to-go function J_k generated using value iteration is also shown in Figure (4-2). Even though the basis function generation algorithm uses the Bellman update to find the next basis function, value iteration takes over 100 iterations (starting from $J_0 = \Phi_0 r_0$ for comparison purposes) to reach the same cost as the method shown in Table (4.1) accomplishes in 13 iterations for the same problem. Therefore,

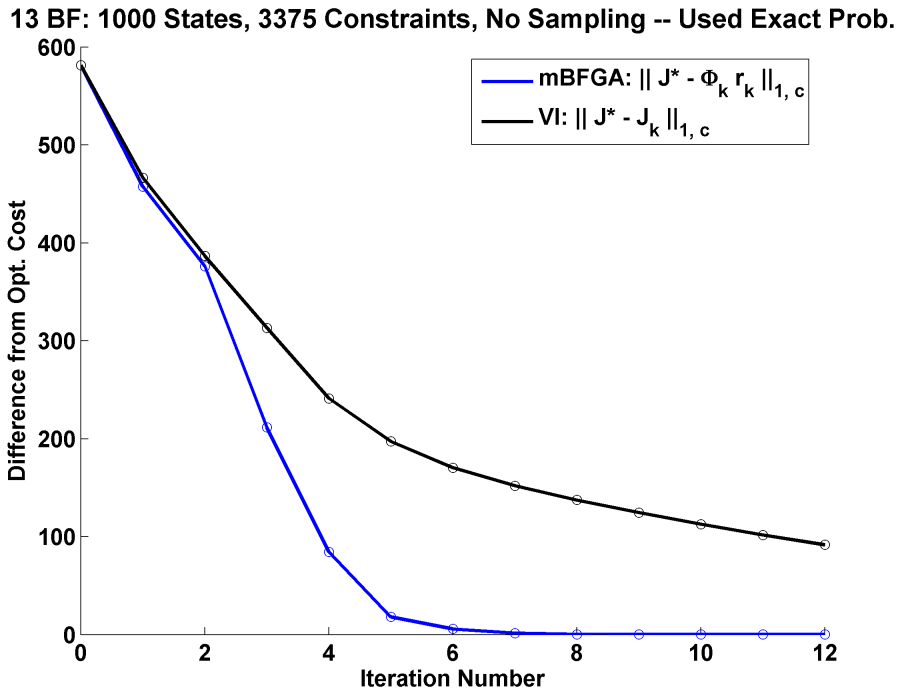
for small problems, the basis function generation algorithm without sampling (as shown in Table (4.1)) can be used to generate an approximation structure for the original problem that provides a good approximation of the optimal cost-to-go J^*.

4.1.2 Results using Basis Function Generation Algorithm

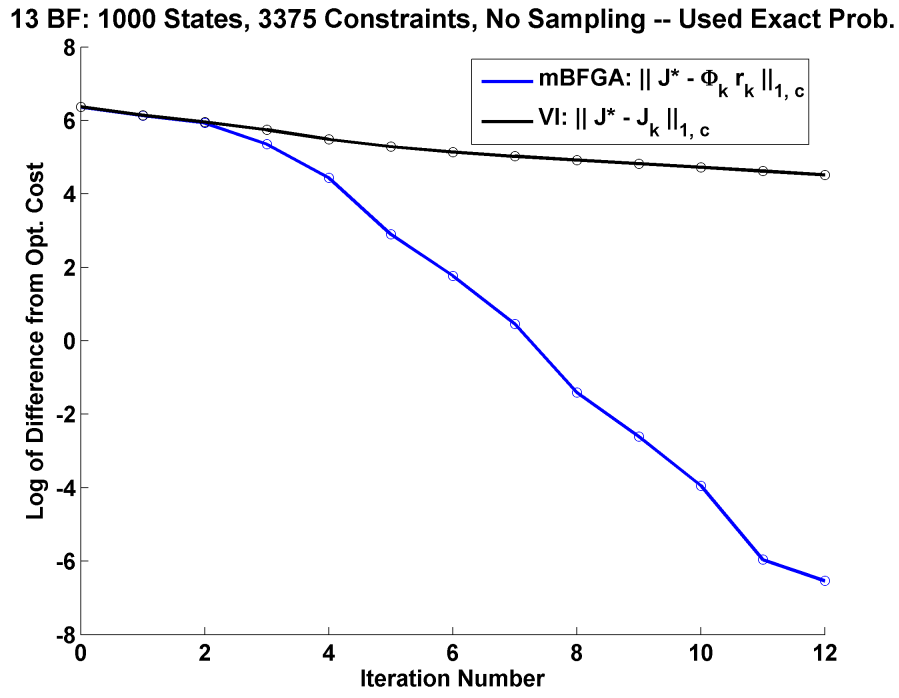
In most practical problems, however, it is not possible to use the exact probability transition matrices in determining the future cost-to-go for a state. For such problems, each successive basis function can be calculated using the method provided in Table (3.1). However, to analyze and compare the results of this method, a number of tests were generated to analyze the performance of the basis function generation algorithm shown in Table (3.1). Once again, the purpose of these simulations is to compare the convergence rate of the cost-to-go approximation Φr using the basis function generation algorithm (shown in Table (3.1)) versus optimal cost-to-go function J^* found using policy iteration in MATLAB [91]. Again, all of the basis functions multipliers r_k generated in this section for these simulation were calculated using the Common Optimization INterface for Operations Research (COIN-OR) Open-Source Software package [63], which can be downloaded from http://www.coin-or.org/.

Figure 4-3 shows the convergence rate of the cost-to-go function to the optimal cost J^* as basis functions are generated. Figure 4-3 (a) and (b) show the difference between the optimal cost-to-go J^* and $\Phi_k r_k$ as each basis function is computed and added at each iteration. Notice that the cost-to-go approximation $\Phi_k r_k$ improves quickly after after 10 basis functions are computed for this problem. However, after approximately 15 iterations, the cost-to-go approximation $\Phi_k r_k$ improvement begins to slow. This error between J^* and $\Phi_k r_k$ is a result of the sampling approximations used to estimate the future cost-to-go. Note that as more trajectories are sampled, the cost-to-go approximation improves. Finally, note that the "ripples" seen in the cost-to-go comparison are also a result of deviations in the sampled trajectories at each iteration. Since the basis functions are calculated using trajectories that are re-generated at each iteration, slight variations in the cost-to-go approximation structure result due to sampling differences from iteration to iteration.

In addition, note that the difference between J^* and the cost-to-go function J_k generated using value iteration (starting from $J_0 = \Phi_0 r_0$ for comparison purposes) is also provided in Figure 4-3 for comparison purposes. Therefore, the basis function generation algorithm (as shown in Table (3.1)) can be used to generate an approximation structure for the original problem that provides a good approximation of the



(a)



(b)

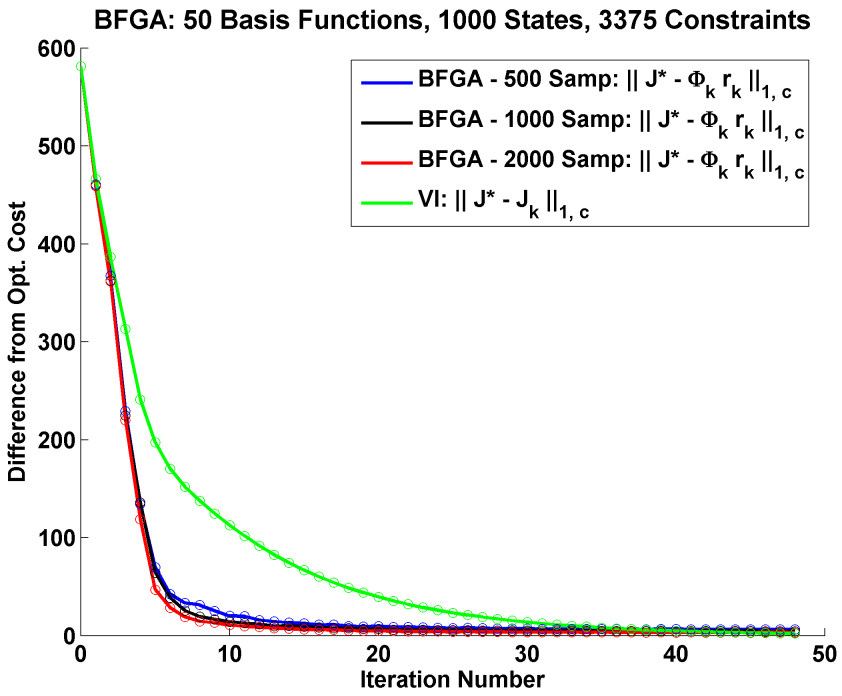
Figure 4-2: Cost-to-go Comparison for modified Basis Function Generation Algorithm (with no sampling) for 13 Basis Functions generated using the procedure shown in Table (4.1). (a) and (b) show the difference between the optimal cost-to-go J^* and $\Phi_k r_k$ as a basis function is computed and added at each iteration.

optimal cost-to-go J^*.

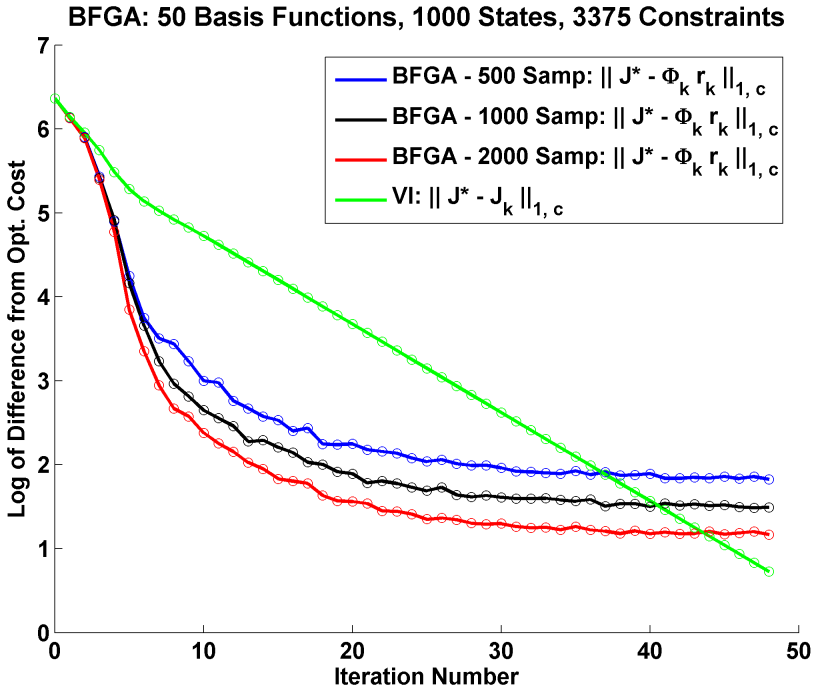
4.2 Implementation: Large-Scale Multi-Vehicle Mission Management Problem

Next, we consider a much larger problem mission management problem were the number of vehicles is large. To implement the UAV SWARM Resource Management problem formulation using the ALP-based approach (as described above), we must address a variety of issues related to the size of the state and action space. For example, since we have defined the state as the vector $x = (I, H, L) \in \mathcal{I} \times \mathcal{H} \times \mathcal{L}$ where $I \in \mathcal{I}$ indicates the allocation of each agent in the task space, $H \in \mathcal{H}$ indicates the maintenance / health state of each agent in the task space, and the symbol $L \in \mathcal{L}$ denote a vector of aggregated task requests, this means that this vector has $2M+NK$ elements. Here, $I = (I^1, ..., I^M)$ where $I^j \in \{-1, 0, ..., N\}$ such that "-1" represents the condition where the vehicle is either "en-route" to a task location or "waiting" at a forward operating location for assignment, "0" represents the maintenance or base location, and $\{1, ..., N\}$ represent the task locations each vehicle can be assigned to. Once again note that before a vehicle can be allocated to a task, it must transition through the "en-route to a task location" state. Next, $H = (H^1, ..., H^M)$ where $H^j \in \{0, ..., T_F\}$ is the current fuel condition of the vehicle and T_F represents the maximum period of time each vehicle can participate in the mission before returning to the maintenance location. Finally, $L = (L^1, ..., L^{NK})$ where $L^j \in \{0, 1, ..., L_{max}\}$ and represents the number of tasks that need to be completed by a task group in the mission system (where L_{max} is a large positive integer). Therefore, the size of the state space in this problem is very large, such that we will be unable to generate the probability transition matrix for each state in the problem exactly. In addition, since each action $A \in \mathcal{A} \equiv \mathcal{I}$, the action space is also very large, such that for given state we will be unable to examine and generate every possible action for evaluation in a period of time that would allow us to implement this for real-time use.

As a result, to be able to formulate and solve this problem we must use a technique called constraint sampling, as discussed in [23]. The main idea behind constraint sampling is that we can select a subset of the constraints from the original problem that will generate a good approximation of the cost-to-go function when all of the constraints are present. However, although constraint sampling reduces the number of state-action pairs that are considered in the problem formulation, calculating the



(a)



(b)

Figure 4-3: Cost-to-go Comparison for Basis Function Generation Algorithm for 50 Basis Functions generated using the procedure shown in Table (3.1). (a) and (b) show the difference between the optimal cost-to-go J^* and $\Phi_k r_k$ as a basis function is computed and added at each iteration.

future cost-to-go remains a formidable problem. As used in Step 1 of the basis function generation algorithm in Table (3.1), we use sampling to generate an estimate of the cost-to-go from a given state for a given action of the form

$$\sum_{y \in \mathcal{S}} P_a(x,y)\Phi^T(y)r \quad \approx \quad \frac{1}{W}\sum_{i=1}^{W} \Phi^T(y_i)r \tag{4.2}$$

where $\{y_1, ..., y_W\}$ are the set of sampled future states from $x \in S$.

4.2.1 Action Space Complexity

Although we now have a method to approximate the cost-to-go from each state, we still must address the fact that the problem's state and action space are both very large. As such, for a large problem, we may not be able to evaluate or find the best action to be used from this particular state (since the complexity of the action space is on the order of $\mathcal{O}(M^{N+1})$. Therefore, we must find a way to reduce the number of actions that are considered at each iteration so that a solution can be calculated in real-time. [23] provides many suggestions on how to deal with a system that has a large action space.

One such method that can be used is to determine the task group / maintenance allocation of each vehicle in turn. Note that at each time step, either each vehicle is allocated to a particular task (N mission tasks, the "waiting" or "enroute to a location" task and one maintenance task), or the vehicle has yet to be allocated to a task. A similar method was used with success to manage the web server farm system in [21]. For the UAV SWARM Resource Management problem, the action space could be pruned to consider following actions:

- For vehicles that are already allocated, there are three decisions: determine whether the vehicle will remain allocated to the same task, become re-allocated to another task, or return to the base for maintenance
- For vehicles that are not allocated, there are $N + 1$ possible allocations – each associated with a possible mission task group and the "waiting" or "enroute to a location" allocation

The main problem with this approach is that it will not reduce the complexity of the action space in states where many of the vehicles are not specifically allocated to

mission task groups. In this case, the action space complexity remains approximately $\mathcal{O}(M_{Deallocated}^{N+1})$ where $M_{Deallocated}$ is the number of vehicles that are deallocated.

Therefore, to reduce the size of the action space we are considering, we have selected an alternative approach that takes advantage of the problem structure and the information above to reduce the number of actions considered at any time step. First, we estimate how many vehicles will be required at each task location and order these needs based on the ratio $\frac{B_k}{\mu_k}$. Next, we determine if there are any vehicles that must be sent back to the base for maintenance due to flight time constraints, and preset the action associated with this vehicle to the maintenance task. Here, the number of vehicles being sent back for maintenance is defined as $M_{Maintaintance}$. Then, we use the scoring function Φr generated using basis function generation algorithm in Table (3.1) to decide on the total number of vehicles to be allocated (ranging from 0 to $M_{Deallocated}$) and deallocated (ranging from 0 to $max(M - M_{Deallocated} - M_{Maintaintance}, 0)$. Note that vehicles are allocated in each task following the task-based ordering (as mentioned above). This method dramatically reduces the number of actions that are considered from each state, allowing us to find an approximate solution and vehicle-mission task group allocation to the UAV SWARM Resource Management problem in a real-time.

4.2.2 Approximate Linear Programming-Based Mission Problem Implementation

Note that a key component of this formulation is the calculation / generation of basis functions. Since the entire UAV SWARM Resource Management problem cannot be solved in a short period of time where the size of the state space is large, we developed an approximation procedure to generate the constraints for this problem using the following method. First, we define the base policy for the UAV SWARM Resource Management problem as discussed in Table (4.2).

Using this base policy, we use the following approximation procedure to generate the constraints for this problem in order to generate the ALP used in Step 1 of the basis function generation algorithm:

1. Select Q possible future states from the system's current state x_0
2. For each future state $x^q \in S$ where $q \in \{1, ..., Q\}$, find the local cost for each state $g_a(x^q) = g(I^q, L^q)$. Note in this formulation, the local cost is calculated by

Step 1: Determine if there are any vehicles that must be sent back to the base for maintenance due to flight time constraints. If so, re-allocate these vehicle to the base location.

Step 2: Count the remaining number of vehicles assigned to each task location, and count the number of tasks that exist at each task location.

- If there are exactly enough vehicles to handle the current task load, then allocate the same vehicles to the current task.
- If there are more than enough vehicle to handle the task load, make the extra vehicles (with the most flying hours) available to be re-allocated to either assist other task groups (in need of additional resources)
- If there are not enough vehicles to handle the current task load, then allocate the same vehicles to the current task and request any available flying vehicles to handle the remaining task needs. Note that vehicles are allocated to requesting tasks based on the ratio $\frac{B_k}{\mu_k}$ to minimize the overall operation's local cost at any given stage.

Step 3: If there is still a need for additional vehicles (since the number of allocated vehicles cannot be used to meet current task goals), request more vehicles from the base location.

Step 4: If there are more enough vehicles available to cover the current task needs, deallocate the remaining extra vehicles and send them to base.

Table 4.2: Base Policy for the Large-Scale Mission Management problem)

assessing a small fuel penalty for each flying vehicle and assessing a penalty for each task that is not being served by a vehicle (where the vehicles are allocated in each task location based on the ratio $\frac{B_\cdot}{\mu_\cdot}$)

3. Generate W trajectories using the base policy defined in Table (4.2) to generate the cost-to-go structure Φ_k
4. Using the basis function generation algorithm in Table (3.1), find the multipliers used to generate basis functions online.

Note that this formulation can be used off-line and on-line to calculate new basis functions for this problem. However, once the basis function multipliers are generated, a very simple online procedure can be used to implement and evaluate the best policy on-line:

1. From the current state $x_t = (I_t, H_t, L_t)$, find the local cost for each state $g(I_t, L_t)$

2. For each possible action $a \in A_{x_t}$ (using the guidelines outlined in Section 4.2.1), estimate the future cost-to-go using the relationship in Equation (4.2). To do so, each action is used in selecting W future states $y_i \in S$ for $i \in \{1, ..., W\}$, from which W trajectories are generated using the base policy defined in Table (4.2) to generate the cost-to-go structure $\Phi(y_i)r$ (where r is the $\bar{N}$th multiplier vector from the basis function generation algorithm).

3. Finally, the best possible action is used as determined by:

$$\bar{a} \quad = \quad \arg\min_{a \in A_{x_t}} \left[g_a(x_t) + \frac{\alpha}{W} \sum_{i=1}^{W} \Phi(y_i) r \right]$$

This method provides a computationally efficient method for evaluating the best policy at each state in real-time.

4.2.3 Simulation Results

To evaluate this approach, we developed a mission simulator to examine the policies developed using this approach in real-time. For these simulation, we assumed that there were 40 vehicles, 5 task locations and 3 task types: classification tasks, search tasks, and surveillance tasks. These three task types were selected since the represent short-, medium- and long-duration tasks (respectively). During each operation, search and classification tasks may produce a follow-on surveillance task upon their completion. In addition, each vehicle has flight time limitations due to fuel and maintenance concerns. Likewise, vehicles can experience failures that cause them to return-to-base during the middle of a mission.

For these tests, we assumed that new search task requests occur with probability 0.33 per time step and have an average completion time of 3 time steps, new classification task requests occur with probability 0.25 per time step and have an average completion time of 1 time step, and new surveillance task requests occur with probability 0.16 per time step and have a maximum completion time of 10 time steps. In addition, search tasks produce a follow-on surveillance tasks with probability 0.5 and classification tasks produce a follow-on surveillance tasks with probability 0.25. Next, the costs were setup such that the ratio of cost benefit to average time of completion favored resolving search tasks first, classification tasks second, and surveillance tasks

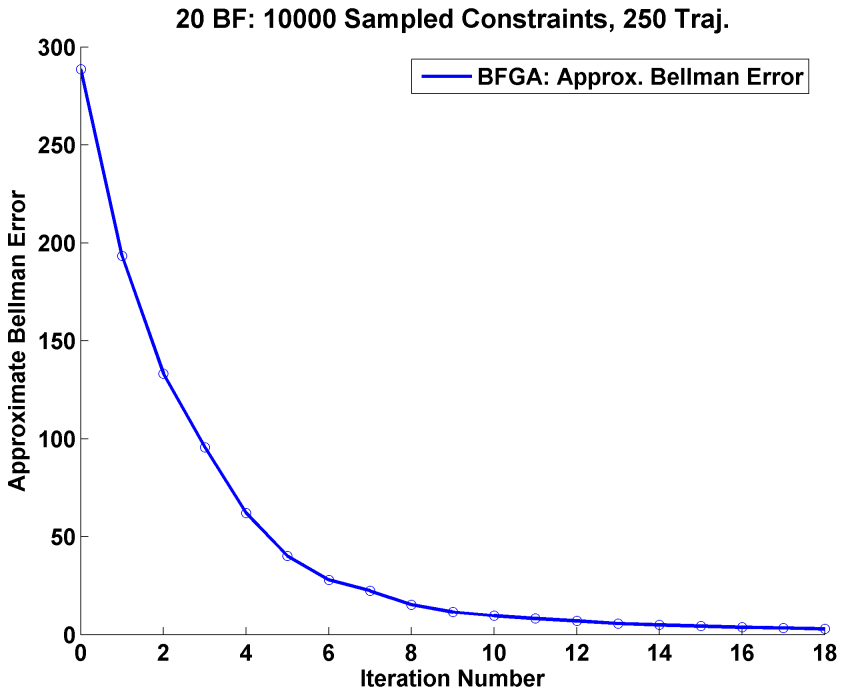


Figure 4-4: Approx. Bellman Error improvement Basis Function Generation Algorithm for Large Scale Problem for 20 Basis Functions generated using the procedure shown in Table (3.1) for the large-scale mission problem

third at all task locations. In addition, each vehicle has flight time limitations of 10 time steps. Finally, two different vehicle failure rates were tested in these simulations: vehicles could experience a failure with probability 0.1 in half the simulations, and in the second set of simulations, the vehicle failure rate was set to 0.

The basis function generation algorithm is used to generate the cost approximation structure for each problem. In Figure 4-4, shows the convergence rate of the approximate Bellman error function $\|\hat{T}\Phi_k r_k - \Phi_k r_k\|_{1,c}$ at each iteration for the case when the vehicle failure probability is 0.1 at each iteration. This plot shows that the approximate Bellman error decreases for sampled constraints. Once again, sampled trajectories are generated at each iteration to generate each basis functions at each iteration using their implicit representation (via the multipliers).

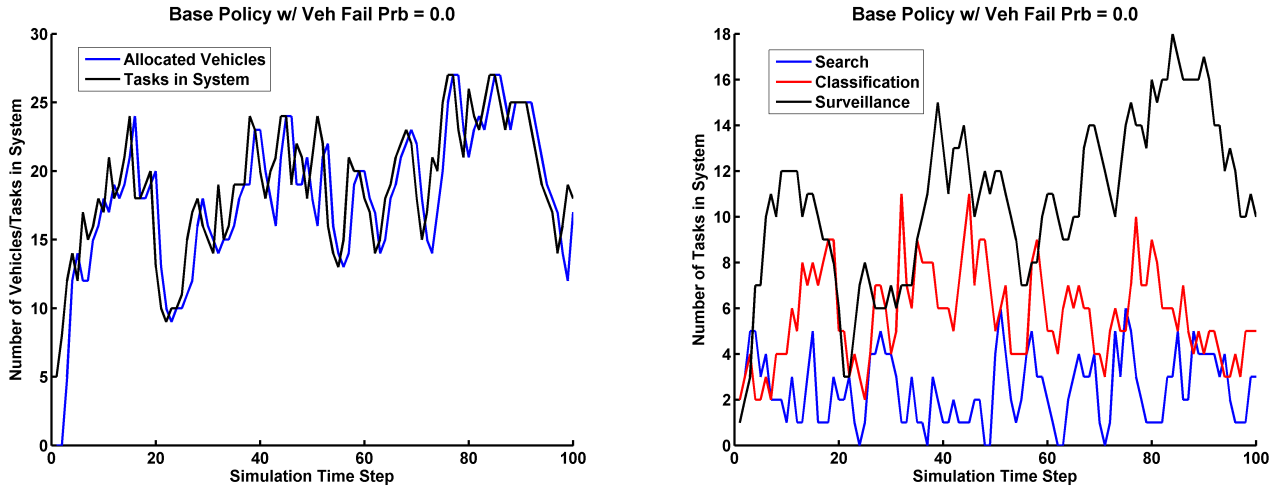
In the simulation tests, the base policy was compared against the policies generated using the method described at the end of Section 4.2.2. In Figure 4-5 shows simulation results for the base policy where the vehicle failure probability is 0.0 (a) and 0.1 (b) at each iteration. Note that the plots in the left column compare the number of vehicles allocated to tasks versus the number of task requests in the system

at each iteration. The plots in the right column show the number of each task type request for each iteration. First, note that the base policy allocates only the number of vehicles needed to perform the tasks in the system at each iteration. Therefore, this policy does not take into account future requests or task request loads on the system. In the case where the vehicle failure probability is set to 0 at each iteration, the base policy is able to maintain a number of vehicles in the system to complete the task requests filed by the system.

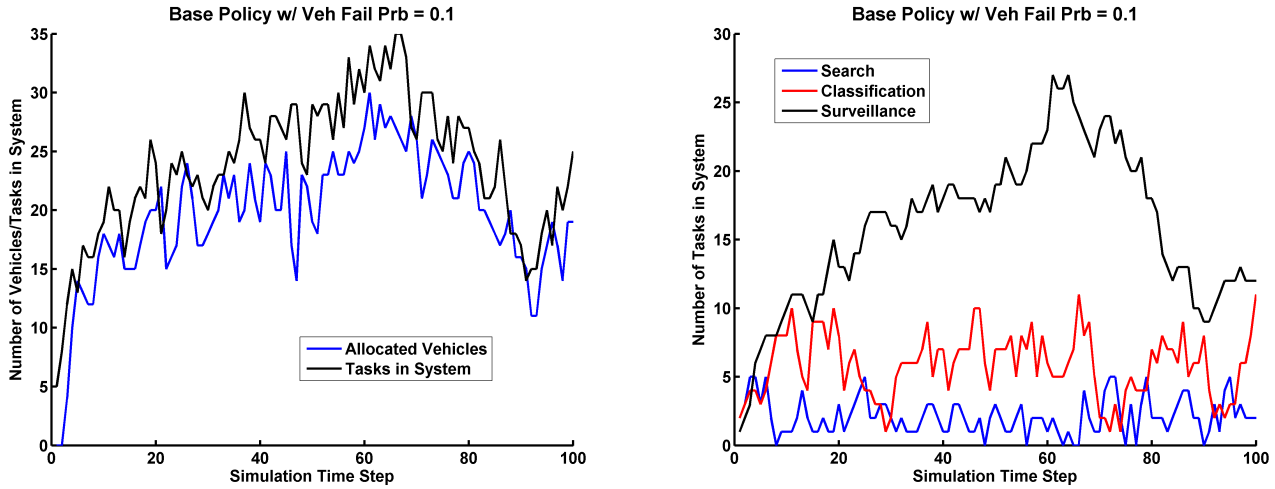
However, notice that in Figure 4-5(b) when the vehicle failure probability is set to 0.1 at each iteration, the system has trouble allocating enough vehicles to meet task demands. More specifically, note that surveillance tasks require vehicles to be on station much longer than other tasks and surveillance tasks are follow-on tasks to search and classification tasks. Therefore, the problem was setup so that surveillance tasks provided a smaller cost benefit to average time of completion ratio, which meant that surveillance tasks were the last tasks assigned in the system. Therefore, since the base policy allocated all of its vehicles to search and classification tasks to minimize the local cost and does not take into account vehicle failures or future task allocations, the mission system seldom had enough vehicles allocated to manage all of the tasks due to vehicle failures. In fact, at one point, the number of surveillance task requests in the system became extremely large because the base policy did not allocate enough vehicle resources to meet these task requests.

In contrast, Figure 4-6 shows simulation results for policies generated using 20 basis functions based on the method discussed at the end of Section 4.2.2 where the vehicle failure probability is 0.0 (a) and 0.1 (b) at each iteration. Again, the plots in the left column compare the number of vehicles allocated to tasks versus the number of task requests in the system at each iteration and the plots in the right column show the number of each task type request for each iteration.

While base policy allocates only the number of vehicles needed to perform the tasks in the system at each iteration, the cost function-based policies to seek keep more vehicles allocated on-station to account for future task requests and vehicle failures. Also notice that the policy calibrated for the vehicle failure probability of 0.1 appears to keep a more constant presence of vehicles allocated in the system than the policy associated with the system calibrated for the vehicle failure probability of 0. The main reason for this phenomenon is that since vehicles are failing and being sent back to base in the case where the vehicle failure probability of 0.1, the mission system attempts to ensure that vehicles are constantly being cycled to meet task demands while ensuring that there are enough resources available to service these tasks. This

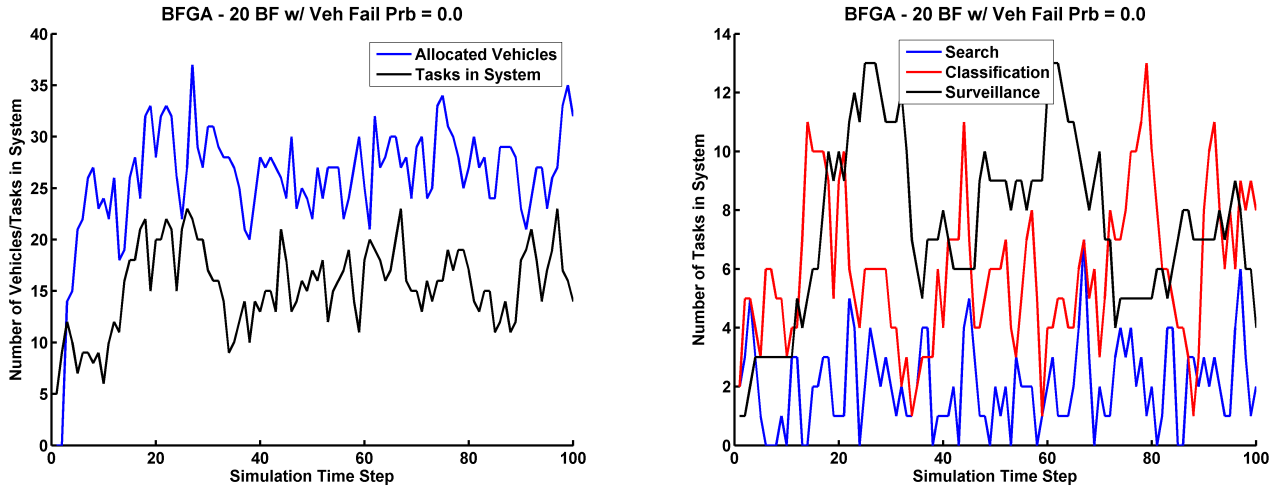


(a)

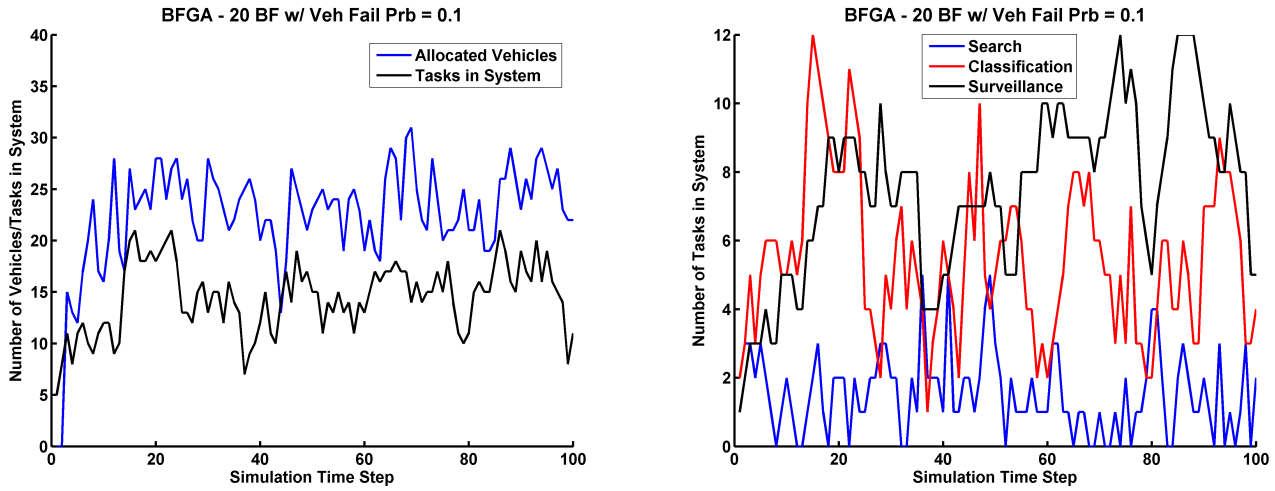


(b)

Figure 4-5: Simulation Results for the Base Policy where the vehicle failure probability is 0.0 (a) and 0.1 (b) at each iteration. The plots in the left column compare the number of vehicles allocated to tasks versus the number of task requests in the system at each iteration. The plots in the right column show the number of each task type request for each iteration.



(a)



(b)

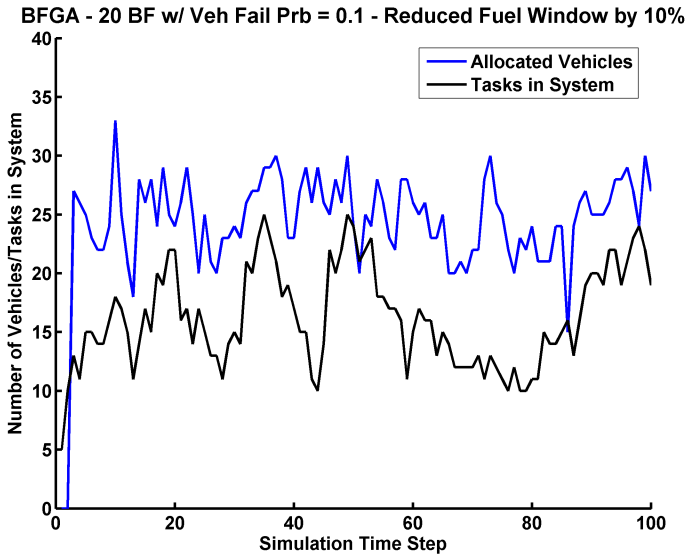
Figure 4-6: Simulation Results for BFGA Policies Generated using 20 Basis Functions where the vehicle failure probability is 0.0 (a) and 0.1 (b) at each iteration. The plots in the left column compare the number of vehicles allocated to tasks versus the number of task requests in the system at each iteration. The plots in the right column show the number of each task type request for each iteration.

also allows the mission system to allocated new vehicles directly to task areas that require more vehicle assets, thus reducing the number of vehicles switching between task areas. On the other hand, in the case where the vehicle failure probability is 0, the mission system's main concern is to balance the vehicle's fuel constraints against system needs. Therefore, vehicles remain in the task area as long as they are needed before getting sent back to base. This means that vehicles may be allocated to task regions that do not require vehicles at that moment in order to ensure that there are enough vehicles to serve future task requests. Regardless, both policies successfully keep the number of task requests down and perform better than the base policy.

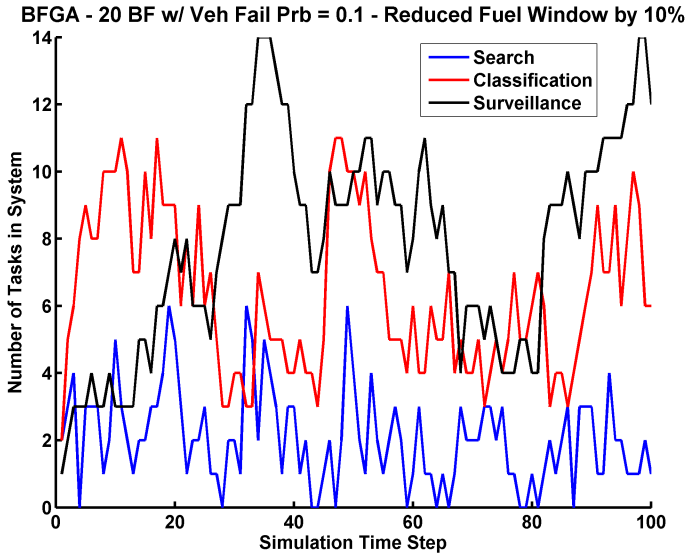
The next question one may ask is how does the policy change if the same basis functions are used when changes are made to the underlying problem. For example, assume that the vehicle's fuel window is limited by 10% because of fuel shortages or due to the distance a vehicle is traveling. In Figure 4-7 shows simulation results for policies generated using 20 basis functions based on the method discussed at the end of Section 4.2.2 where the vehicle failure probability is 0.1 at each iteration and the vehicle's fuel window in the simulation is reduced by 10%. Note that basis function parameters were calculated based on a fuel window of 10 time steps, while the simulation reduces the vehicle fuel window to 9 time steps. Despite this change, the mission manager is able to manage the number of tasks in the system. Note that although there are more task requests in the system, the mission system is still able ensure that there are enough vehicles allocated to meet the task requests.

Finally, Figure 4-8 shows simulation results for policies generated using 20 basis functions based on the method discussed at the end of Section 4.2.2 where the vehicle failure probability is 0.0 (a) and 0.1 (b) at each iteration, except the fuel cost per vehicle is increased by 250%. Each plot on this page compares the number of vehicles allocated to tasks versus the number of task requests in the system at each iteration. The plots in the left column are for the original fuel cost used in the other experiments (as shown in Figure 4-6) and the plots in the right column show the results where the fuel cost is increased by 250%.

Once again, the cost function-based policies seek keep more vehicles allocated on-station to account for future task requests and vehicle failures; however, notice that as the fuel cost per flying vehicle is increased, the gap between the number of vehicles allocated and the number of tasks in the system is smaller (as shown by comparing the plots in the left column with the original fuel cost and the plots in the right column with the increased fuel cost). In addition, these plots also show that there is a balance between the performance of the policy and the fuel cost of the

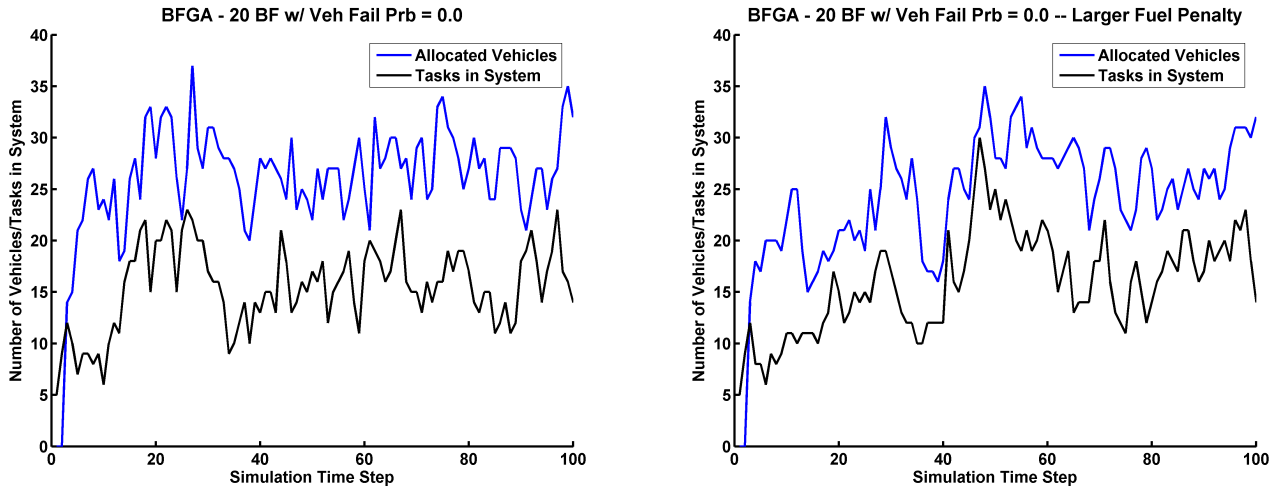


(a)

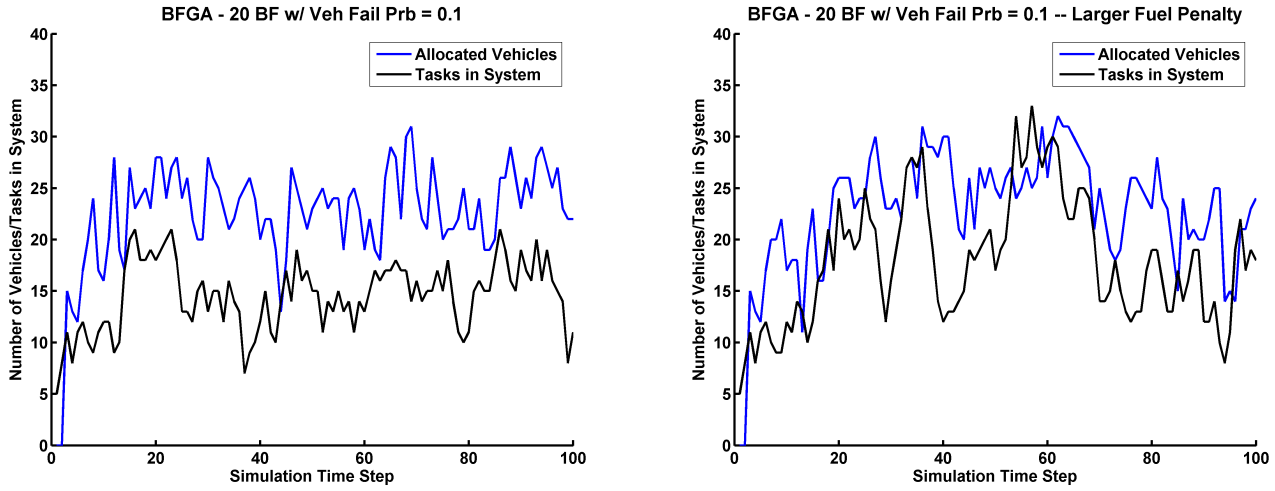


(b)

Figure 4-7: Simulation Results for BFGA Policies Generated using 20 Basis Functions where the vehicle failure probability is 0.1 at each iteration and the fuel window in the simulation is reduced by 10%. (a) compares the number of vehicles allocated to tasks versus the number of task requests in the system at each iteration, while (b) shows the number of each task type request for each iteration.



(a)



(b)

Figure 4-8: Simulation Results for BFGA Policies Generated using 20 Basis Functions where the vehicle failure probability is 0.0 (a) and 0.1 (b) at each iteration and the fuel cost per flying vehicle is increased by 250% in the simulation. Each plot on this page compares the number of vehicles allocated to tasks versus the number of task requests in the system at each iteration. The plots in the left column are the original fuel cost used in the other experiments (as shown in Figure 4-6) and the plots in the right column show the results where the fuel cost is increased by 250%.

vehicles. Note that when there is a rapid increase in the number task requests, the policy associated with the basis functions calibrated for the vehicle failure probability of 0.1 has less vehicles allocated to account for the task requests in the system (see the lower right plot in Figure 4-8 near the 60th time step). As mentioned before, though vehicles are failing and being sent back to base in the case where the vehicle failure probability of 0.1, the mission system attempts to ensure that vehicles are constantly being cycled to meet task demands while ensuring that there are enough resources available to service these tasks. However, due to the raised fuel cost, the mission system is slow to allocate vehicle assets to the task area to ensure that there are an adequate number of resources to serve the tasks requests while balancing fuel costs by limiting the number of allocated vehicles. Regardless, even when there is a rapid increase in task requests, over time the system is able to reduce these task loads by allocating more vehicles to account for these periods of large demand. Therefore, both policies successfully keep the number of task requests down and perform better than the base policy even with the increase in the fuel cost.

4.3 Summary

In summary, the basis function generation methods posed in Chapter 3 can be used to find effective policies for both small- and large-scale problems. These tests also demonstrate this method can be used in real-time to manage mission system tasks autonomously. As we will see in Chapter 6, the basis function generation method is the only method in the literature that has been used a real-time mission management system. The flight tests in Section 6.5.1 and the simulation results presented in this chapter mark a large step in the development of on-line cost approximation structures used to manage autonomous systems in real-time.

Chapter 5

Multi-Agent Mission System Testbed and Health Management

As mentioned in Chapter 1, unmanned aerial vehicles (UAVs) are becoming vital warfare and homeland security platforms because they significantly reduce costs and the risk to human life while amplifying warfighter and first-responder capabilities. These vehicles have been used a number of military and civilian applications with success, but there remains a formidable barrier to achieving the future vision of multiple UAVs operating cooperatively with other manned and unmanned vehicles in the national airspace and beyond. In addition, there is very little in the literature to date about how to perform multi-day autonomous system operations. In extended mission operations, autonomous mission systems must use health management techniques to evaluate and assess the capability of system components when creating an effective strategy to meet task and mission goals. In prior work, the term *health management* is used to define systems that actively monitor and manage vehicle components (for example actuators, flight control, engine, avionics and fuel management hardware) in the event of failures [5, 30].

To investigate and develop health management techniques for autonomous multi-agent mission platforms, an indoor multi-vehicle testbed called RAVEN (Real-time indoor Autonomous Vehicle test ENvironment) was developed to study long-duration missions in a controlled environment. Normally, multiple human operators are needed to manage flight hardware, navigation, control, and vehicle tasking during multi-vehicle coordination and control demonstrations. However, the RAVEN allows researchers to focus on high-level tasks by autonomously managing the navigation, control, and tasking operations of the platform's realistic air and ground vehicles

during multi-vehicle operations. This characteristic promotes the *rapid prototyping* of UAV technologies by means of flight testing new vehicle configurations and algorithms without redesigning vehicle hardware.

As a result, RAVEN is being used to implement and analyze techniques for embedding the fleet and vehicle health state into UAV mission planning. In particular, using RAVEN we are examining key research questions related to vehicle and multi-agent health management such as vehicle failures, refueling, and maintenance using real hardware. RAVEN is comprised of both aerial and ground vehicles, allowing researchers to conduct tests for a wide variety of mission scenarios. This chapter describes the components and architecture of RAVEN and presents recent flight test results.

5.1 Background

As mentioned in Chapter 1, a variety of research platforms have been developed to study advanced theories and approaches in the development of innovative UAV concepts [2, 19, 26, 27, 33, 46, 47, 49, 52, 50, 53, 54, 56, 58, 70, 85, 90, 103, 104]. However, these testbeds have several limitations that inhibit their utility for investigating health management questions related to multi-day, multi-agent mission operations. For example, outdoor platforms can be tested only during good weather and environmental conditions. Since most outdoor UAV test platforms can be flown safely only during daylight operations, these systems cannot be used to examine research questions related to long-duration missions, which may need to run overnight. In addition, many of these vehicles are modified to carry additional vehicle hardware for flight operations. As a result, these vehicles have to be redesigned to meet payload, onboard sensing, power plant, and other requirements. Thus, these vehicles must be flown in specific environmental conditions, unrelated to flight hour constraints, to avoid damage to the vehicle hardware. These external UAVs also typically require a large support team, which makes long-term testing logistically difficult and expensive.

In contrast, RAVEN is designed to test and examine a wide variety of multi-vehicle missions using both autonomous ground and air vehicles. Since the platform uses small, essentially unmodified electric helicopters and airplanes, we can fly more than five air vehicles in a typical-sized room at the same time. In fact, one operator can set up the platform for flight testing multiple UAVs in under 20 minutes. As a result, researchers can perform a large number of test flights in a short period of time with little logistical overhead.

At the heart of the testbed is a global metrology system that yields accurate, high bandwidth position and attitude data for all vehicles in the entire room. Since the position markers are lightweight, the position system is able to sense vehicle position and attitude without adding any significant payload to the vehicles. As a result, RAVEN's configuration does not require significant modifications to off-the-shelf radio-controlled (R/C) vehicle hardware. This configuration enables researchers to avoid being overly conservative in flight testing. Thus, this platform is ideal for the rapid prototyping of multi-vehicle mission management algorithms since one person can operate the system over long periods of time at a fraction of the cost needed to support multi-day external flight demonstrations.

5.2 System Architecture and Components

One objective of RAVEN is to enable researchers to test a variety of algorithms and technologies applicable to multi-UAV mission problems in real time. Therefore, the architecture must allow users the flexibility to add and remove both hardware and software components as needed. Another design objective is to provide one operator with the capability to command several autonomous vehicles at the same time. Consequently, the architecture must include components that provide the mission operator with sufficient situational awareness to verify or issue changes to a vehicle's plan in real time.

To meet these requirements, RAVEN's architecture has the hierarchical design shown in Figure 5-1(a). This architecture is used since it separates the mission and task components of the architecture from the testbed's vehicle infrastructure. Thus, changes in the mission and task components of the system do not require changes to the testbed's core software infrastructure and can be made in real time. This approach builds on an earlier version of the system architecture used in the DARPA-sponsored Software Enabled Control capstone flight demonstration by the MIT team [82, 100]. As discussed in [100], this architecture was used to enable the Weapons Systems Officer on-board an F-15E fighter aircraft to successfully command a UAV during a mission in real-time using a Natural Language interpreter. During flight testing, multiple successful F-15/T-33 sorties were flown successfully using this mission software on three separate flights. It was the first time a manned aircraft controlled a UAV via natural language in real-time in flight.

The planning part of the RAVEN's architecture has four major components, namely, a mission planning level designed to set the system goals and monitor system

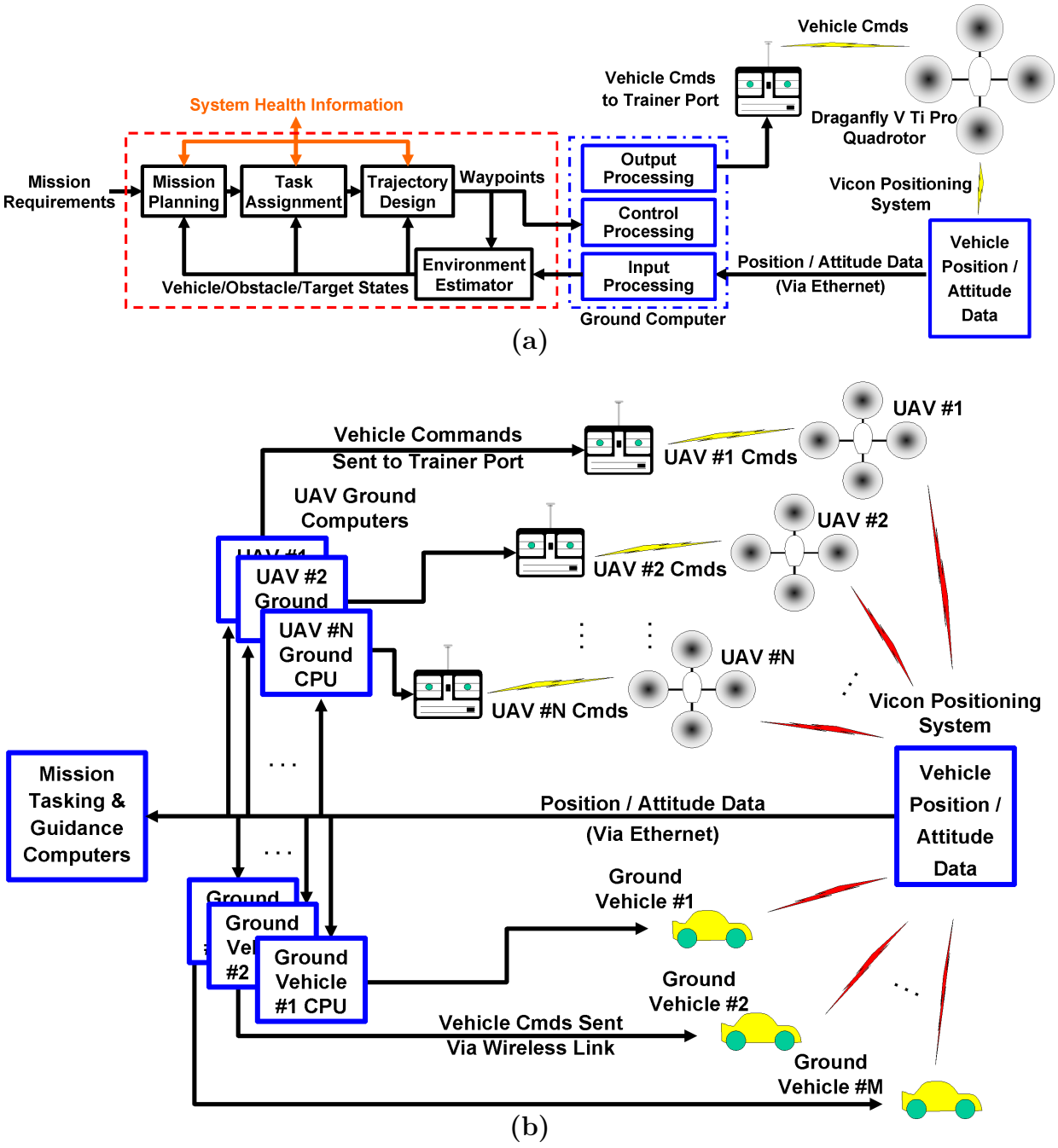


Figure 5-1: (a) Multi-vehicle command and control architecture block diagram and (b) the integrated vehicle system block diagram. This system architecture is designed to separate the mission- and task-related components of the architecture from RAVEN's vehicle infrastructure. Therefore, adjustments in the mission, task, and trajectory planning components of the system do not require changes to the testbed's core software infrastructure and can be made in real time.

progress, a task assignment level that is designed to issue and assign specific tasks to a vehicle or vehicle group to support the overall mission goals, a trajectory design level that directs each vehicle and its subsystems on how to best perform the actual tasks provided by the task processing level, and a control processing level designed to carry out the activities set by higher levels in the system. In addition, health information about each component in the system is used by each component in the architecture to make informed decisions on the capabilities of each subsystem in the architecture. As a result, each component is designed to support the decisions made in each level to ensure that vehicles make an informed decision that is in the team's best interest for any given task.

The architecture used in RAVEN allows researchers to rapidly interchange different mission system components for the purpose of testing a variety of algorithms in a real-time environment. For example, in order to allow users to rapidly prototype mission, task, and other vehicle planning algorithms with RAVEN's vehicle hardware, the vehicles must be able to accept command inputs, such as waypoints, from any high level planning system. Although low-level commands like "fly at a set speed" can be issued to the vehicles in the platform, a waypoint interface to the vehicles allows users to easily substitute mission, task, and path planning components into the architecture without changing the vehicle's base capabilities. This interface allows users to add, remove, and test algorithms and other related components as needed. In fact, the waypoint interface to the vehicles has already allowed users to develop and implement code on the platform in real time using programs like Matlab to test centralized and distributed planning algorithms using computers from other locations on campus. As a result, researchers can implement various control, navigation, vehicle tasking, health, and mission management algorithms on the system [11, 20, 59, 94, 96, 97]. Likewise, users can easily incorporate new vehicles into the testbed architecture. New vehicle controllers and base capabilities can be added, removed, and tested in the architecture without affecting the rest of the system components.

5.3 Main Testbed Hardware

In order to test and demonstrate the real-time capabilities of health management algorithms in a realistic real-time environment, we sought to develop a testbed that uses simple, robust vehicles in an indoor test environment that can be flown for extended periods of time. As a result, RAVEN's hardware architecture is designed to allow researchers to use a variety of R/C vehicles without requiring significant

modifications. Figure 5-1(b) shows a diagram of the components and setup of the integrated control system. Since the platform's primary computing, data collection, and sensing resources are offboard the vehicles, users can examine research questions related to autonomous multi-vehicle operations using simple, inexpensive vehicles.

Currently, the control processing and command data for each vehicle is processed by a dedicated computer and sent over a USB connection from the vehicle's control computer to the trainer port interface on the vehicle's R/C transmitter. All computing for this system is performed on ground-based computers, which have two AMD 64-bit Opteron processors, 2 Gb of memory, and run Gentoo Linux.

A Vicon MX camera system measure the position and attitude for each vehicle in the testbed [102]. This system yields accurate, high bandwidth position and attitude data in real time for all vehicles in the testing area. By attaching lightweight reflective markers to the vehicle's structure, the Vicon MX camera system and Tarsus software can track and compute the vehicle's position and attitude for each vehicle in the flight space at rates up to 120 Hz. This sensing data is processed by a central computer, and then broadcast over a high-speed network for any system component to use as needed. This motion capture system provides a simple, baseline capability for sensing and controlling the vehicle motion, which enables researchers to explore research topics, such as multi-vehicle coordination, vision-based navigation and control, or new propulsion mechanisms such as flapping flight. Just as GPS spurred the development of large-scale UAVs, we expect this new sensing capability to have a significant impact on indoor flight, which has historically been restricted to constrained 3D regions.

The accuracy of the Vicon MX camera systems's position and attitude measurements are difficult to confirm during flight operations since the vehicles are prone to external disturbances. However, Figure 5-2 shows a scatter plot of the measured (x,y) position (in meters) of a quadrotor sitting on the floor at position (0,0). Note the scale on the plot – with the rotors not turning, the maximum x-position error measured by the system in this test is 0.325 mm and the maximum y-position error measured by the system in this test is 0.199 mm. Tracking multiple reflectors in a unique orientation on each vehicle enables the Vicon camera system to determine the position of the center of mass and the attitude of each air/ground vehicle that is within range. For example, an eighteen camera configuration can easily track five air vehicles and multiple ground vehicles in a 8-m by 5-m by 3-m flight volume.

Currently, RAVEN is comprised of a variety of rotary-wing, fixed-wing, and ground-based R/C vehicles types. However, most testbed flight experiments are performed using the Draganflyer V Ti Pro quadrotor [24]. This quadrotor model is a

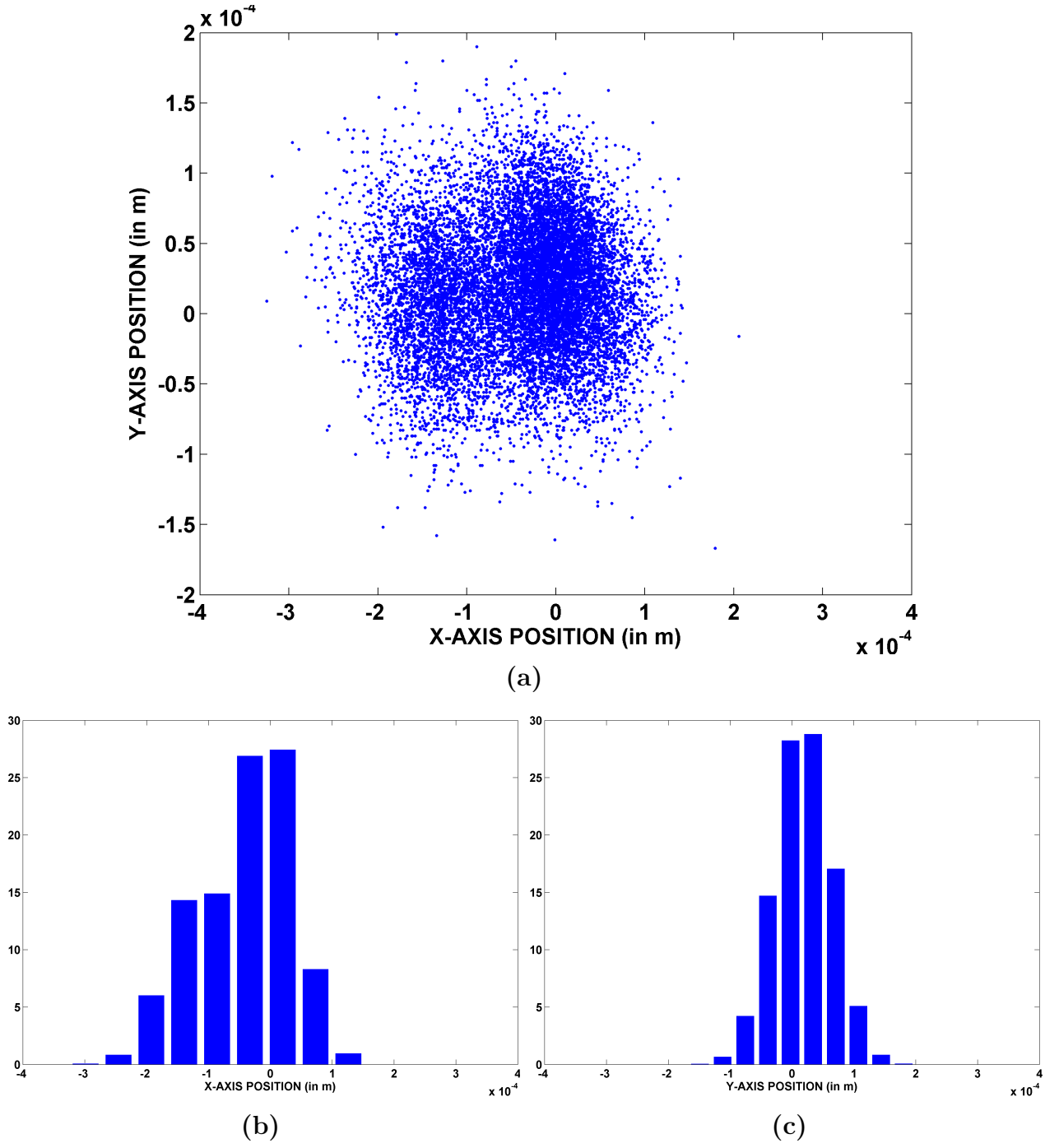


Figure 5-2: Scatter plot of (x, y) vehicle position. (a) shows a scatter plot of the measured (x,y) position (in meters) of a quadrotor sitting on the floor at position (0,0). (b) and (c) show histograms of the measured percentage of time of the vehicle's measured x- (b), and y-positions (c). Note that the scale in these plots is meters x 10^{-4} – with the rotors not turning, the maximum x-position error measured by the system in this test is 0.325 mm and the maximum y-position error measured by the system in this test is 0.199 mm.

small ($\approx$ 0.7 m from blade tip to blade tip), lightweight (under 500 g) air vehicle with a payload capacity of about 100 g that can fly between 13–17 minutes on one battery charge (using a 2000 mAh battery) while carrying a small camera. The four-propeller design simplifies the dynamics and control, and the vehicle's airframe is robust and easy to repair in the event of a crash. The rotor blades are designed to fracture or bend when they hit a solid object. These qualities make the Draganflyer V Ti Pro quadrotor durable and safe for indoor flight.

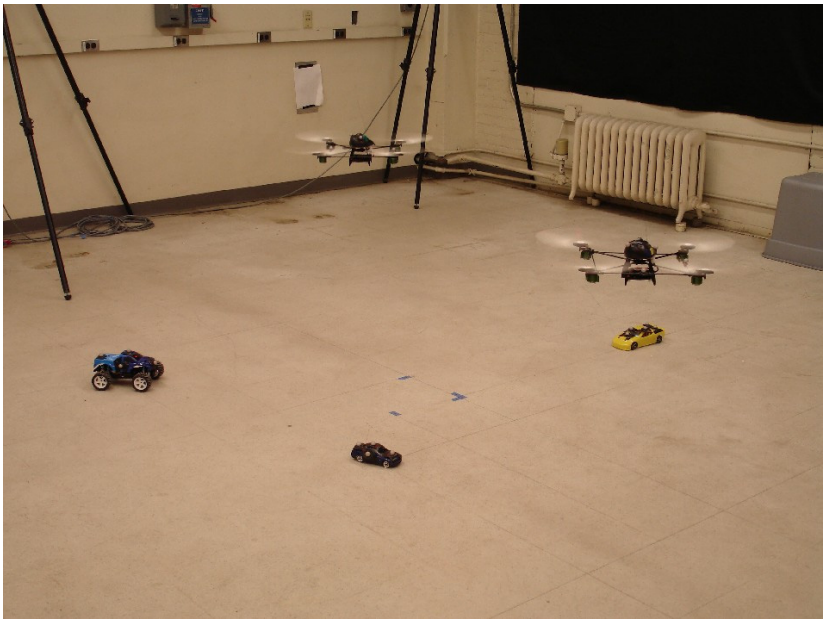
A separate landing and ground maintenance system are used to support the quadrotor vehicle hardware in an around-the-clock environment. More specifically, the landing hardware and its associated real-time processing aids the vehicle's guidance and control logic during the takeoff and landing tasks. In addition, a maintenance module has been developed to evaluate whether the actual vehicles are due for maintenance and monitor the recharging of the batteries prior to flight. This concept has been successfully demonstrated several times. Currently, a new version of the quadrotor recharge platform is being tested prior to its integration into the testbed.

Likewise, R/C cars and trucks are being used as ground vehicles in the platform. Most of these ground vehicles are modified R/C trucks made by DuraTrax [44]. These modifications consist of replacing the stock onboard motor controller and R/C receiver with a custom motor driver circuit, Robostix microcontroller [37], and RF communication module with 802.15.4 wireless connectivity. These modifications are made to improve the vehicle's precision driving capabilities while making it possible to autonomously command and control multiple ground vehicles by means of one ground computer in mission scenarios, such as airborne search and track missions, search and recovery operations, networking experiments, and air and ground cooperative mission scenarios.

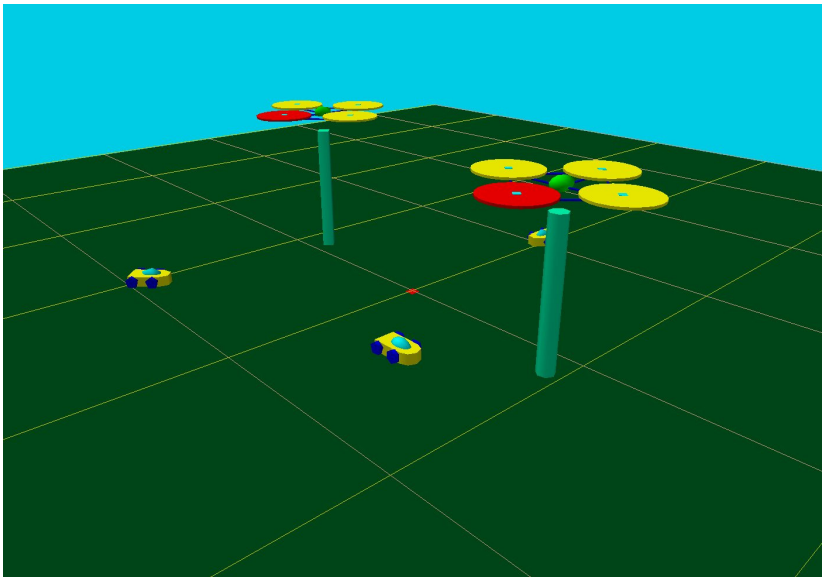
In addition to the quadrotors, RAVEN offers a unique indoor environment for conducting dynamic flight control experiments. For example, foam R/C aircraft are being used to explore the properties of an aircraft flying in a prop-hang (that is, nose-up) for the purposes of landing vertically and performing other complex maneuvers.

5.3.1 Task Processing and Operator Interface Components

The control system for each vehicle in the testbed can process and implement tasks defined by a system component or user. For example, each vehicle has a vehicle manager module designed to handle task processing, trajectory generation, and control processing for the vehicle. This module is designed to allow an external system or



(a)



(b)

Figure 5-3: Multi-vehicle search and track experiment (a) and operator interface visualization (b). The sensing system records the ground vehicle locations in real time. When the vehicles are being tracked by the UAVs, then the location is displayed to the operator

user to communicate with the vehicle using task-level commands, such as, "fly to waypoint A", "hover/loiter over area B", "search region C", "classify/assess Object D", "track/follow object E", and "takeoff/land at location F". These commands can be sent to the vehicle at any time during vehicle operations. Each agent's vehicle manager processes these tasks as they arrive and responds to the sender acknowledging the task request.

RAVEN is also designed with an automated system task manager. Since each air vehicle in the system can takeoff and land autonomously, this task manager can autonomously manage any air and ground vehicle controlled by the system using task-level commands. As a result, multi-vehicle mission scenarios (for instance, search, persistent surveillance, area denial, and others) can be organized and implemented by the task manager autonomously. Likewise, coordinated multi-vehicle flight tasks can also be managed by the task advisor with little operator interaction with the system, thus allowing one operator to command and control multiple vehicles at the same time.

Although this system reduces the operator load by handling many tasks autonomously, the system has an operator interface with vehicle tasking capability. The task manager system is designed to allow an operator to issue a command to any vehicle at any time. Currently, the operator interface includes a 3D display of the objects in the testing area, as shown in Figure 5-3, and a command and control user interface, which displays vehicle health and state data, task information, and other mission-relevant data.

The vehicle trajectory is specified by the planner as a sequence of waypoints consisting of a location $\bar{x}_i = (x, y, z)$, vehicle heading ψ_i, and speed v_i. Given these waypoints, several options are available for selecting the actual reference inputs to the vehicle, with perhaps the simplest smooth path being to follow a linear interpolation of the points defined by

$$\bar{x}_{ref}(t) \quad = \quad \frac{(\bar{x}_{i+1} - \bar{x}_i)}{|\bar{x}_{i+1} - \bar{x}_i|} v_i t, \tag{5.1}$$

where the choice of v_i can be used to move between waypoints at varying speeds. This same approach is used to automate the takeoff and landing procedure for the quadrotor vehicles. As a result, the quadrotor vehicles are fully autonomous from takeoff to landing during all flight operations.

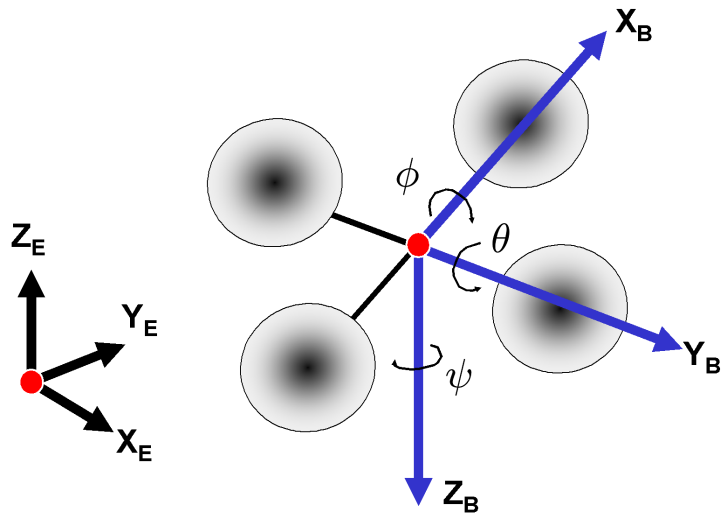


Figure 5-4: Quadrotor model axis definition

5.4 Quadrotor Control Design Model

A model of the quadrotor vehicle dynamics is needed to design a hover controller. Figure 5-4 shows the definition of the inertial frame (x_E, y_E, z_E), which is used to specify the vehicle's position, and the Euler angles ϕ, θ, and ψ, which are used to specify the vehicle orientation (the roll, pitch, and yaw, respectively). The body frame is specified by the x_B-, y_B-, and z_B-axes, the body moment of inertias are I_x, I_y, and I_z, and p, q, and r denote the angular rates in the body frame. Additional parameters include the distance from the vehicle center-of-mass and a motor L, the mass of the vehicle m, the moment of inertia of a rotor blade J_R, and a disturbance generated by differences in rotor speed d. The inputs to the system are $\delta_{\text{collective}}$, δ_{roll}, δ_{pitch}, and δ_{yaw}, which are the collective, roll, pitch and yaw input commands, respectively. Starting with the flat earth, body axis six-degree-of-freedom (6DOF) equations [89], the kinematic and moment equations for the nonlinear quadrotor model can be written as

$$\dot{p} = q\,r\left(\frac{I_y - I_z}{I_x}\right) - \frac{J_R}{I_x}\,q\,d + \frac{L}{I_x}\delta_{\text{roll}}, \tag{5.2}$$

$$\dot{q} = p\,r\left(\frac{I_z - I_x}{I_y}\right) + \frac{J_R}{I_y}\,r\,d + \frac{L}{I_y}\delta_{\text{pitch}}, \tag{5.3}$$

$$\dot{r} = p\,q\left(\frac{I_x - I_y}{I_z}\right) + \frac{1}{I_z}\delta_{\text{yaw}}, \tag{5.4}$$

$$\dot{\phi} = p + \tan\theta\ (\ q\ \sin\phi\ +\ r\ \cos\phi\ )\,, \tag{5.5}$$
$$\dot{\theta} = q\ \cos\phi\ -\ r\ \sin\phi, \tag{5.6}$$
$$\dot{\psi} = (\ q\ \sin\phi\ +\ r\ \cos\phi\ )\sec\theta, \tag{5.7}$$

where the terms $-\frac{J_R}{I_x}\ q\ d$ and $-\frac{J_R}{I_y}\ p\ d$ are included here to represent the disturbances caused by changing rotor speeds (as discussed in [12]), although the gyroscopic effects for the Draganflyer [24] model are small. Also, the cross-coupled inertia terms in this description have been excluded since I_{xz} is considerably smaller than the I_x, I_y, I_z due to the shape of the quadrotor, while $I_{xy} = I_{yz} = 0$ due to the vehicle symmetry. Since the force applied to the vehicle as a result of the collective command can be represented as a thrust vector along the negative z_B-body axis, the nonlinear navigation equations for the quadrotor in the reference frame are defined in Figure 5-4 as

$$\ddot{x}_E = (\ \sin\phi\ \cos\psi\ -\ \cos\phi\ \sin\theta\sin\psi\ )\ \frac{1}{m}u, \tag{5.8}$$
$$\ddot{y}_E = (\ -\sin\phi\ \sin\psi\ -\ \cos\phi\ \sin\theta\ \cos\psi)\ \frac{1}{m}u, \tag{5.9}$$
$$\ddot{z}_E = -g\ +\ (\ \cos\phi\ \cos\theta\ )\ \frac{1}{m}u, \tag{5.10}$$

where in this case, $u = \delta_{\text{collective}}$. After linearizing this model, setting $\delta_{\text{collective}} = m\ g + \hat{\delta}_{\text{collective}}$, and dropping small terms in the x_E and y_E dynamics, yields

$$\ddot{x}_E = g\phi, \tag{5.11}$$
$$\ddot{y}_E = -g\theta, \tag{5.12}$$
$$\ddot{z}_E = \frac{1}{m}\hat{\delta}_{\text{collective}}, \tag{5.13}$$

$$\dot{\phi} = p, \tag{5.14}$$
$$\dot{\theta} = q, \tag{5.15}$$
$$\dot{\psi} = r, \tag{5.16}$$

$$\dot{p} = \frac{L}{I_x}\delta_{\text{roll}}, \tag{5.17}$$

$$\dot{q} = \frac{L}{I_y}\delta_{\text{pitch}}, \tag{5.18}$$

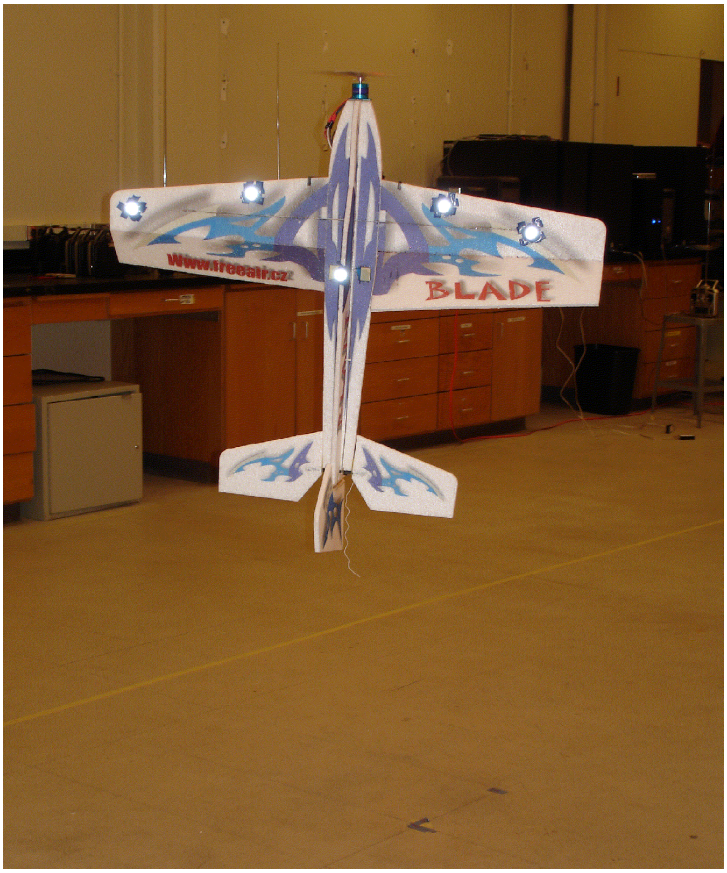
$$\dot{r} = \frac{1}{I_z}\delta_{\text{yaw}}, \tag{5.19}$$

which is the simplified linearized vehicle model around the hover flight condition where $\psi \approx 0$. Here, I_x, I_y, I_z, L, m, and J_R have been measured or determined experimentally for this model.

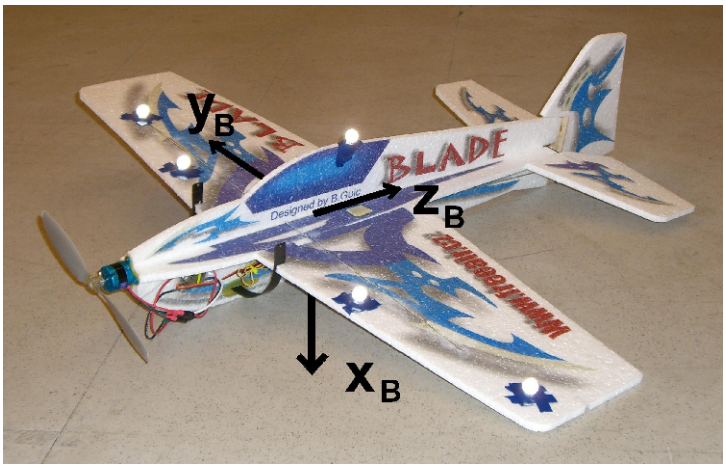
An integrator of the position and heading error is added to the model of each loop so that the controller can remove steady state position and heading errors in hover. The Vicon sensing system accurately and directly measures the system's state, so four simple linear quadratic regulators (LQR) are used to stabilize and control the quadrotor. The controllers use four combinations of vehicle states: ϕ and x_E, θ and y_E, ψ, and z_E. The regulators are designed to optimize the vehicle's capabilities in the hover flight condition, while ensuring that the vehicle can respond quickly to position errors without over-tilting the vehicle. To achieve this behavior, a large cost term is used on the angular position of the vehicle in proportion to the cost terms on position and velocity terms. Likewise, to encourage a faster response, a small cost term is applied to the angular rate. This design is used because the platform is currently optimized for surveillance experiments. In these experiments, a camera is mounted on the quadrotor vehicles facing toward the ground. Thus, large changes in pitch, roll, and yaw affect the vehicle's ability to focus on an item on the ground during surveillance activities. In addition, an anti-windup bumpless transfer scheme (similar to those described in [25]) with adjustable saturation bounds is used to prevent the integrators from winding up while the vehicle is on the ground before, during, and after takeoff and landing. Detailed flight results for this vehicle can be found in the results section.

5.5 Hovering Airplane Control Design Model

In addition to quadrotors, RAVEN can be used to rapidly prototype and test other air and ground systems. For example, a foam R/C aircraft is being used to explore the properties of an aircraft flying in a prop-hang (that is, nose-up) for the purposes of landing vertically and performing other complex maneuvers, such as perching. When hovering an airplane autonomously as shown in Figure 5-5(a), special attention



(a)



(b)

Figure 5-5: Airplane model in a autonomous hover (a) and the airplane axis setup for hover experiments (b)

must be paid when describing its attitude using Euler angles as to avoid angular singularities around $\theta = \pm 90^\circ$ [89]. As shown in Figure 5-5(b), the airplane's body axis reference frame is defined such that the positive x-axis points down from the airplane's undercarriage, the positive y-axis points out along the right wing, and the positive z-axis points out the tail of the aircraft. Using this reference frame, the vehicle's nominal attitude reference in hover corresponds to $\phi = \theta = \psi = 0^\circ$.

Using the assumptions of the earth being an inertial reference frame, the aircraft being a rigid body, and the body frame being fixed to the aircraft, the basic equations of aircraft motion [48, 89] can be written as (5.5)-(5.7), (5.8)-(5.10) with $u = \delta_{\text{throttle}}$, and

$$\dot{p} = qr\left(\frac{I_y - I_z}{I_x}\right) + \frac{A_{\text{rudder}} L_{\text{rudder}}}{I_x}\delta_{\text{rudder}}, \tag{5.20}$$

$$\dot{q} = pr\left(\frac{I_z - I_x}{I_y}\right) + \frac{A_{\text{elevator}} L_{\text{elevator}}}{I_y}\delta_{\text{elevator}}, \tag{5.21}$$

$$\dot{r} = pq\left(\frac{I_x - I_y}{I_z}\right) + \frac{\dot{\omega}_{\text{prop}} I_{\text{prop}}}{I_z} + C_{\text{l,prop}}\rho\left(\frac{\omega_{\text{prop}}}{2\pi}\right)^2 \frac{d_{\text{prop}}^5}{I_z} + \frac{A_{\text{aileron}} L_{\text{aileron}}}{I_z}\delta_{\text{aileron}}. \tag{5.22}$$

In this description, the cross-coupled inertia terms have been excluded since I_{xz} is considerably smaller than the I_x, I_y, I_z due to the shape of the aircraft, while $I_{xy} = I_{yz} = 0$ due to symmetry of the airframe.

Since the flight condition of interest is hover, the influence of external forces and moments on the aircraft, other than gravity, are assumed to be negligible. In addition, since velocities and rotational velocities will be small, multiples thereof can be disregarded. Using small angle approximations, these equations can be considerably simplified. Since $I_{\text{prop}} \ll I_z$, the torque due to a change in the rotational speed of the motor can also be disregarded. Next, define $\delta_{\text{throttle}} = mg + \hat{\delta}_{\text{throttle}}$ and

$$\delta_{\text{aileron}} = -C_{\text{l,prop}}\rho\left(\frac{\omega_{\text{prop,0}}}{2\pi}\right)^2 \frac{d_{\text{prop}}^5}{I_z} + \hat{\delta}_{\text{aileron}}, \tag{5.23}$$

where $\omega_{\text{prop,0}}$ is the average rotational speed of the motor to keep the airplane in hover [57]. Next, since the vehicle's reflective markers are mounted on top of the both wings, the sensors are visible to cameras only facing the top side of the vehicle when the vehicle is in hover. Therefore, $\psi_{\text{reference}} = -\frac{\pi}{2}$ for safety reasons to ensure that there are at least three or more cameras facing the sensors on the wing for this testing. Therefore, by linearizing the equations of motion the simplified equations for

the airplane in hover become

$$\ddot{x}_E = g\theta, \tag{5.24}$$
$$\ddot{y}_E = g\phi, \tag{5.25}$$
$$\ddot{z}_E = \frac{1}{m}\hat{\delta}_{\text{throttle}}, \tag{5.26}$$

$$\dot{\theta} = q, \tag{5.27}$$
$$\dot{\phi} = p, \tag{5.28}$$
$$\dot{\psi} = r, \tag{5.29}$$

$$\dot{p} = \frac{A_{\text{rudder}} L_{\text{rudder}}}{I_x}\delta_{\text{rudder}}, \tag{5.30}$$
$$\dot{q} = \frac{A_{\text{elevator}} L_{\text{elevator}}}{I_y}\delta_{\text{elevator}}, \tag{5.31}$$
$$\dot{r} = \frac{A_{\text{aileron}} L_{\text{aileron}}}{I_z}\hat{\delta}_{\text{aileron}}. \tag{5.32}$$

Here, I_x, I_y, and I_z correspond to the body moment of inertia terms and A_{elevator}, A_{rudder}, and A_{aileron} correspond to the deflected area of each control surface that is subject to propeller flow while the airplane is in hover. In addition, L_{elevator}, L_{rudder}, and L_{aileron} are the lengths of the control surface moment arms. These terms are measured or determined experimentally for this model.

Four control schemes of types proportional plus derivative (PD) and proportional plus integrator plus derivative (PID) are used to stabilize and control the airplane in hover. These controller schemes are applied to combinations of vehicle states ϕ and x_E, θ and y_E, ψ, and z_E and are designed to optimize the vehicle's capabilities in hover. In particular, to prevent the vehicle from moving too quickly around the hover condition, the controllers use large gains on state derivative errors. Position gains are small in comparison to angular gains in order to maintain stability in hover when correcting position. This design ensures that a disturbance (for instance, wind) does not cause the vehicle to oscillate when trying to reachieve its original position.

Several issues are involved in trying to control an airplane in hover. Firstly, as motor speed changes, so does propeller drag torque. This issue is resolved by adding an aileron deflection proportional to motor speed error around the equilibrium speed. The varying speed of the propeller also affects the speed of airflow over control

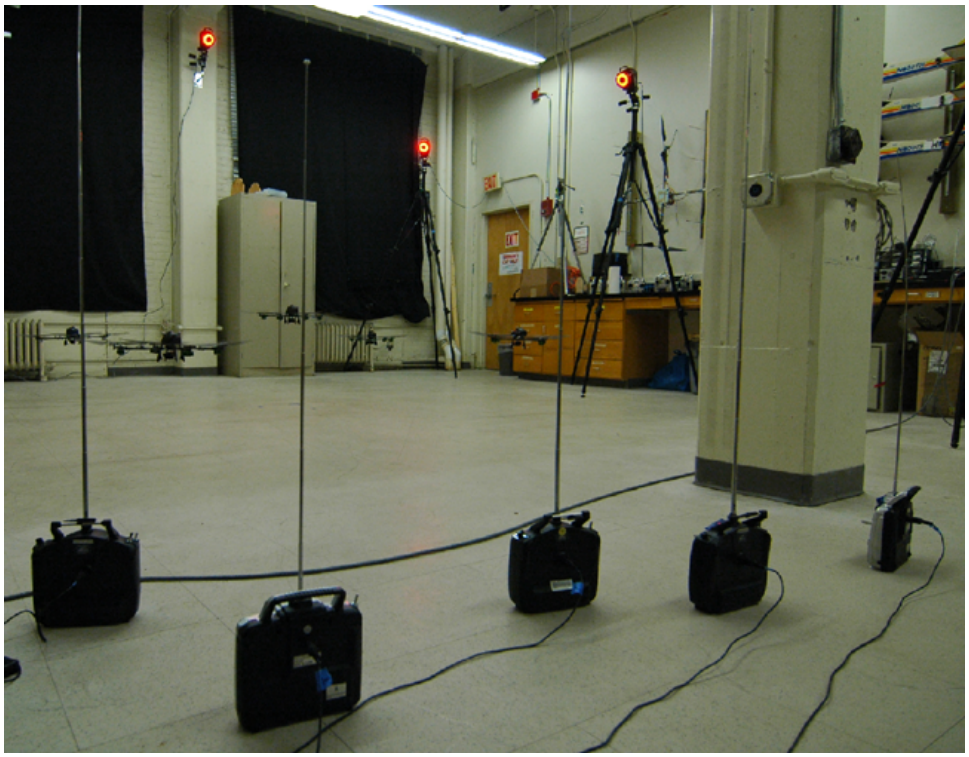
surfaces, hence affecting control surface actuation. This issue is most prominent in roll control. To resolve this issue, the ailerons are deflected additionally at low throttle settings.

Unfortunately, adding additional deflection to the ailerons does not solve all problems. As aileron deflection is increased, the ailerons block part of the propeller wash that would otherwise have reached the elevator. To resolve this issue, a gain proportional to aileron deflection is added in the elevator control path to compensate for reduced airflow over the elevator and improve the vehicle's pitch response when the ailerons are deflected.

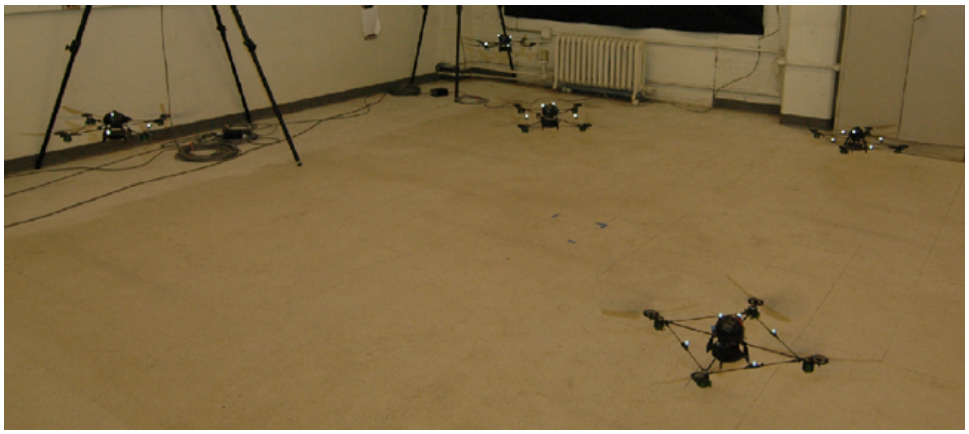
Finally, since the airframe of the vehicle lacks rigidity, fast control surface motions cause the vehicle's airframe to twist. This twist causes the reflective markers used by the offboard positioning system to shift in position, thereby giving an incorrect estimate of the airplane's momentary location. Although part of this issue is resolved by the servos being unable to track large amplitude inputs at high frequencies, the controller gains are sized to minimize rapid changes in control surface position. Flight results for the airplane in hover are given in the next section.

5.6 Results

Various components of RAVEN have been under development since May 2005. The goal of the testbed is to study long-duration missions in a controlled environment, so the recent focus of our lab has been to ensure that RAVEN can reliably fly multiple mission sorties. As shown in Figure 5-6, a variety of multi-vehicle tests and mission scenarios have been flown using the testbed. Since January 2006, over 2000 vehicle experiments have been performed, including approximately 60 flight demonstrations (around 30 per day) during a 16-hour period at the Boeing Technology Exposition at Hanscom Air Force Base near Lexington, Massachusetts, on May 3rd and 4th, 2006. Each of the tests performed at the event involved two vehicles. One test involved two air vehicles flying a 3D coordinated pattern (as shown in Figure 5-9), and the other involved an air vehicle following a ground vehicle. These demonstrations show that the platform can perform multiple UAV missions repeatedly, on demand, and with minimal setup.



(a)



(b)

Figure 5-6: Fully autonomous flight test with five UAVs (a), and a close-up of five UAVs in flight (b). In this flight, the autonomous tasking system commanded all five vehicles to takeoff and hover 0.5 m above the ground for two minutes. In (b) the five vehicles are shown as they hover during the test. In (a) the five transmitters for each vehicle are shown. Each transmitter is connected directly to a ground computer that monitors and commands one vehicle during flight.

5.6.1 Quadrotor

Typical results from a 10-min hover test are shown in Figure 5-7. In this test a single quadrotor is commanded to hold its position at $(x, y, z) = (0, 0, 0.7)$ m for a 10-min period of time. Figure 5-7 shows four plots, including a plot of the vehicle's x- and y-positions during this test. The dashed red box in the picture is ± 10 cm from the center point. As shown in Figure 5-7, the vehicle maintains its position inside this 20-cm box during the entire flight. The remaining plots in the figure are the histograms of the vehicle's x-, y-, and z-positions during these tests. This test shows that a quadrotor can maintain both its altitude (staying between 0.65 to 0.75 m) and position (staying mostly within 0.05 m from center) in hover over full charge cycle of a battery.

The results of a single-vehicle waypoint tracking experiment are shown in Figure 5-8. In this test the vehicle is commanded to hold its altitude at one meter while flying at a velocity of 0.05 m/s to and hovering for ten seconds at each of the following waypoints: (-1.5,0,1) m, (-1.5,-0.5,1) m, (1.5,-0.5,1) m, (1.5,5.5,1) m, (-1.5,5.5,1) m and back to (-1.5,0,1) m. The purpose of this test is to observe the vehicle as it tries to follow a set trajectory around the indoor laboratory flight space. The plots show that the vehicle follows the trajectory around a 3-m by 6-m rectangular box as specified. The cross-track error is less than 15 cm from the specified trajectory at any given time during the flight.

In addition to these single-vehicle experiments, several multi-vehicle experiments and test scenarios have been conducted. These tests include, but are not limited to, formation flight tests, coordinated vehicle tests involving three air vehicles, and multi-vehicle search and track scenarios.

Figure 5-9 shows the results from a two-vehicle coordinated flight experiment. In this experiment, the vehicles are commanded by the system's task advisor to takeoff and fly a circular trajectory maintaining a constant speed of 0.25 m/s and 180 degrees of angular separation. In particular, the vehicles are flying in a circle (as projected in the x–y coordinate frame) while they change altitude (flying from an altitude of 0.5 m to 1.5 m) as they move around the flight pattern. The upper left plot of Figure 5-9 shows the x–y projection of one of the five circle test flights that were completed as part of this experiment. Notice that the vehicle trajectories in the lower right corner of the plot appear to be more noisy. This disruption is partially caused by the quadrotors flying through the rotor downwash from another vehicle. Flight testing has shown that the downwash from these quadrotor vehicles is substantial, thus making it difficult to fly one quadrotor underneath a second quadrotor without

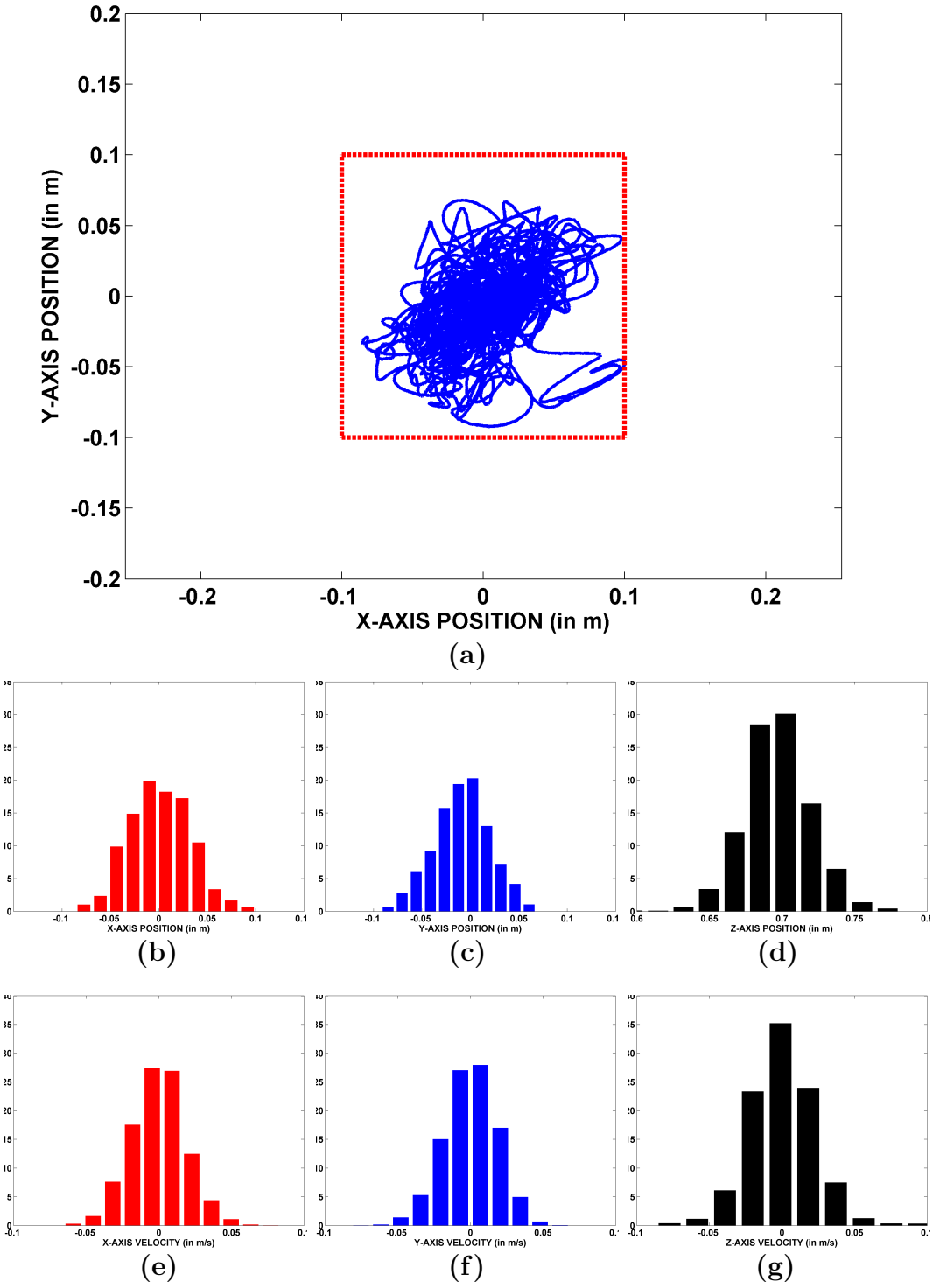


Figure 5-7: Single vehicle hover experiment. In this test flight, a quadrotor UAV is commanded to hover at $(x, y, z) = (0, 0, 0.7)$ m for 10 min. (a) shows the x–y plot of vehicle position (a), (b)-(d) show histograms with percentage of time at location for x, y, and z positions, and (e)-(g) show histograms with percentage of time at each flight condition for x, y, and z velocities.

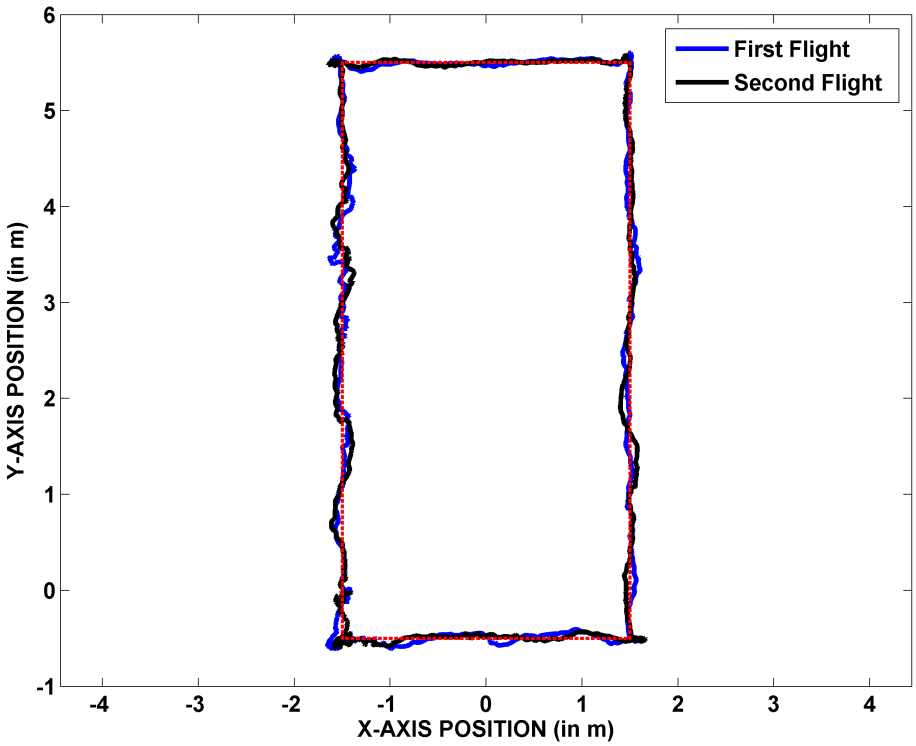


Figure 5-8: Single vehicle waypoint tracking experiments. In this test, a quadrotor vehicle is commanded to fly from points (-1.5,0,1), (-1.5,-0.5,1), (1.5,-0.5,1), (1.5,5.5,1), (-1.5,5.5,1) and back to (-1.5,0,1) m

significant altitude separation. The lower plot shows a 3D view of the trajectory, making it clear that the vehicles are also changing altitude during the flight. The upper right plot shows the results of five consecutive two-vehicle circle test flights. These test flights were performed over a 20-min time span. These results demonstrate that experiments run on RAVEN are repeatable and that the platform can be used to perform multiple test flights over a short period of time.

5.6.2 Hovering Airplane

Just as with the quadrotor, numerous hover tests were performed with the foam airplane. Typical results are shown in Figure 5-10 in which the vehicle is commanded to hold its position at $(x_E, y_E, z_E) = (0, 0, 0.7)$ m for five minutes. Figure 5-10 shows four plots, including a plot of the vehicle x–y location while it maintained its position and attitude. The dashed red box in the picture is ± 0.5 m from the center point. As

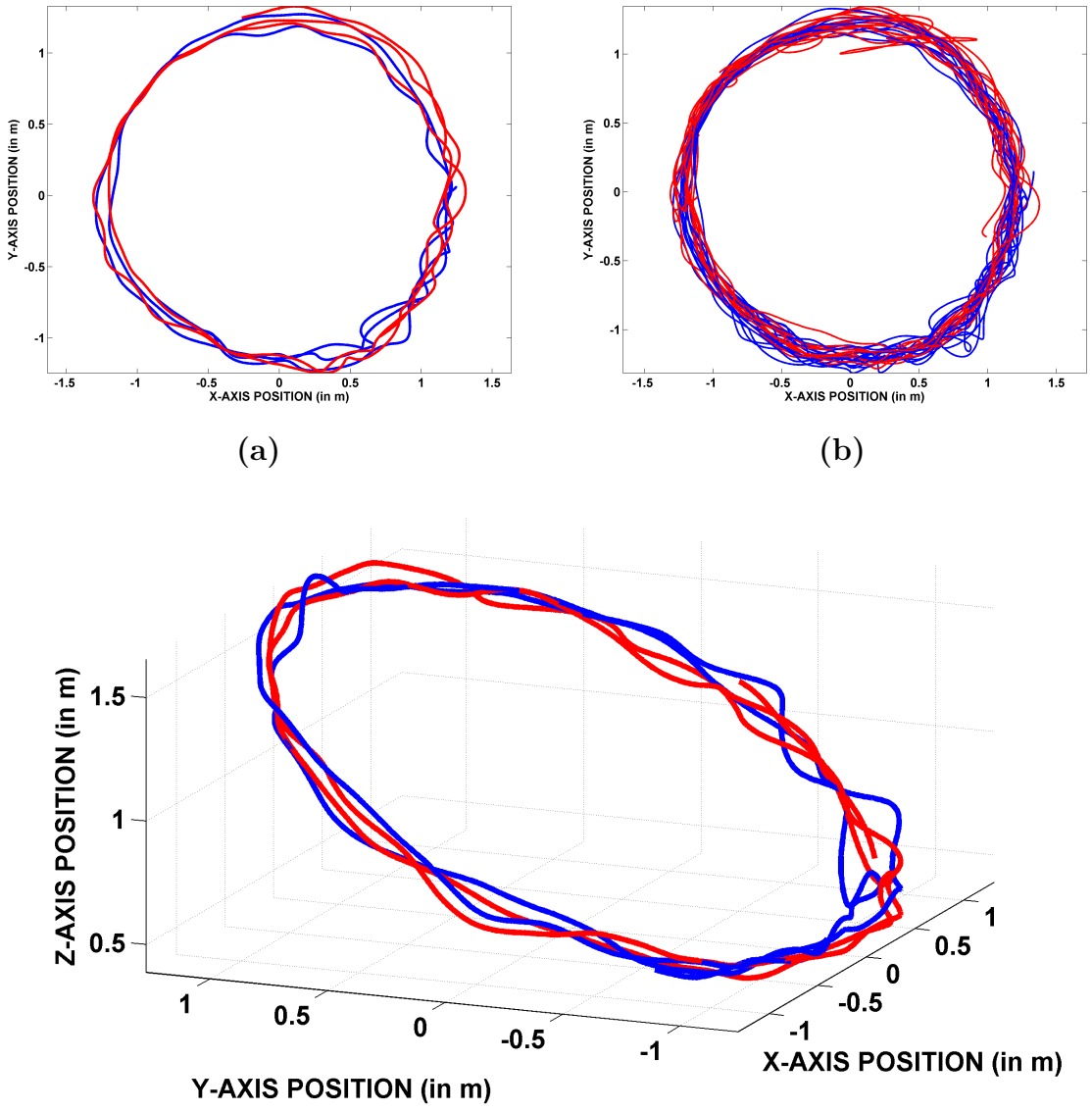


Figure 5-9: Multi-vehicle coordinated flight experiment. In the test, two vehicles are commanded to fly at a constant speed in a circular pattern with changes in altitude. In this experiment, the vehicles are commanded by the system's task advisor to takeoff and fly a circular trajectory maintaining a constant speed of 0.25 m/s and 180 degrees of angular separation. In particular, the vehicles are flying in a circle (as projected in the x–y coordinate frame shown in (a) and (b)), while they are changing altitude (flying from an altitude of 0.5 m to 1.5 m) as they move around the flight pattern as shown in (c). This test is repeated multiple times and the vehicles fly similar flight paths in five consecutive tests as shown in (b).

shown in Figure 5-10, the vehicle maintains its position inside this 1-m box for most of the 5-min test period. The remaining plots give histograms of the vehicle's x, y, and z positions. These plots confirm that the vehicle is within a 20-cm box around the target point over 63% of the time. These plots also show that the vehicle precisely maintains its altitude (staying between 0.65 to 0.75 m) during the entire hover test.

Figure 5-11 shows two waypoint tracking tests for the airplane starting from hover and then moving at a horizontal rate of 0.3 m/s between a set of waypoints while remaining nose-up. In the top figure the vehicle started from $(1.5, 5.5, 1)$ m. The vehicle started from $(-1.5, 0, 1)$ m in the bottom figure. In both tests, the airplane flies along the desired flight path as it hovers between each waypoint despite the fact that the vehicle had less control authority in its yaw axis, making it difficult to maintain the vehicle's heading during flight. The reduced control authority of the vehicle's yaw axis in hover is due to the fact that propeller wash covers less than 10% of the ailerons in the hover flight condition, thus reducing the vehicle's ability to counteract disturbances in propeller torque while maintain vehicle heading. As a result, external disturbances cause the vehicle to deviate from a straight flight path between waypoints. However, the vehicle stays within 0.5 m of the intended flight path throughout most of the tests.

Finally, the development and incorporation of this aircraft into the testbed in this flight condition was accomplished less than three weeks after acquiring the aircraft. This time period includes a week to construct the airplane. In fact, three days after the airplane made its first human-controlled flight, the airplane made its first autonomous hover flight. This activity was performed starting at the end of September 2006 to the middle of October 2006, validating the platform's rapid prototyping capability for new vehicle hardware. A video of the aircraft in the hover flight condition can be found online at [1].

5.6.3 Multi-Vehicle Testing using Mission and Tasking System Components

A number of multi-vehicle tests have been flown using the RAVEN at MIT to demonstrate the mission, task assignment and control level health management algorithms and capabilities. In this test suite, three UAVs equipped with cameras and 2000 mAh batteries were used to search for ground vehicles in the test area. A number of multi-vehicle tests were flown as part of this test suite to demonstrate the mission, task assignment and control level health management algorithms.

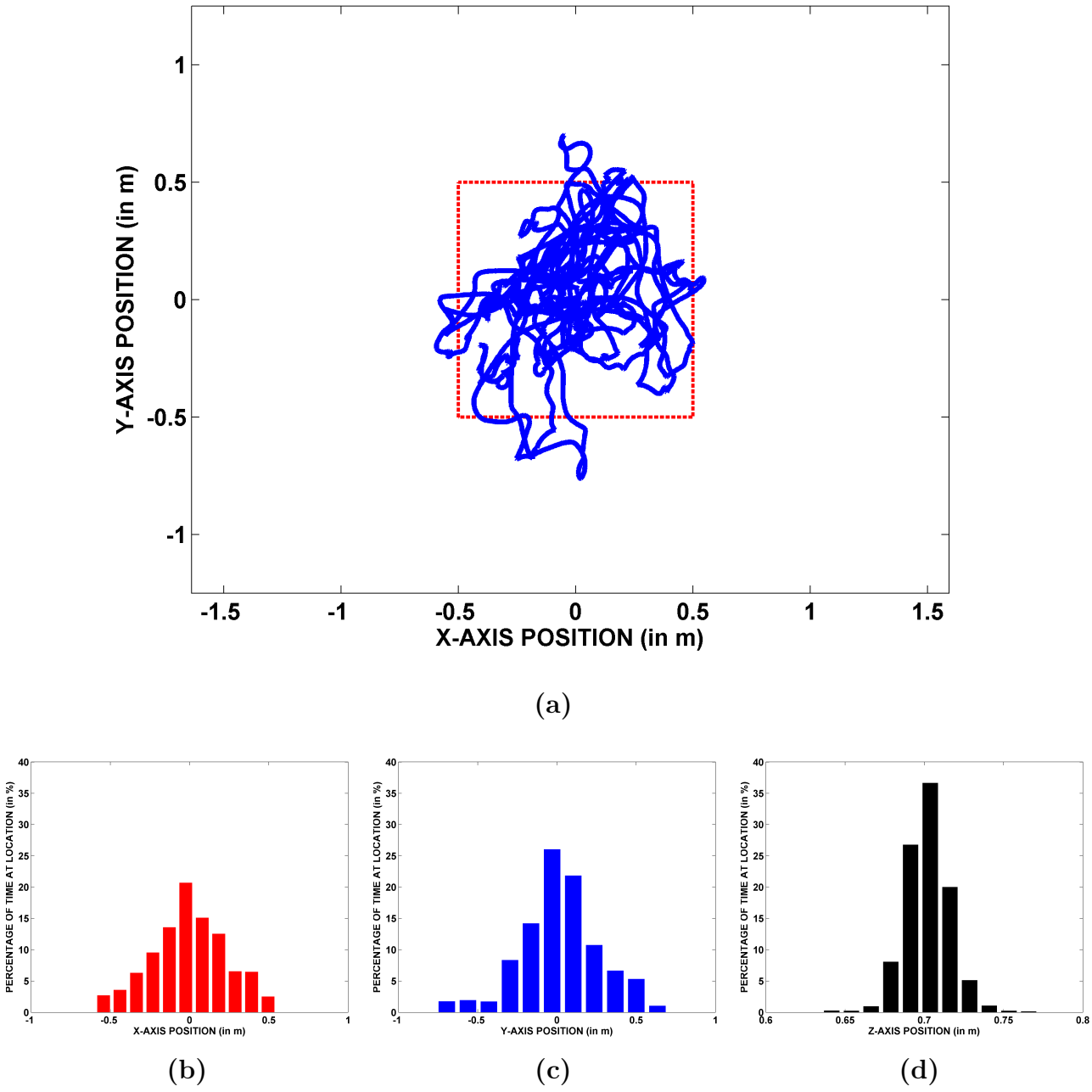
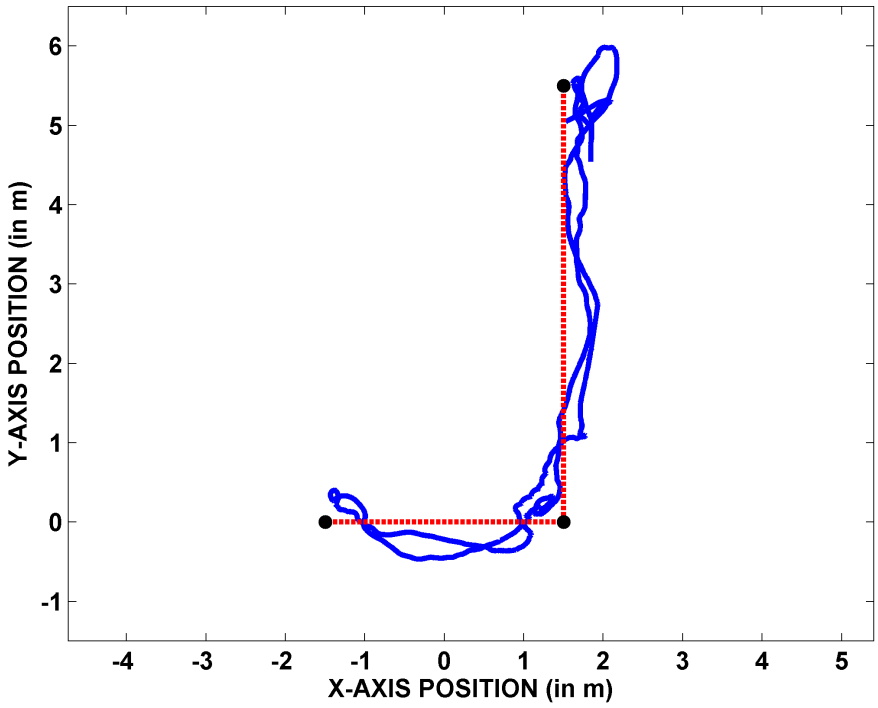
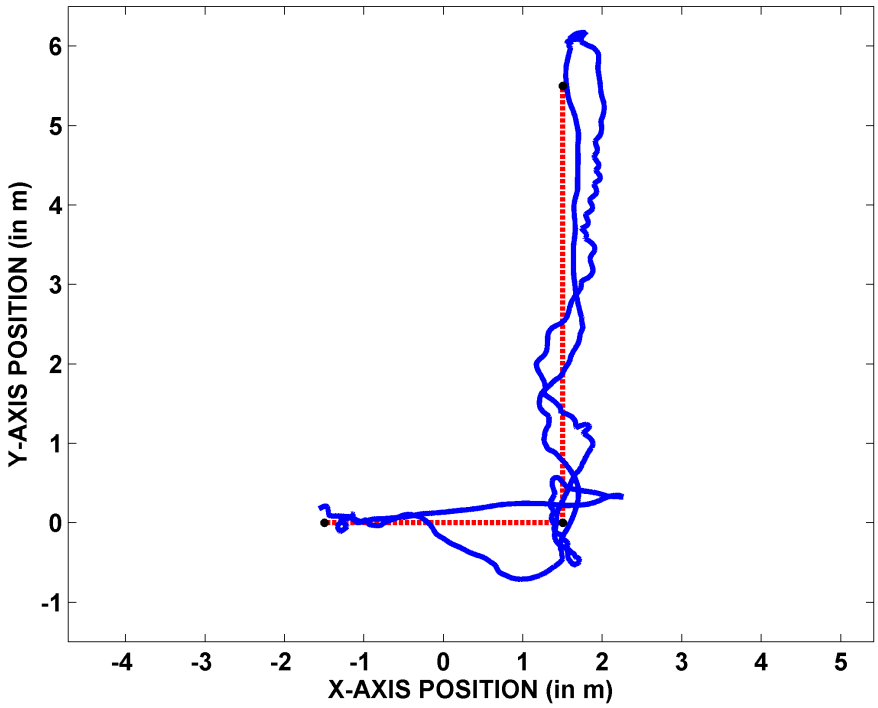


Figure 5-10: Airplane hover experiment. In this flight test, the airplane is commanded to hover at $(x, y, z) = (0, 0, 0.7)$ m for 5 min. (a) shows the x–y plot of vehicle position, while (b)-(d) show histograms with percentage of time at location for x, y, and z positions. These results demonstrate that the vehicle can hold its position reliably during flight. In this test, the vehicle remains inside the 20-cm box over 63% of the time during the 5-min hover test flight.



(a)



(b)

Figure 5-11: Hovering airplane waypoint tracking experiments. In these flight tests, the vehicle flies between points (-1.5,0,1), (1.5,0,1), and (1.5,5.5,1) m. These results show that the hovering airplane can fly between waypoints in hover.

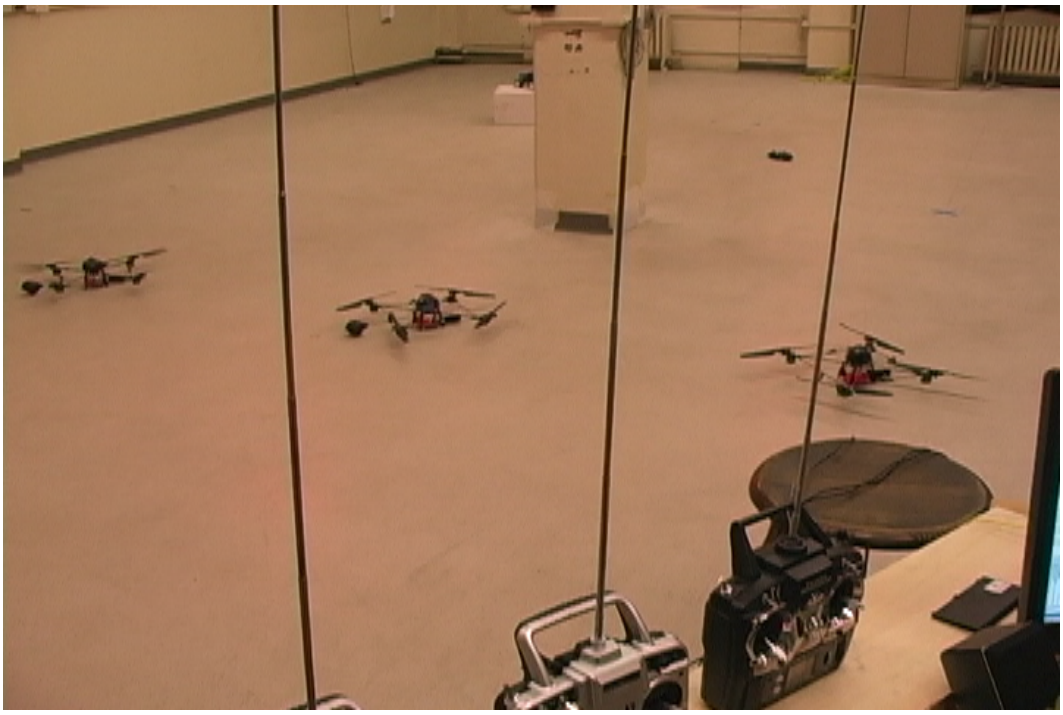


Figure 5-12: Multi-vehicle mission flight test setup

Each vehicle-battery combination had measured flight times of 11 to 12 mins under normal operating conditions prior to these tests. The initial vehicle layout for each test is shown in Figure 5-12. Each UAV was placed 3 m north of the search area behind a concrete column. Therefore, these UAVs had to avoid the pole during transitions between the search and maintenance areas. In addition, two ground vehicles were used in these tests. The ground vehicle in the western end of the search area could moved via remote control. Therefore, the actions of the UAVs tracking this vehicle using vision had to be coordinated via the task advisor (running the modified version of the Receding-Horizon Task Assignment (RHTA) algorithm described in [98]) to estimate the ground object's position and velocity. Likewise, the second ground object was placed on top of a box. To accurately detect its position, UAVs had to make multiple observations of this object (from different orientations) since the terrain of the search area was not known to the mission system a priori.

Flight test data from one of these test flights is shown in Figures 5-13 to 5-16. In this 12 min test flight, the camera from UAV #3 was removed to demonstrate a complete camera failure. At the beginning of this flight demonstration, the mission system commanded one vehicle to search test area for ground objects. Once airborne, the system's task advisor used the UAV's camera images to locate ground objects. Each time a ground vehicle was detected by the UAV's vision system, information of the object's location was sent to the task advisor for future reference.

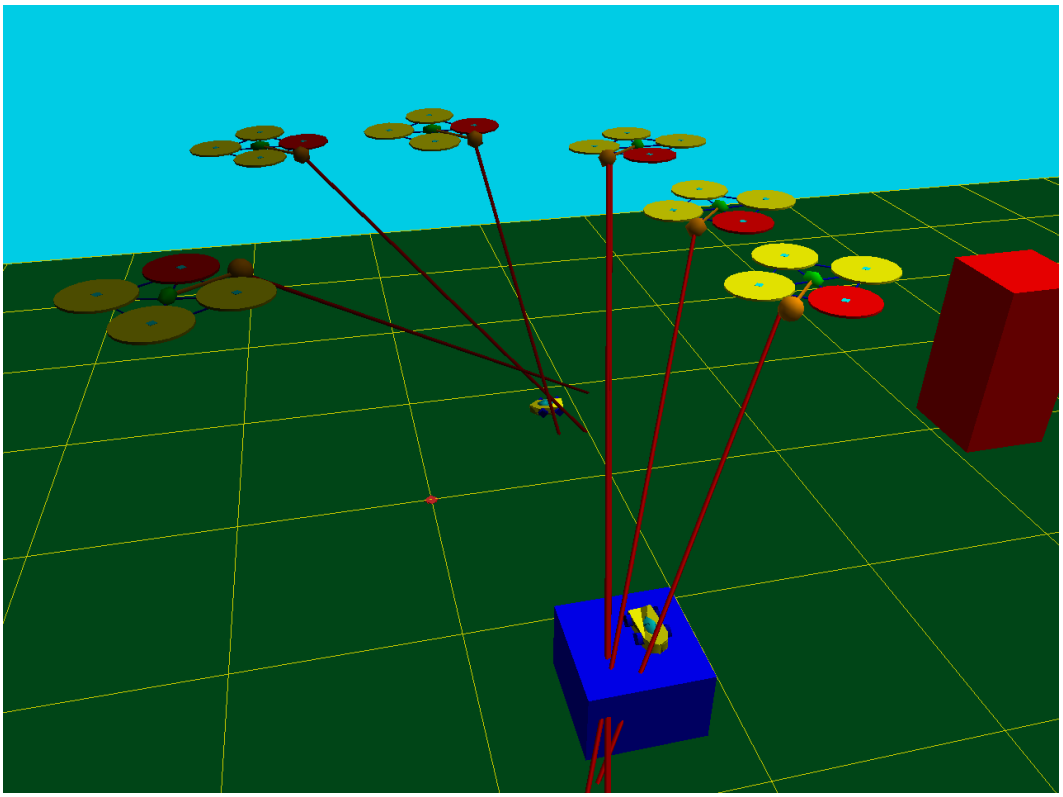


Figure 5-13: Single vehicle searching the test area

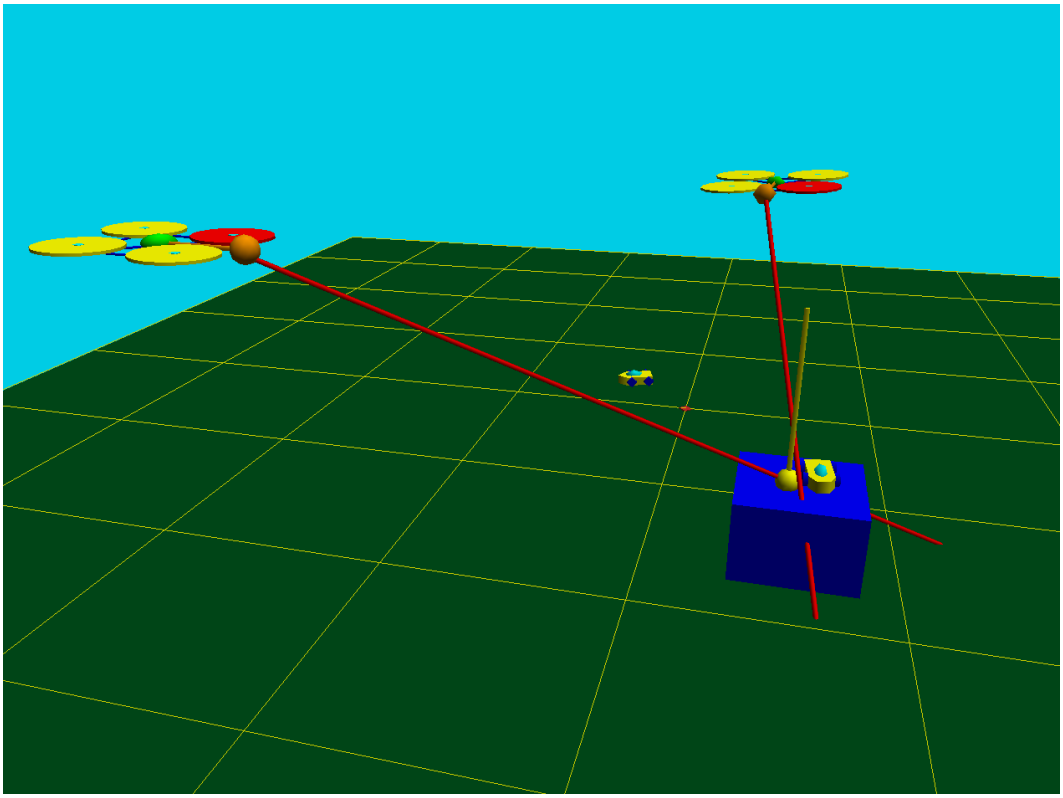


Figure 5-14: Two UAVs observing a ground vehicle on box

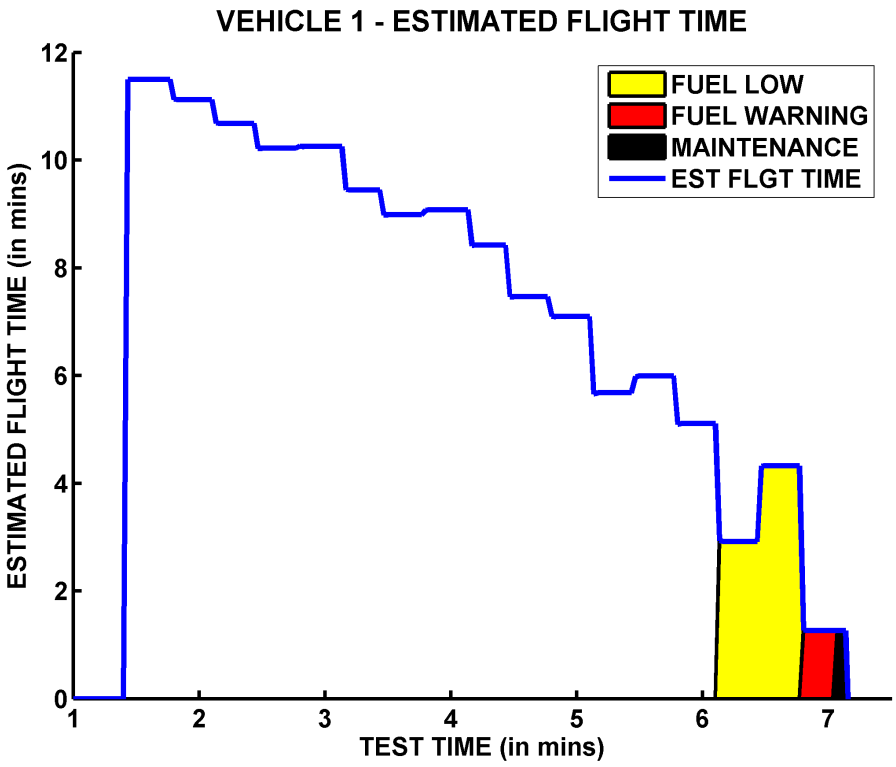


Figure 5-15: UAV #1 estimated vs actual flight time

To verify the reported ground vehicle locations, the system's task advisor requested a second UAV from the mission manager. After the second vehicle reaches the search area, the task advisor commands both vehicles to monitor the ground vehicles. Figure 5-14 shows UAVs #1 and #2 tracking the ground objects during the flight test. These images were generated using the RAVEN 3D operator interface playback utility. In Figure 5-14 both vehicles are commanded by the task advisor to remain about 90° from one another to identify the ground vehicle's location and orientation on the box. In this figure, the yellow car-like object is the actual location of the car (as determined by the RAVEN's positioning system) and the bobbin-like spherical object is the task advisor's estimate of the ground vehicle location (as generated using the processed images by the camera). More information on the vision-based task assignment portion of this demonstration can be found in [11].

During the test the system's health monitors were used to manage UAVs in the flight space. As shown in Figure 5-15, the estimated flight time of UAV #1 was monitored by the mission system to determine when it needed maintenance during the mission. Here, the data shows that about 2.5 mins into its flight, the vehicle's battery started to degrade faster than expected. This degradation can be attributed

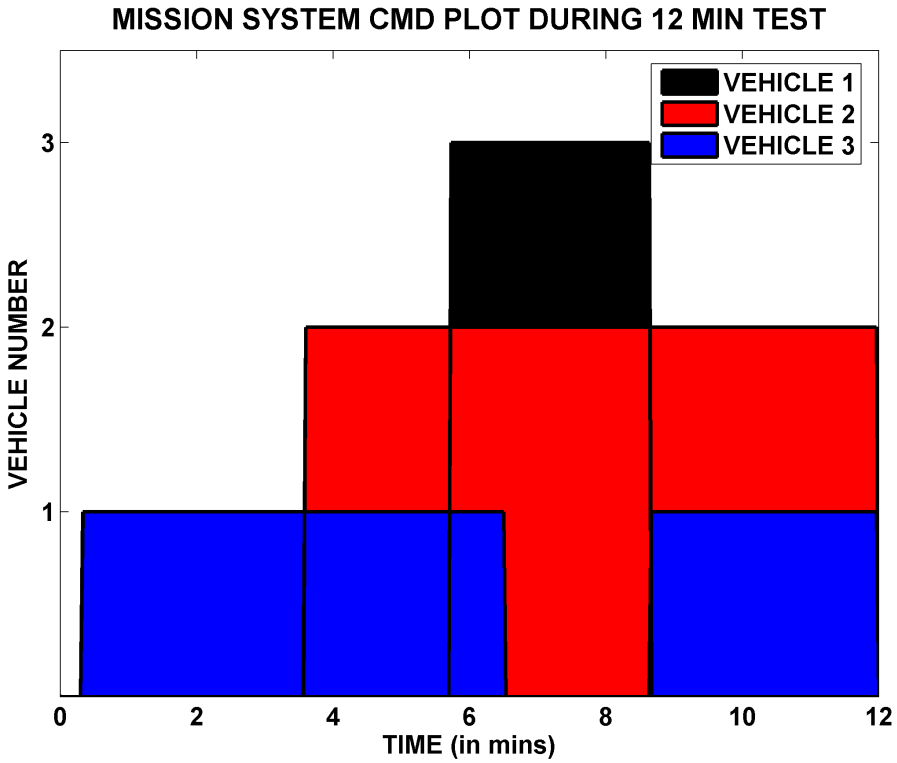


Figure 5-16: Mission system commands during flight

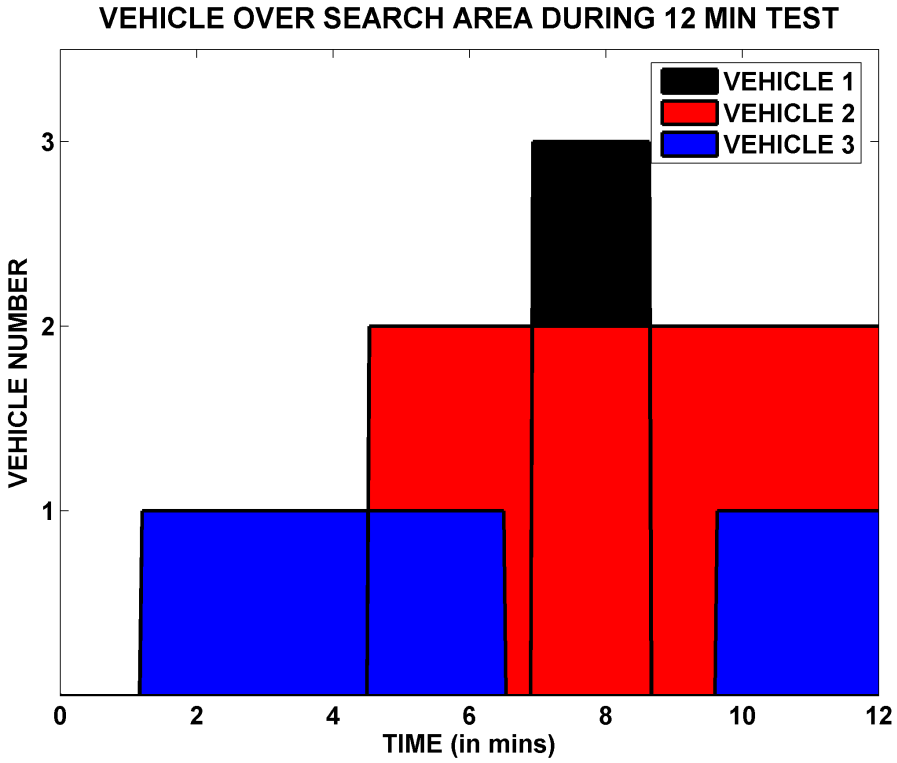


Figure 5-17: UAVs over the search area during test

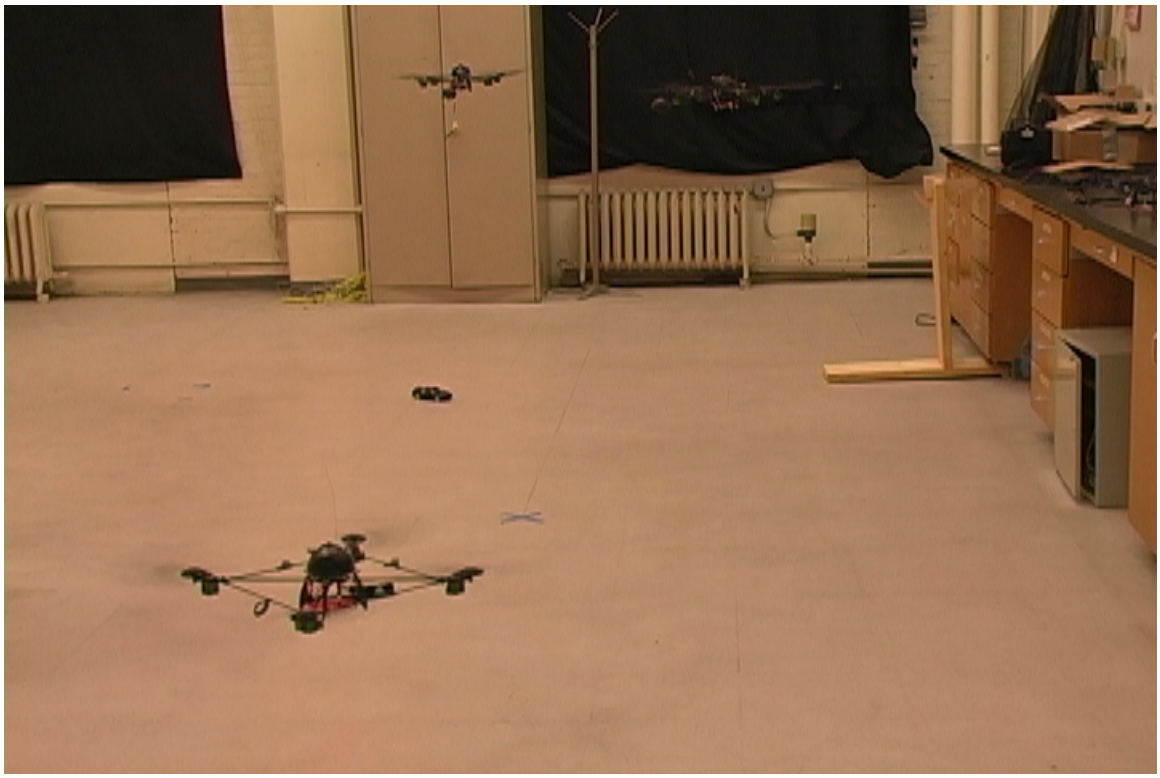


Figure 5-18: Two UAVs tracking a ground vehicle during vehicle cycling for maintenance

to the fact that this vehicle was flown many times and had been involved in collision prior to this experiment. Since the motors on the vehicle were warn, UAV #1 had trouble flying with UAV #2 while tracking the moving ground vehicle (due to the fact that it got caught in UAV #2's propeller wash). Hence, 6 mins into the experiment, UAV #1 required more power to maintain its position while observing the ground vehicles.

As shown in Figure 5-16, the mission system automatically commands each vehicle to participate in the mission based on the task advisor's needs and the vehicle's health. After receiving the take off command by the mission manager, vehicles required about 1 min to reach the search area. In this case, the mission system proactively commands UAV #3 to take-off (as shown in Figure 5-18) and replace UAV #1 since the mission system recognized that UAV #1 was using a abnormal level of collective to maintain its position during the experiment.

Normally, after the "Fuel Low" message is sent, it takes 1.5 mins for the "Fuel Warning" message to be generated for a vehicle in normal condition. However, Figure 5-15 shows a similar time plot of the vehicles on location in the task area. UAV #1 was commanded back to base early to ensure that it safely arrived at base to receive

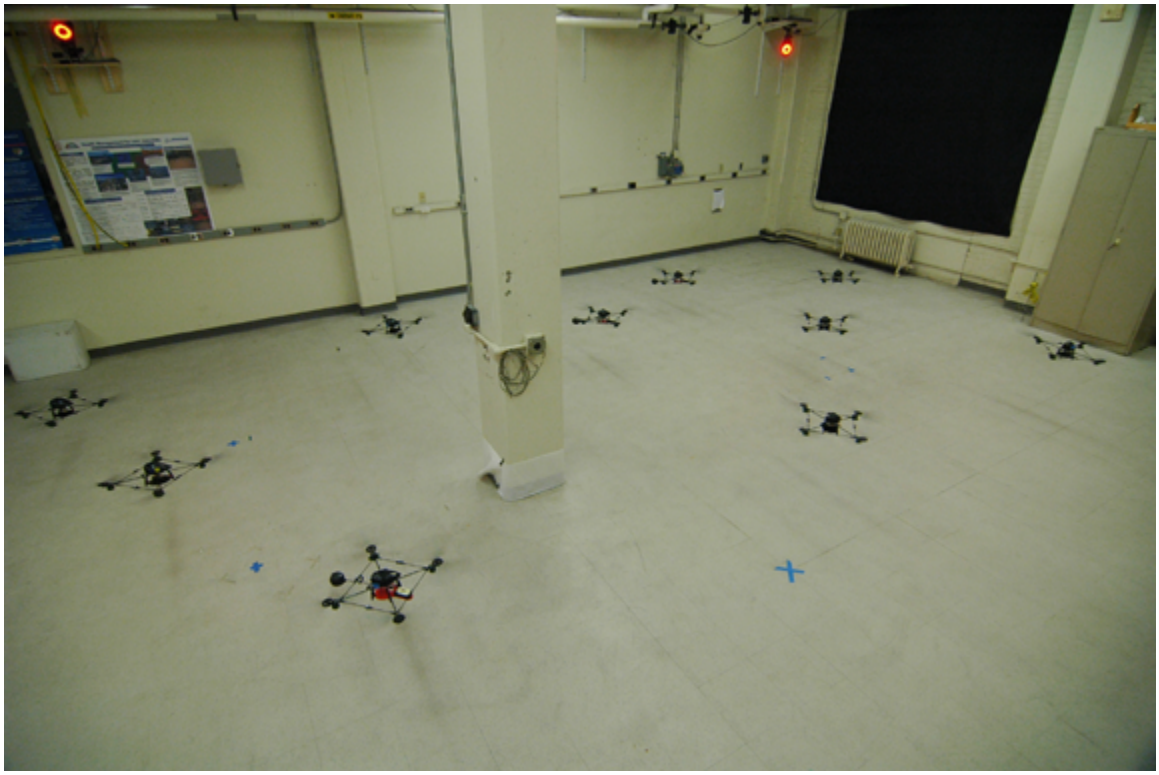


Figure 5-19: Fully autonomous flight test with 10 UAVs and 1 operator

maintenance. To prevent a collision, the task manager reactively modified UAV #3's flight path so that UAV #1 could safely exit the flight area, thus delaying UAV #3's entry into the search area (as shown in Figure 5-17).

Finally, since UAV #3 did not have a camera, the system operator manually commanded UAV #3 back to base. Since the operator was able to replace the battery on UAV #1 before failing UAV #3, the mission system recognized that a second vehicle was available. As shown in Figure 5-16, as soon as UAV #3 is registered as unavailable, the mission system reactively commands UAV #1 back to the search area for the rest of the test. A video of a similar multi-vehicle search test flight using vision, as well as other flights, can be found online at `http://vertol.mit.edu`.

5.7 Summary

This chapter describes a new indoor multi-vehicle testbed developed at MIT to study long-duration missions in a controlled environment. RAVEN has enabled an increase in the unmanned vehicle to operator ratio for multi-vehicle missions to 10:1 (shown in Figure 5-19), which is an important achievement in the overall goal of reducing the cost and logistic support needed to operate these systems.

Furthermore, while other testbeds have been developed to investigate the command and control of multiple UAVs, RAVEN is designed to explore long-duration autonomous air operations using multiple UAVs with virtually no flight-time restrictions. RAVEN represents a significant technological step forward for UAV development by enabling the rapid prototyping of coordination and control algorithms in a controlled environment. In fact, over the last year, RAVEN has allowed researchers to demonstrate multi-vehicle obstacle avoidance, coordination, formation flight, search, autonomous recharge, vision-based landing, and object tracking scenarios using the quadrotors in real time. Videos showing most of these demonstrations can be found online at [1].

Chapter 6

Multi-Agent Health Management and Extended Mission Testing

Today, unmanned aerial vehicles (UAVs) are used to aid in search and rescue, surveillance, and other missions over a variety of locations throughout the world. Teams of multiple UAVs can provide valuable information to operators for making mission critical decisions in real-time. These multi-agent teams offer a promising alternative to a number of high-risk manned mission scenarios by performing missions in dangerous and unknown environments without considerable risk to the human operators. However, one of the main problems with any air operation is the coordination of resources when operations require a persistent capability over the desired operational area. Although the actual timing of the replacement of vehicle assets plays an important role in regard to the continuity of coverage in such missions, the maintenance systems and operators supporting these vehicles also have an equal role in supporting these systems by ensuring assets are readily available if needed in an emergency situation.

As mentioned earlier, many questions related to the timing and upkeep of such systems are very similar to many questions arising in manufacturing and the airline industry. Though many of the issues presented in previous research apply to problems related to scheduling concerns, some of the challenges specific to persistent operations include (but are not limited to): several multi-agent teams may be operating simultaneously that may or may not coordinate tasks and information about the current task, vehicle assets may be lost during the course of a mission, and little or no information about the vehicles may be directly available to the operator during the mission. For example, a vehicle failure may become known only after the vehicle

has failed to show up for refueling past a given deadline.

Though many papers discuss how to perform the operations using aerial platforms, the purpose of this chapter is to discuss the development and implementation of techniques used to manage autonomous unmanned aerial vehicles (UAVs) performing long-term (24/7) persistent surveillance operations. This chapter presents mission health monitors aimed at identifying and improving mission system performance to avoid down time, increase mission system efficiency and reduce operator loading. Using an indoor multi-vehicle testbed presented in Chapter 5, this chapter will present the infrastructure used to execute an autonomous persistent surveillance operation over a 24-hr period and show flight test results from recent automated persistent surveillance missions and UAV recharging experiments.

6.1 Simplified Persistent Surveillance Mission

To understand the issues related to a long-term persistent surveillance mission, we begin by investigating questions related to a smaller group of vehicles with a centralized task manager and a single surveillance task requiring a vehicle on-site for an extended period of time. In this problem, the major underlying assumption is that all vehicles are **not** expendable. As a result, the main goal of the mission system is to ensure that not only is a persistent presence maintained over the target area, but also that each vehicle is returned to the base location safely.

As discussed in Chapter 4, consider a collection of M agents of the same capability and a single surveillance location. This problem setup provides a simplified resource management problem for which we can find a tractable solution for the task assignment and agent task scheduling problem under simplified assumptions. Assume that a vehicle's availability is based on its current health state. At the vehicle level, many aspects related to a vehicle's current performance and capabilities can be considered as part of this state. For example, battery charge, sensor capabilities, and motor wear play a large role in an electric vehicle's capability to perform a mission. To simplify this problem, assume that the vehicle's remaining flight time will denote the vehicle's health state. In addition, assume that the all of the vehicles have the same task capabilities (e.g., all vehicles can only be used for surveillance) and leave from the same general location.

Earlier, we showed that the simplified persistent surveillance mission resource management problem can be posed as a Markov Decision Process (MDP). Once again

note that as the number of vehicles added to the mission increases, the size of the state and action space increases quickly. Therefore, we can used the methods described in Chapter 4 to find policies for this problem. Using Approximate Linear Programming (ALP), we can generate an approximate cost-to-go structure for this problem using the basis function generation algorithm in Table (3.1). As shown in Chapter 4, this approach can be used to find an approximate solution to the original problem that will provide us with a "good" policy in real-time. Note that we define a "good" policy as a policy that allocates enough vehicles to each task location to meet mission goals, while managing the health situation of each vehicle.

Another approach is to use a heuristic policy based on experience. For example, one heuristic policy that can be used is that for every state use the following rule:

- Let T_D represent the traveling time from the base location the surveillance location and C be a safety factor time constant. If there is a vehicle at the surveillance location with enough fuel to continue at the surveillance location, then continue with the same action from the previous state

- Else if the vehicle at the surveillance location needs to be replaced, then send the vehicle that has the least number of flights that is at the base location fully refueled. If there are more than one vehicle in this condition, choose the vehicle with the lowest index $i \in \{1, ..., M\}$

- Else the vehicle at the surveillance location needs to be replaced and there are no vehicles at the base location then command the vehicle with the least number of flights to go to the surveillance location when ready. If there are more than one vehicle in this condition, choose the vehicle with the lowest index $i \in \{1, ..., M\}$

This heuristic approach has been used to manage mission flight tests during RAVEN multi-vehicle testing with success.

6.2 Mission Planning and Vehicle Health Monitoring

In planning and monitoring the persistent surveillance mission, the automated task manager must be able to assess the status of the vehicles to determine whether a failure has occurred during the flight. Most persistent surveillance mission models make the assumption that the remaining flight time for a vehicle is known or decays

in a well-defined way. However, this assumption does not hold in many cases. For example, small electric-powered UAVs are powered by batteries. Since these batteries are charged and discharged over time, they decay at different rates. This rate of battery decay can change based on the type of vehicle that is used, how the battery is stressed during vehicle use, the charger used to re-charge the battery, the temperature of the environment, and many other characteristics. Therefore, one cannot assume that every electric-powered vehicle with a fully-charged battery will be able to sustain a specific flight time without considering at least some of the other information provided.

For this reason, knowing the current health state of the vehicle can improve the performance of the overall mission system. Therefore, in order to automate persistent surveillance missions, two additional components are required: 1) vehicle health monitoring – and more specifically, a fuel or battery health monitor, and 2) an automatic vehicle maintenance and refueling / recharging station. In the next two sections, this chapter will describe the methods used to implement these features to automate a persistent surveillance mission.

6.3 Battery Health Monitoring

As mentioned earlier, the RAVEN was constructed to study, implement and analyze the performance of techniques for embedding the fleet and vehicle health state into UAV mission planning [97]. In order for higher-level mission systems to adequately assess and diagnose potential issues with a mission plan, each vehicle must inform other system components of its current status (e.g., location, health, mission-related information). For example, air vehicles have flight time limitations based on fuel and maintenance constraints. To determine the current capabilities of these vehicles, sensors can be added to measure the vehicle's nominal health state. Using this information, health monitors can be developed to evaluate the vehicle's current capabilities. In some cases, *non-invasive* health monitors can also be used to evaluate a system's health without adding sensors or making changes to existing vehicle hardware. Since our multi-vehicle testbed uses commercially available off-the-shelf (COTS) R/C hardware, sensors cannot be easily added without making intrusive modifications to the vehicle hardware, causing it to under perform. For this reason, a portion of our research effort is focused on developing and testing non-invasive health monitors with the COTS hardware used in the RAVEN.

Since the RAVEN is an indoor test environment, to date only electric-powered

vehicles have been used during autonomous testing as active vehicle components. One of the main issues in using electric-powered vehicles is that an electric-powered vehicle's endurance scales with motor power consumption and battery size, and thus battery weight (which has a negative effect on vehicle endurance). Although battery technology has improved in recent years (for example, a 3-cell, 11.1 V 1320 mAh Li-Poly battery from Thunder Power Batteries [92] is approximately 100 grams), an electric-powered air vehicle's flight time is largely impacted by the vehicle's lift capabilities. In addition, the average flight time of electric-powered helicopters and quadrotors (such as the Draganflyer V Ti Pro [24]) are limited by the types of motors used and desired payload capacity. Note that for all of the recharge test flights, two different versions of quadrotors were used: the Draganflyer V Ti Pro [24] and the X-UFO Quadrotor [39]. The Draganflyer quadrotors are COTS vehicles that use motors with brushes, while as of the writing of this thesis the X-UFO Quadrotor has just begun to enter into production for commercial use (April 2007) and uses brushless motors with an onboard gyro stabilization package for improved attitude control. All of the initial recharge testing results were performed with the Draganflyer quadrotors starting in June 2006, until recently when the X-UFO Quadrotor was incorporated into the test setup. The X-UFO design has proven to be more energy efficient, resulting in longer flights on one battery charge than a COTS Draganflyer quadrotor. Results for both vehicle types are provided in this chapter.

Past research into battery monitoring and state estimation has focused on using direct, invasive measurements of current flow and voltage level to calculate a battery's state of charge (SOC). Most of this research focuses on calculating SOC using complex analytical models of internal battery dynamics [77, 79]. However, these approaches require significant knowledge of battery properties and internal dynamics. Recent research has sought to simplify the construction of battery SOC models by using machine learning techniques using voltage and current measurements from the battery [41]. A learning approach can be advantageous because it does not require knowledge of internal battery chemistry and can be easily extended to multiple battery chemistries.

As part of our on-going research, the relationships between a vehicle's flight capabilities, power system health, propeller wear, battery charge and other parameters are being examined using the electric-powered quadrotor vehicles in the testbed [97]. Flight testing has demonstrated that many of these parameters can be evaluated while observing the vehicle in a simple hover. For example, data from these flight tests has shown there is a strong correlation between the battery voltage and the R/C

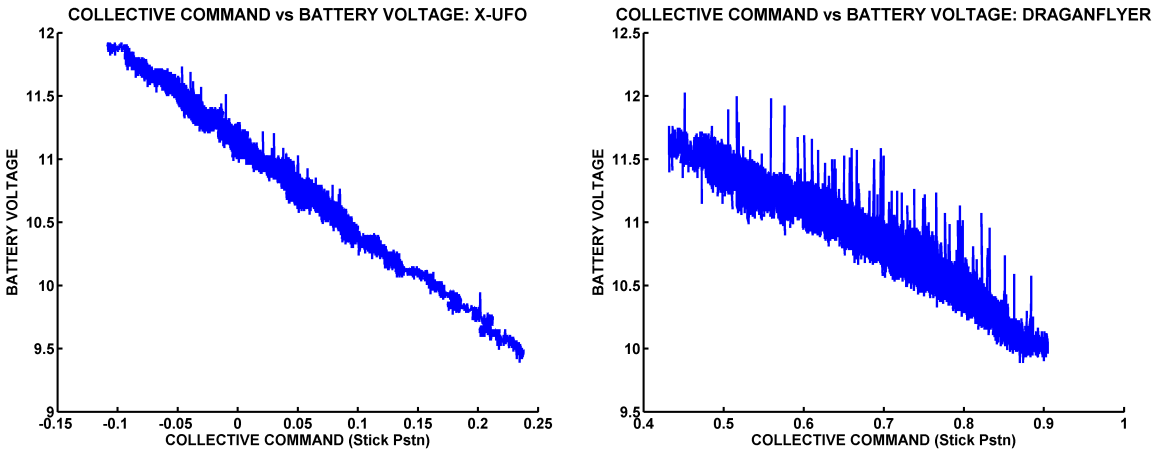
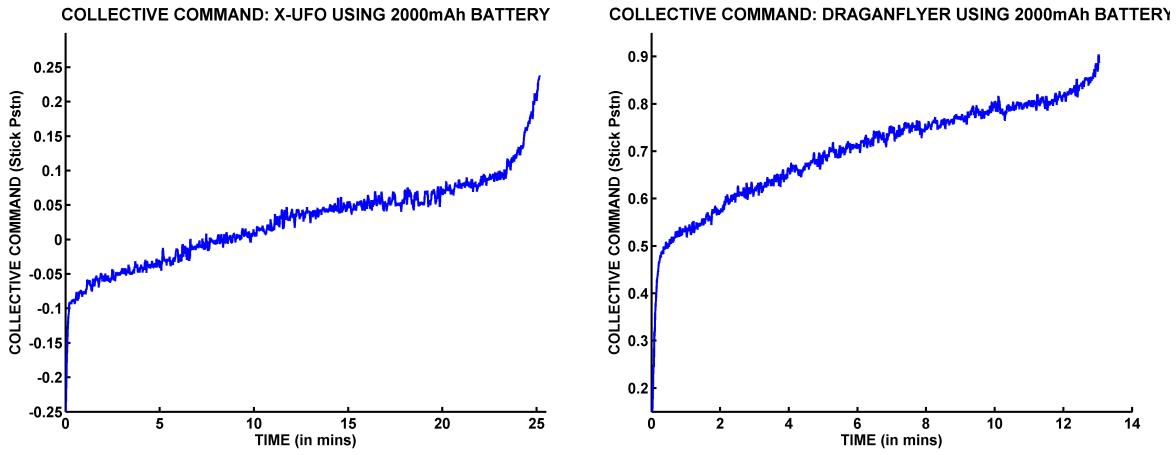


Figure 6-1: Comparison Between Battery Voltage and Collective Stick Position during a Hover Test: X-UFO using a 2000mAh Battery (left) and Draganflyer Quadrotor for a 1320mAh Batteries (right)

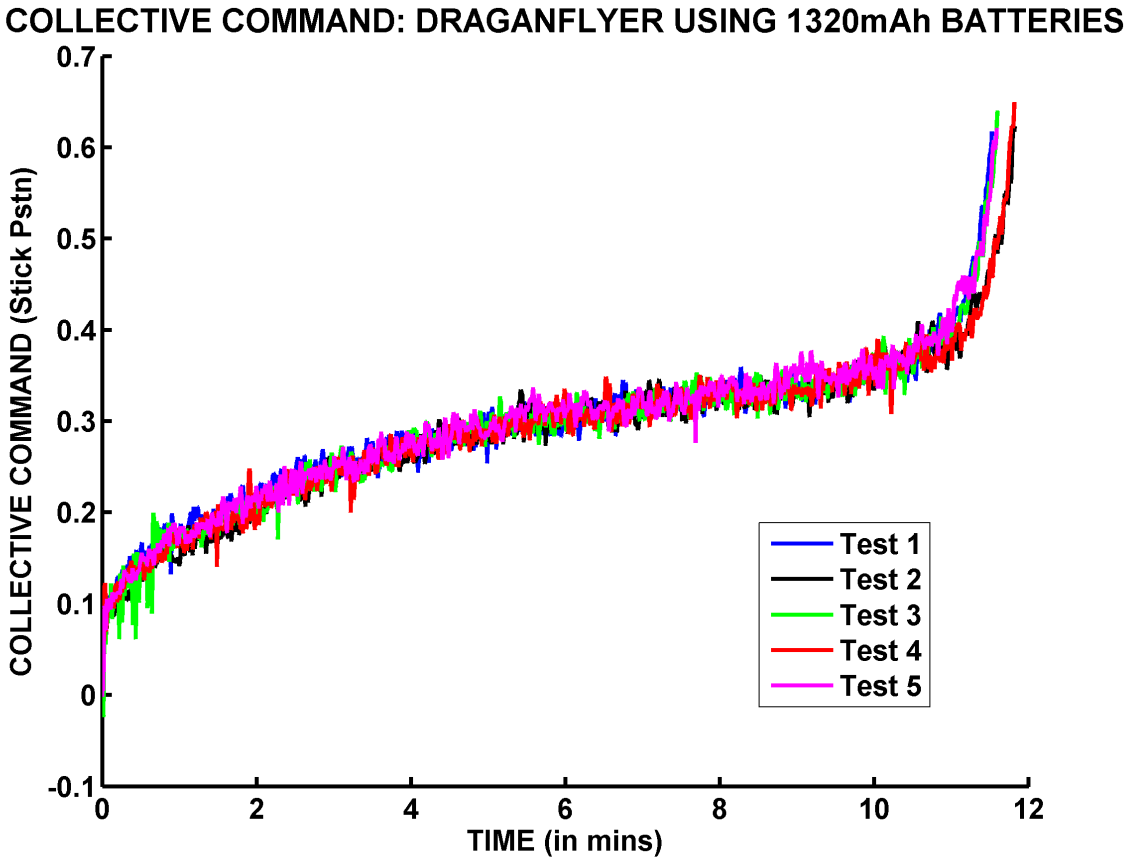
controller's collective stick position value. As the voltage of the battery decreases over time (due to battery use), the collective stick command increases. Typical results for the X-UFO and a Draganflyer with both the white plastic and black nylon blades and are shown in Figure 6-1. Note that the Draganflyer's new stock black nylon blades using a 2000 mAh battery produce slightly higher collective commands than the older white plastic blades (which were very brittle and cracked easily), but yield similar shaped collective command curves over the vehicle's flight time (as shown in Figure 6-1). Figure 6-2 shows that as the battery's charge level decreases, the average collective command must increase over time for the vehicle to maintain its current position. Note that for each quadrotor, the collective command increases rapidly initially during take-off and steadily increases almost linearly until the vehicle's battery begins to lose charge rapidly near the end of the flight.

Since our goal is to command and control multiple air vehicles over extended time periods, a reliable monitor that estimates a vehicle's flight time is a critical part of this research. Therefore, one of our main goals was to generate an estimate of the vehicle's remaining flight time *non-invasively* using the vehicle's altitude and collective input position. In other words, this battery monitor does *not* use sensors on the vehicle to generate this estimate of the vehicle's remaining flight time.

For this task, a support vector regression (SVR) model (based on real flight data) was generated using the vehicle's current altitude and collective position to estimate the vehicle's remaining flight time. The main idea behind using an SVR model is that a model is created (using experimental data) from input-output pairs that can



(a)



(b)

Figure 6-2: Collective Stick Position vs Time (in mins): (a) X-UFO and Draganflyer (with black nylon blades) using a 2000mAh Battery and (b) for Five Consecutive Hover Tests using a Draganflyer (with white plastic blades) and Different 1320mAh Batteries

be used to predict the output to a system given an input without explicitly knowing the model of the system [38, 88, 101]. This model of the form

$$f(x) \quad = \quad \sum_{i \in SV} (\bar{\alpha}_i - \bar{\alpha}_i^*) K(x, x_i) + \bar{b} \tag{6.1}$$

is used to generate a non-linear mapping into a higher-dimensional feature space of the form used to perform a regression on any data point $x \in \mathbb{R}^n$ [38]. Here, $K(x, x_i)$ is defined as the kernel function which generates the inner product in the high-dimensional feature space and $\bar{b}$ is a constant [101]. In addition, the weighting parameters $\bar{\alpha}_i^*$, $\bar{\alpha}_i$ for all $i \in SV$ for the ϵ-insensitive SVR problem are determined by solving the optimization problem [38]:

$$\begin{aligned} \max_{\alpha,\, \alpha^*} \quad & \sum_{i=1}^{r} \alpha_i^*(y_i - \epsilon) - \alpha_i(y_i + \epsilon) - \frac{1}{2} \sum_{i,j}^{r} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) K(x_i, x_j) \\ \text{subject to} \quad & \sum_{i=1}^{r} (\alpha_i - \alpha_i^*) = 0, \quad 0 \geq \alpha_i,\ \alpha_i^* \geq C \quad \forall i \in \{1, \ldots, r\} \end{aligned} \tag{6.2}$$

where $\{(x_1, y_1), (x_2, y_2), ..., (x_r, y_r)\}$ is the set of training pairs introduced into the optimization problem. Note that this model can be improved by regenerating the SVR model based on new data, thus ensuring that the model is up-to-date and representative of the current situation.

The battery model uses input vectors x that contain two pieces of information: the collective input to the quadrotor and the altitude difference between the vehicle's current and desired location. The output y generated by this model is the vehicle's predicted flight time remaining. Over 10,000 data points were used to generate the SVR battery model using the LibSVM software package [17]. Here, we used the epsilon state vector regression (epsilon-SVR) setting with the radial basis functions kernel-type. In addition, we changed the standard settings in the software so that the cost of constraints violation (C) is 200, enlarged the kernel cache size to 200 Mb, and set the epsilon parameter to 0.0005. This battery model was tested against actual vehicle hover flights as shown in Figure 6-3. Here, the predicted flight time generated from the model provides a reasonable estimate of the vehicle's remaining flight time. In fact, by filtering the vehicle's collective stick position, a better estimate of the vehicle's remaining flight time is generated. This model is currently being used by the mission manager to estimate the remaining flight time for airborne quadrotor vehicles during test flights.

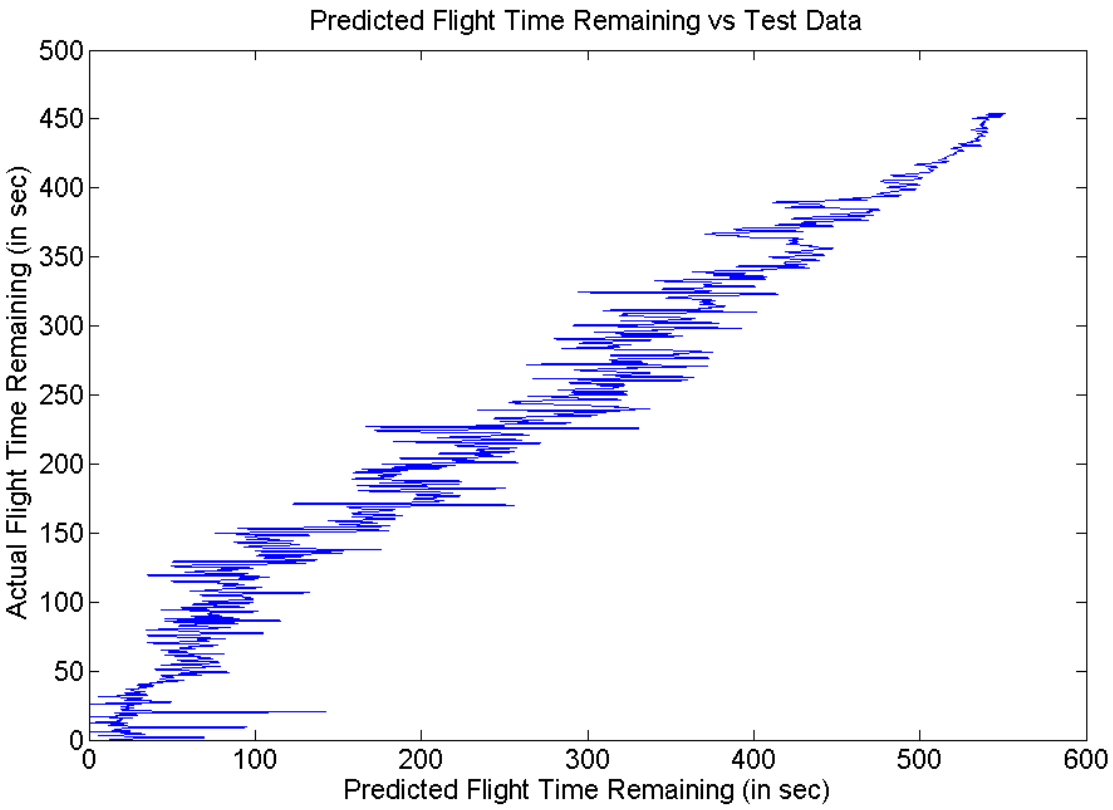
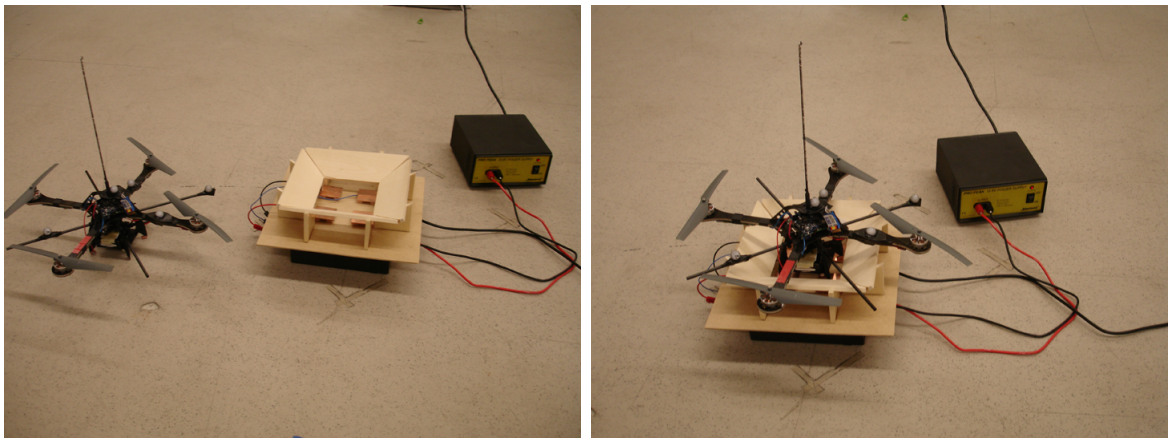


Figure 6-3: Predicted vs Actual Remaining Flight for a Hover Test

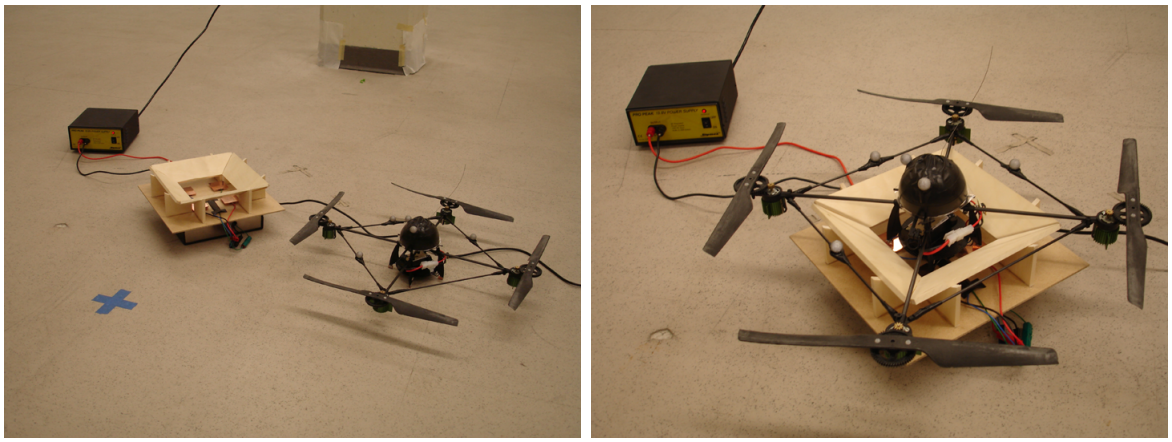
6.4 Battery Charging Station

Even as electric-powered autonomous vehicles and their support systems become smarter, they are fundamentally limited by the capacity of onboard batteries that power the vehicles. As described in the section above, autonomous health management hardware and software allow vehicles to determine the battery's current status and decided when a vehicle must land to replace or recharge itself before continuing its mission. Ground platforms have been developed to allow ground robots to recharge during operations. For example, at the University of Tsukuba, researchers constructed an autonomous ground vehicle and recharge system in order to facilitate autonomous ground vehicle navigation and control experiments [40]. The system was tested by running an autonomous vehicle nonstop for one week. During the week-long experiment, over one thousand recharge dockings were successfully accomplished. However, as of the writing of this thesis, we have not found an instance of an autonomous aerial docking and recharge before reporting it in [97].

Therefore, to conduct research into autonomous, multi-vehicle, persistent surveil-



(a)



(b)

Figure 6-4: Recharge Landing Pad Setups: (a) X-UFO and (b) Draganflyer

lance mission applications, an integrated autonomous recharge landing platform and system was designed. Shown in Figure 6-4, the goal of this system is to allow our aerial vehicles to autonomously recharge their batteries during extended flight operations. This recharge platform provides real-time information on the charge progress of the battery. This information can be used by the mission manager to monitor and estimate the vehicle's current health state during the charge process. The real-time battery data gathered during the recharge process and flight operations can be used to estimate the vehicle's projected flight time. This data provides a second estimate that can be used to predict when a vehicle should be cycled back to base during mission operations.

The recharge system consists of several components: battery isolation electronics, vehicle electrical contacts, landing pad, and ground recharge electronics. While this system is designed to recharge quadrotor vehicles, the electronics and software in

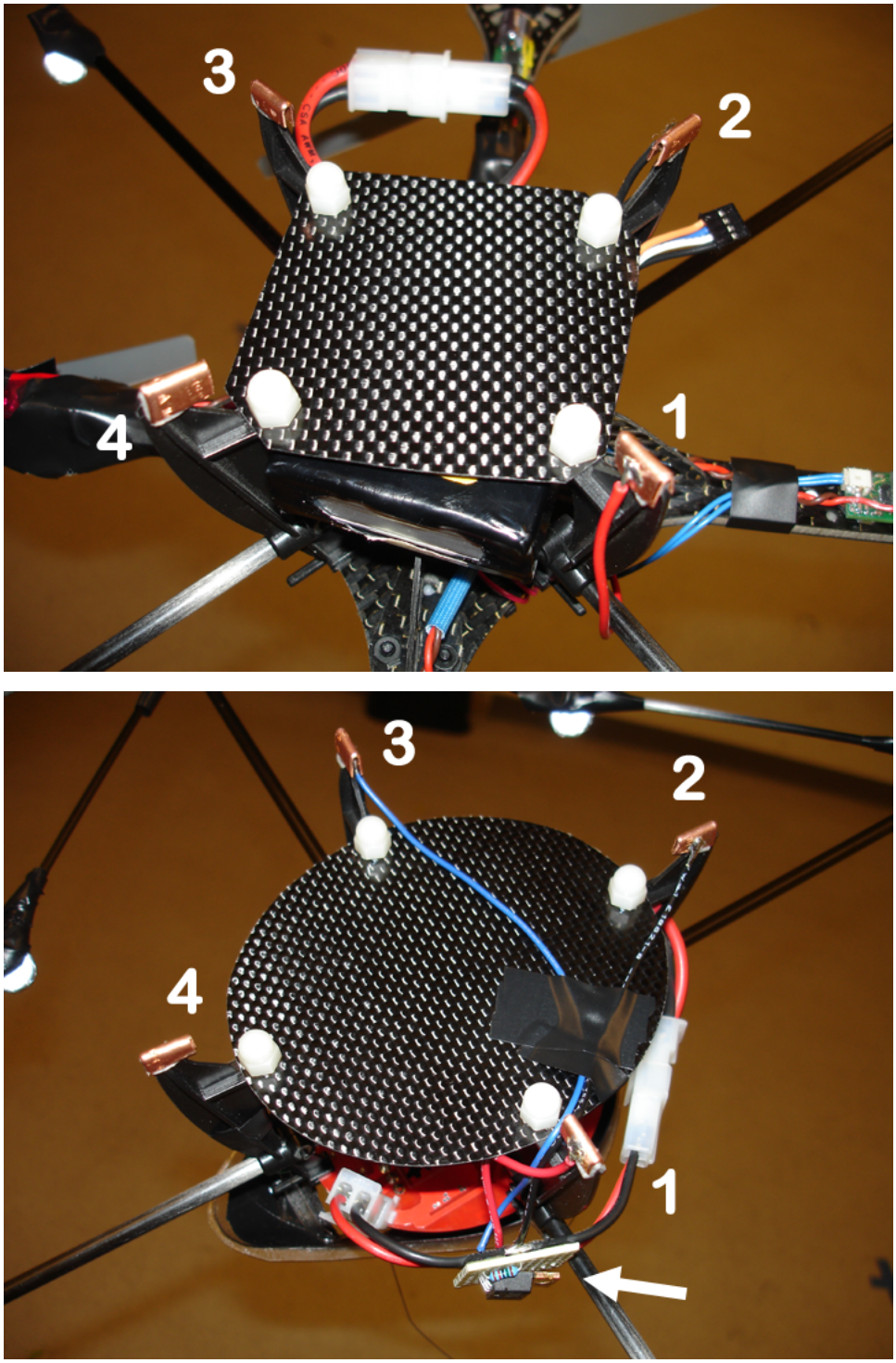


Figure 6-5: Vehicle's Electrical Landing Contacts: X-UFO (left) and Draganflyer (right). Note that in this figure the Draganflyer has a board attached to it between the pins and the recharger. This electronic setup allows the Draganflyer to be reactivated (i.e., toggle the Draganflyer's safety push button) after vehicle power has been turned on.

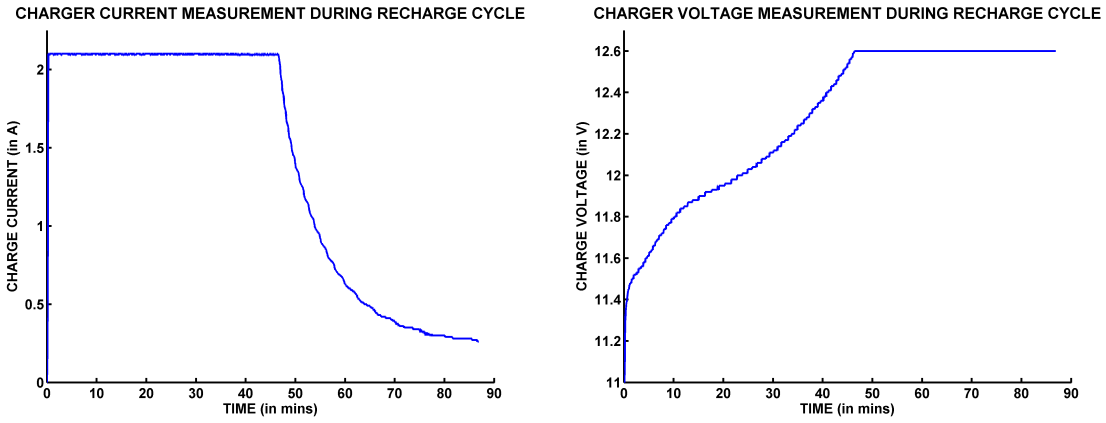


Figure 6-6: Li-Poly Battery Voltage Rebound before a Battery Recharge Cycle

the system can be adapted to other vehicle types, such as ground rovers and other R/C vehicles. In addition, the battery isolation electronics are used to prevent power-related issues during vehicle arrival to or departure from the recharge station. Finally, contact pads on the body of the quadrotor provide the necessary electrical contact to recharge the battery on board the vehicles. As shown in Figure 6-5, each leg (labeled 1 to 4 in the figure) of the quadrotor is outfitted with copper contacts. Electrical tape is used to insulate all contact pads from the carbon fiber quadrotor body. Note that in this figure the Draganflyer has a board attached to it between the pins and the recharger. This electronic setup allows the Draganflyer to be reactivated (i.e., toggle the Draganflyer's safety push button) after vehicle power has been turned on.

Once the vehicle lands in the charger, the vehicle's battery is usually warm from use during flight. As the battery cools, our testing has shown that the battery's voltage will rebound from its "motor's off" state. As shown in Figure 6-7, once the vehicle's motors are turned off, the battery's voltage recovers. Initial experiments have demonstrated that this recovery voltage varies from one vehicle to another due to the differing demands placed on each vehicle during flight. In the test show in Figure 6-7, we flew a vehicle past the flight time limit calculated by the battery monitor to determine if the system could still automatically charge the battery in the event of a flight time emergency. The results from this test show the battery's voltage rebounded considerably after the motors are turned off, thus the voltage measurement on the battery should be taken after the batteries are given a chance to cool down. Therefore, before we start recharging the batteries, we allow the battery to cool down for 5 mins to allow the batteries to reach a steady state before charging. Additional long-term testing is needed to better quantify battery life and the effect of starting a

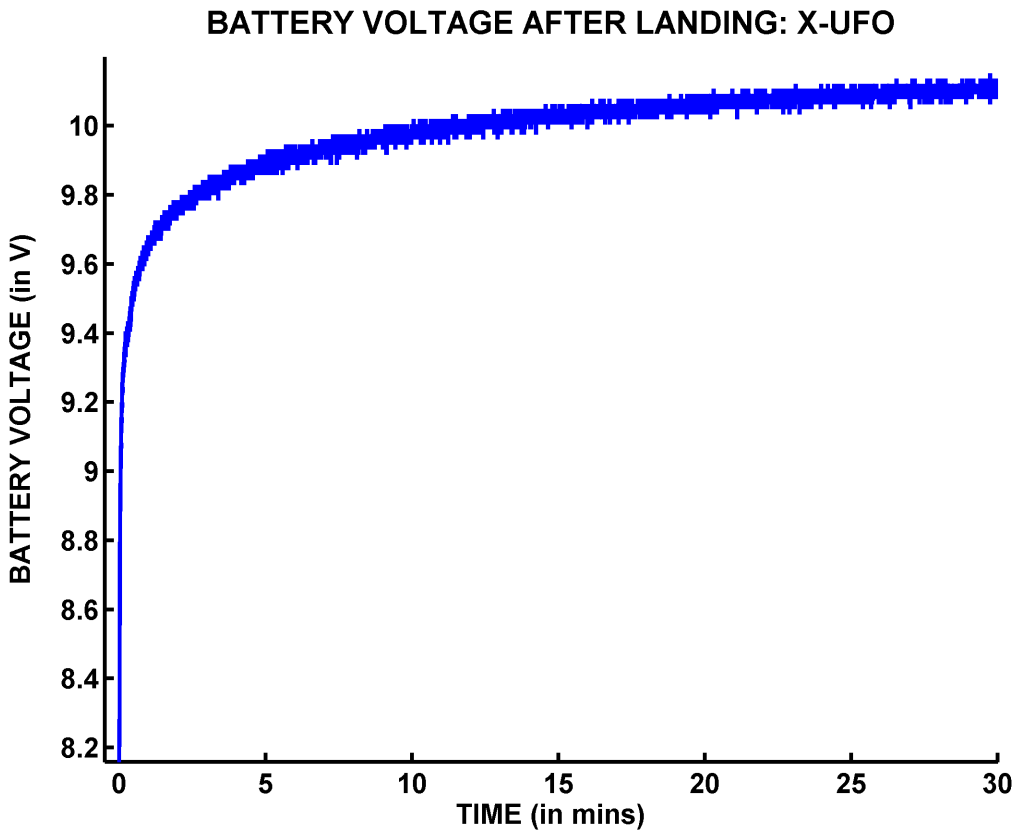


Figure 6-7: Example of Battery Voltage Rebound during Battery Cool Down

charge cycle early.

As with the vehicle when it is flying, we can use the battery's current and voltage measurements on the charger to both estimate the vehicle's remaining charge time and estimated flight time. Figure 6-6 show the voltage and current measurements take from the recharge platform during an automated flight test and recharge sequence. These plots are based on the charge's applied voltage and current to a 2000 mAh battery during a vehicle recharge cycle. Notice that the charger applies a set current (of 2.1 A in this case) until the battery approach 12.6 V. Once the voltage reaches 12.6 V, the charger begins to reduce the current applied to the battery in logarithmic fashion. From experimental testing, we have found that for short term uses of the battery (a single 24-hr test), we are able to end the charge cycle earlier to reduce the flight-maintenance cycle of the battery. However, additional long-term testing is needed to better quantify battery life and the effect of ending a charge cycle early using the current vehicle recharge system.

Using information from the recharger, we have developed a recharge model to

estimate when the vehicle's charging cycle will end. The battery model uses input vectors x that the charger's voltage and current inputs to the battery, and the output y generated by this model is an estimate on the number of minutes left in the vehicle's recharge cycle. Once again, the SVR recharge model was generated using the LibSVM software package [17] and the same parameters used to generate the battery model were used to generate the recharge model. This model provides an estimate of the length of time until the vehicle's charge cycle is completed.

Using this setup, the system has autonomously commanded a vehicle to take-off, fly a small mission, and then land in the recharge platform. Then, the system autonomously commands recharge platform to charge vehicle's battery, while monitoring its progress throughout the charge cycle. After the charging sequence has been completed, the system re-initializes the vehicle and begins the flight-maintenance sequence once again. Pictures of the vehicle system's first landing attempt performed by the system at the end of July 2006 are shown in Figure 6-8. Results from a January 2007 test where an Draganflyer quadrotor was commanded to land into the recharge platform are shown in Figure 6-9. In this test, the landing pad was placed at (0,5,0) m in the room and the vehicle was commanded to land after flying to and hovering above the landing location for five seconds. This waiting time is designed to allow transients in the x- and y-position controller to settle out before continuing with the landing decent. Note that once the vehicle begins its decent, the vehicle corrects its position as necessary to ensure that it can land on the recharging location.

After performing multiple single vehicle flight-recharge sequences, the mission management system was setup to fly a 24-hr, single vehicle flight-recharge test. Using the battery monitor and recharge system described above, the fully-autonomous 24-hr test single vehicle test was successfully performed (and video recorded) on March 31st, 2007. In this test, a single X-UFO quadrotor vehicle flew above its recharge platform between 10 and 13 mins before landing to recharge its batteries. This flight-recharge sequence occurred 21 times during the 24-hr period. This test was performed without any operator interaction and marks the first time in the literature where an air vehicle was able to fully-autonomously perform a routine maintenance activity (such as refueling) multiple times over a 24-hr period. In addition, this test validated the mission system's hardware setup for an extended vehicle mission by showing that mission system could autonomously manage the recharging and flight command operations without human input or interaction. As of the writing of this thesis, the shortened 2 min video version of the test is available at `http://vertol.mit.edu`.

As shown in Figure 6-10, the red line represents the recorded flight time of the

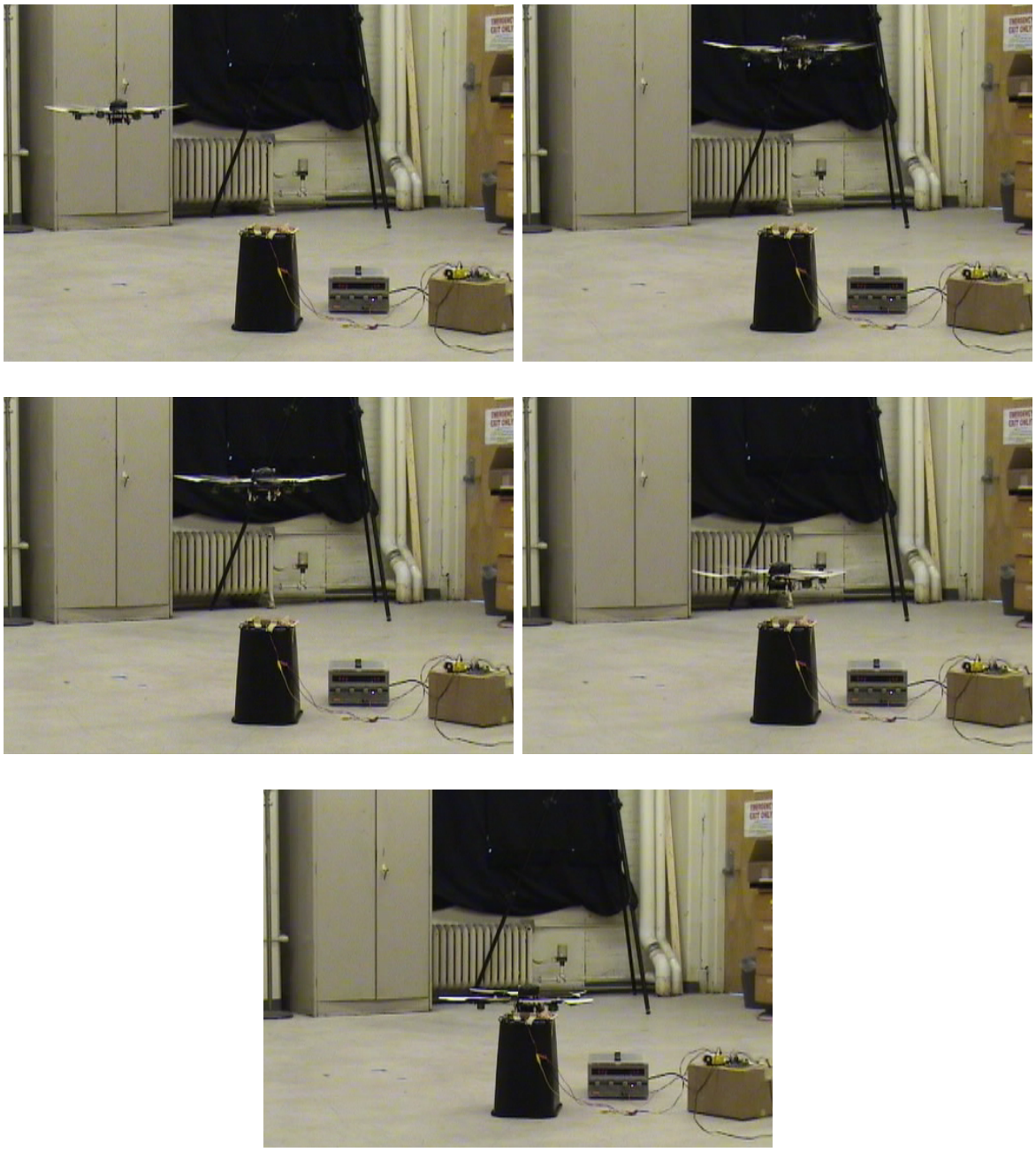


Figure 6-8: Automated Landing and Recharge using the MIT Indoor Flight Test Platform in July 2006

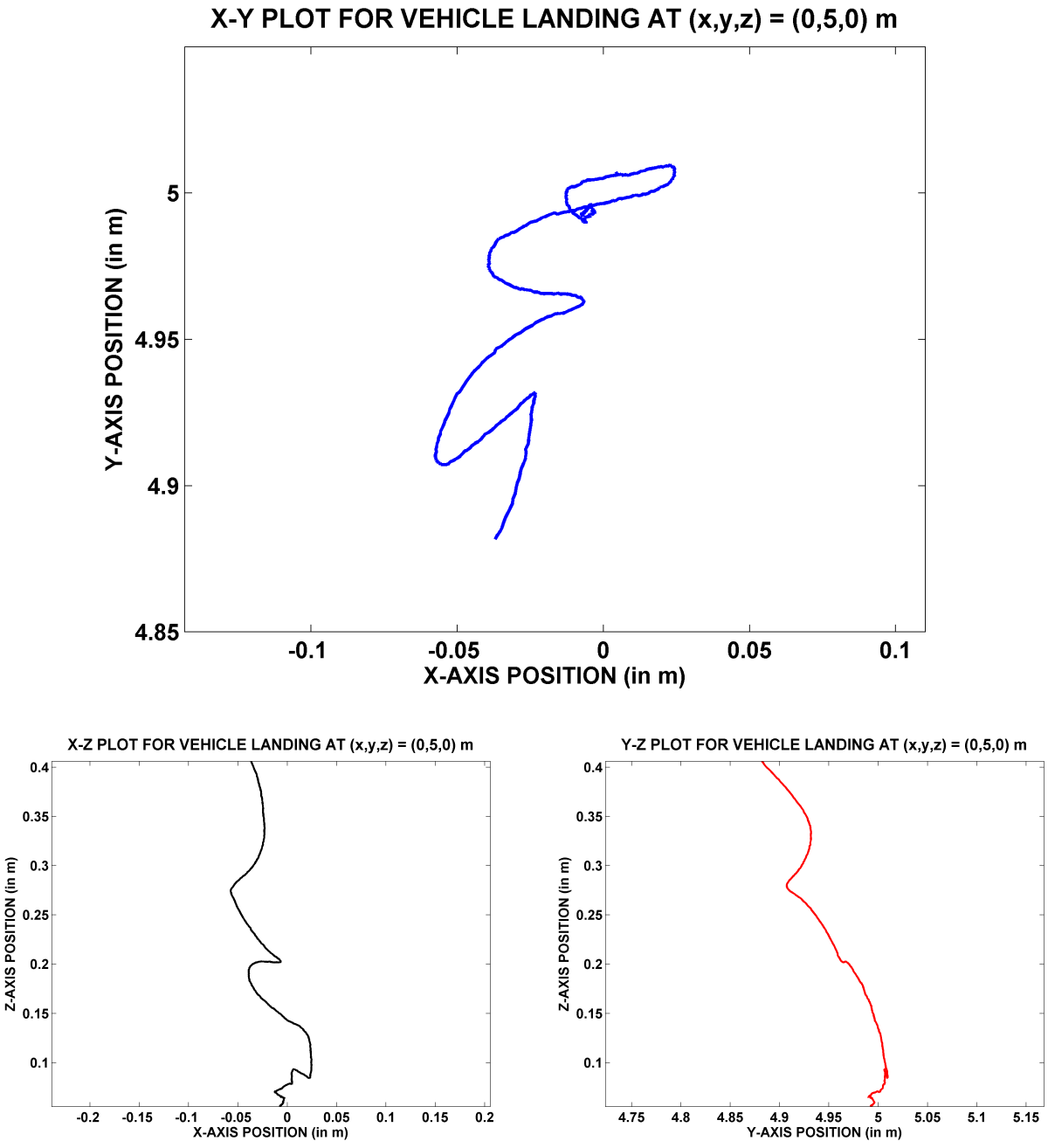
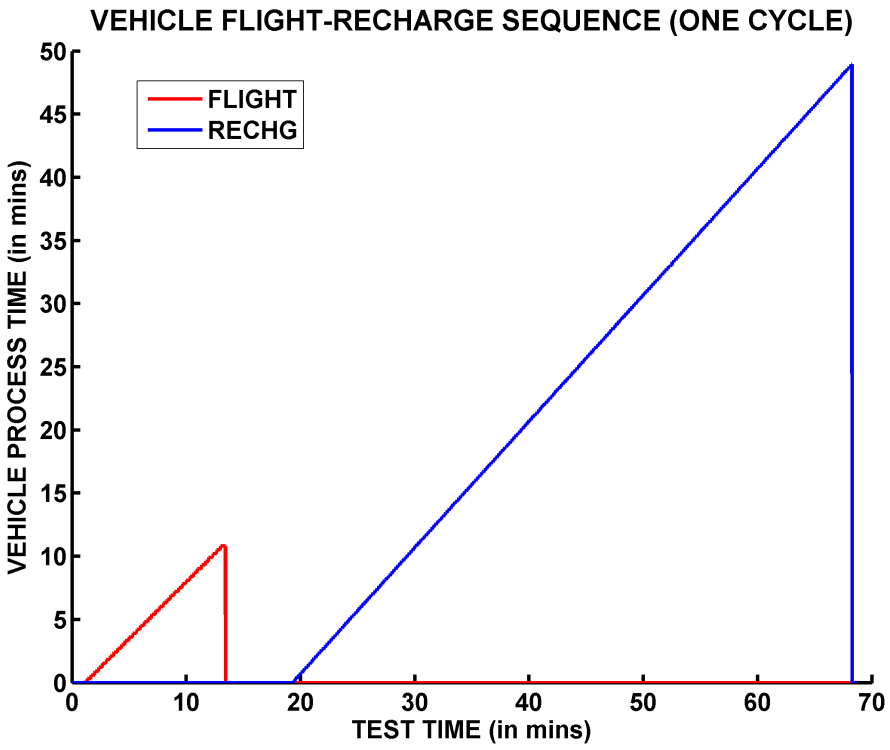
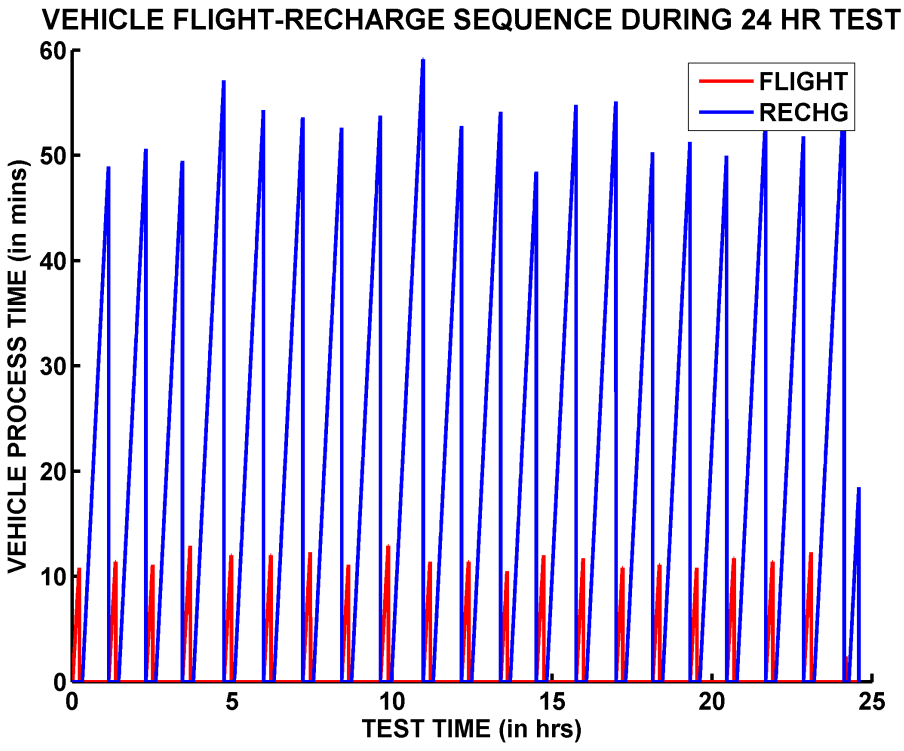


Figure 6-9: Landing in the Recharging Platform – the x-y (top), x-z (bottom left) and y-z (bottom right) plots for landing at (0,5,0) m. Note that as the vehicle descends into the landing platform, the vehicle translates in the x-y plane. This translation is mainly caused by the vehicle's propwash and ground effect as it approaches the landing pad.

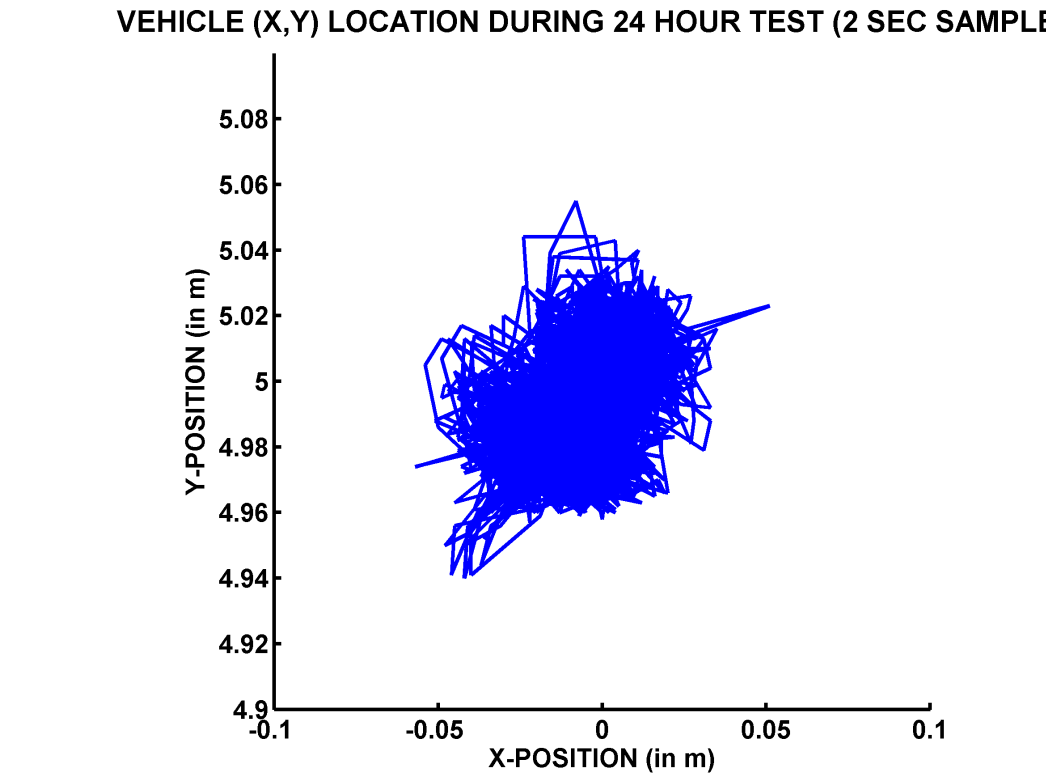


(a)

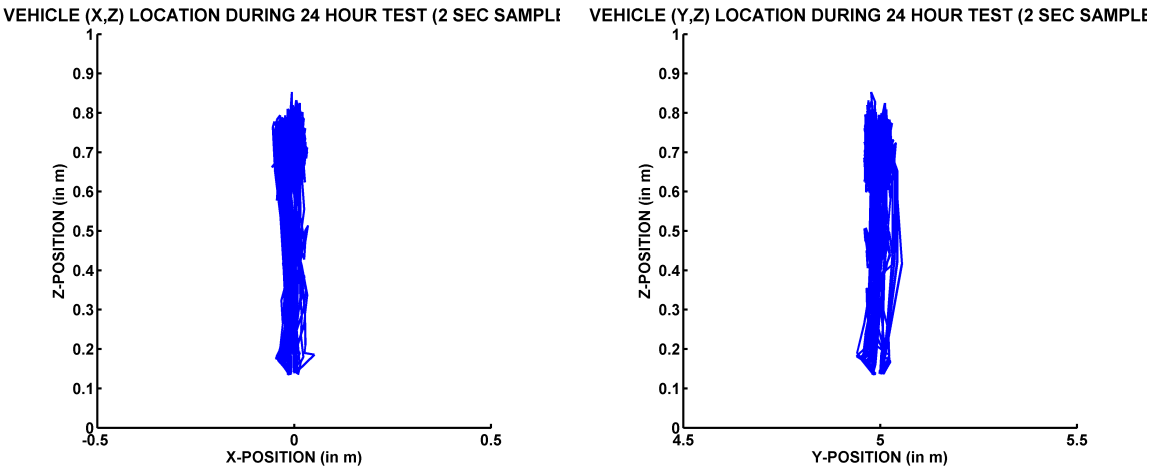


(b)

Figure 6-10: Fully-Autonomous Single Vehicle 24 hr Flight-Recharge Test using an X-UFO: (a) A Single Flight-Recharge Cycle and (b) 24 hr Flight-Recharge Test Cycle



(a)



(b) (c)

Figure 6-11: Fully-Autonomous Single Vehicle 24 hr Flight-Recharge Test using an X-UFO – Position Plots: (a) (X,Y) Position, (b) (X,Z) Position, and (c) (X,Z) Position. This data was saved by the mission manager at 2 second intervals over the 24-hr period. A time-lapse video of this flight can be found at http://vertol.mit.edu

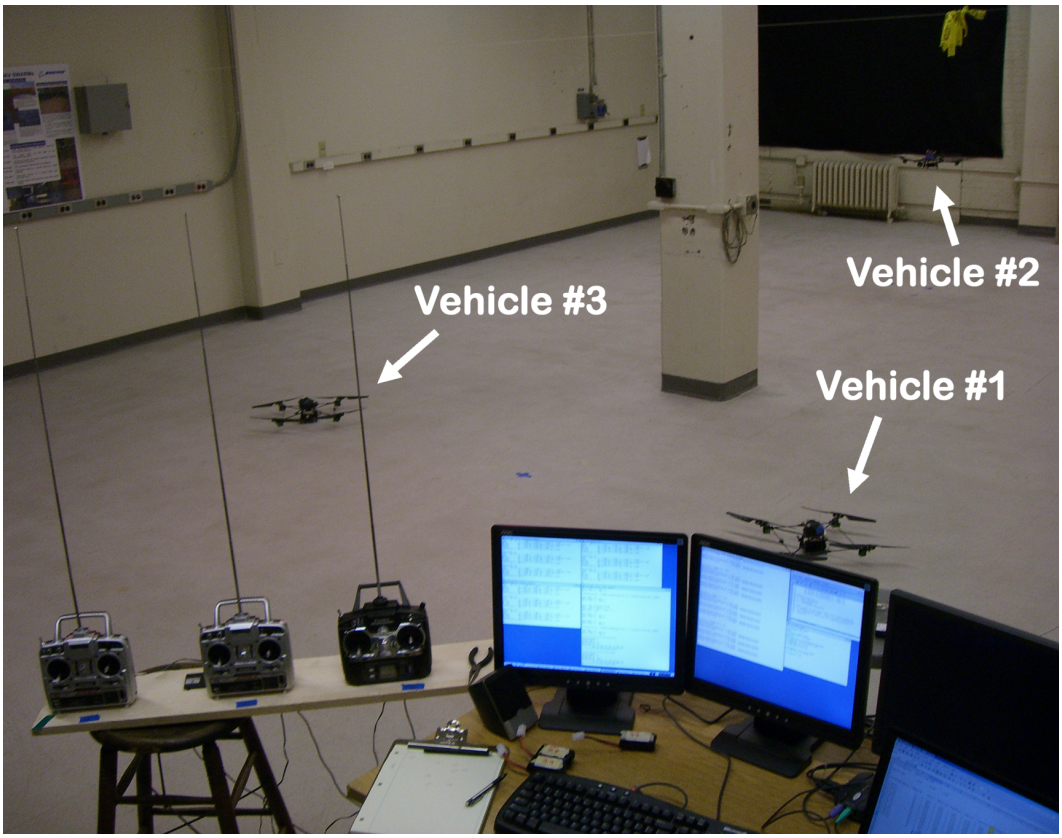


Figure 6-12: Automated 1.5 Hour Persistent Surveillance Mission (PSM) with Three Autonomous Vehicles

vehicle and the blue line represents the recorded recharge time of the vehicle during each flight-recharge cycle. Note that there is a 5 min wait period between the time the vehicle lands and when the recharge sequence begins to allow the battery to cool down before charging. In addition, Figure 6-11 shows the (x,y)-, (x,z)-, and (y,z)-position graphs for the 24-hr test. During the flight portion of the test, the vehicle was commanded to hover at (0,0,0.7) m above the floor. Notice that the vehicle's (x,y) position stays well within 5 cm of center during the flight portion of the test. In addition, notice that during landing, the vehicle uses the sloped sides of the landing platform to ease it into center when it is landing. Overall, the test demonstrated that this vehicle and maintenance system configuration could be used as a component of a larger mission system for an extended multi-vehicle test.

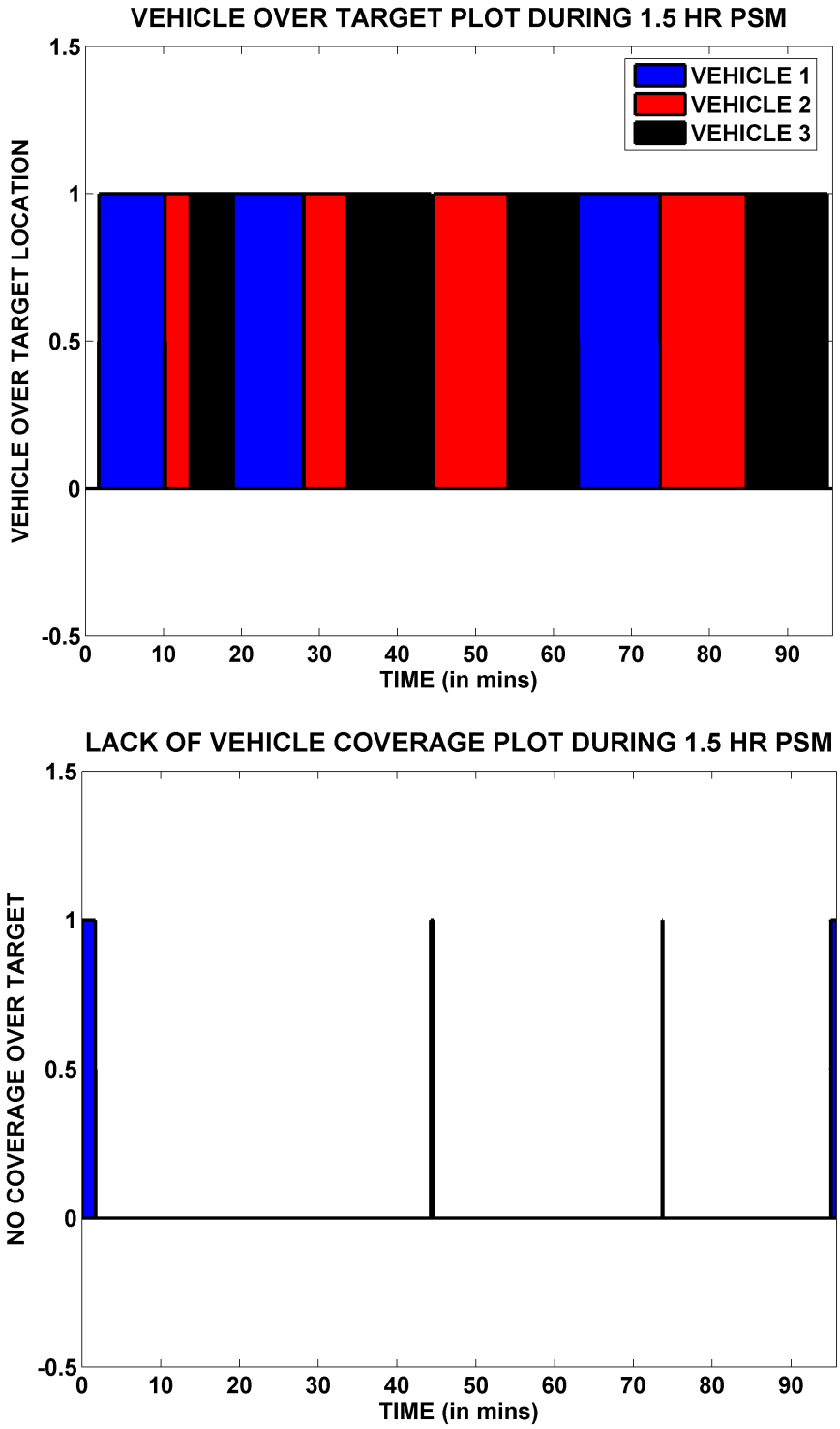


Figure 6-13: Automated 1.5 mission vehicle coverage plot showing the vehicles flying the mission (top), Lack of Coverage Time History during Test (bottom)

6.5 Mission Flight Test Results

Before attempting an autonomous mission system test using the automatic recharging stations, an maintenance operator-in-the-loop test was performed. Here, the maintenance operator was in charge of changing out the batteries of vehicles after a flight. Using the battery monitors described above, a 1.5 hour persistent surveillance mission with three vehicles was setup in the laboratory as shown in Figure 6-12. In this test, the tasking system was responsible for tasking the vehicles to take-off fly to the surveillance location and replace vehicles their batteries were depleted. In addition, in this test the operator was responsible for changing out the vehicle's batteries. Therefore, the tasking system was also responsible for directing the operator to change the batteries of vehicles that had just landed and acknowledging the operator was not in the flight area before taking off a vehicle. This acknowledgement was a safety feature to ensure that the tasking system would not take-off any vehicle in the battery maintenance area while the operator was in the space.

Data from this test is shown in Figure 6-13. During the 93 min test, there were two vehicle failures, both occurring with Vehicle #1. Approximately 45 mins into the test, the tasking system commanded Vehicle #1 to take-off, however the vehicle was unresponsive. As a result, since the tasking system did not see a response from Vehicle #1, it then commanded Vehicle #2 to take Vehicle #3's place. This delay resulted in approximately 25 secs lack of coverage over the surveillance site. In addition, after the next round of rotations, approximately 73 mins into the test Vehicle #1's low-battery alarm went off as Vehicle #2 approached the surveillance location, resulting in a 3 sec lack of coverage.

Overall, over the 93 min period using the heuristic described in Section 6.1, over 99.5% of the overall test a vehicle was over the surveillance area. This test showed that the mission manager could command and control the vehicles effectively during an extended mission. However, since each vehicle was deployed with a 1320 mAh battery, the vehicles averaged about 9.5 mins of flight time. As a result, the operator activity per vehicle cycle time was about 1.8 mins on average. In this test, operator was responsible for replacing batteries, charging batteries, resolved vehicle failures and acknowledging the mission manager for vehicle take-off commands (to prevent the vehicles from taking off while operator manually changed batteries).

Figure 6-14: Automated 1.5 Hour Persistent Surveillance Mission (PSM) using Mission Manager with Basis Function Generation Algorithm with Five Autonomous Vehicles

6.5.1 Mission Flight Test Results using the Basis Function Generation Algorithm

Following this test, the system was setup to perform a second autonomous multi-vehicle mission test. In this test, the mission manager calculated the best policy using basis functions generated by the Basis Function Generation Algorithm in Table (3.1). Figure 6-14 shows the test setup where there were five vehicles: two Draganflyer vehicles and three X-UFOs. Note that in this picture, one of the X-UFOs is flying in the surveillance area, while the other four vehicles are either in recharge mode or waiting to be commanded. The two Draganflyers in this test had a maximum flight time of 13 mins on a new battery, and the X-UFOs had a maximum flight time of approximately 20 mins on a fresh battery. In this test, the mission manager commanded each vehicle to take-off from their base location (in the northern half of the room) and hover in the center of the southern portion of the room.

In this test, each X-UFO was outfitted with a recharging station, and each Dra-

ganflyer was setup for the operator to change the batteries manually. The main reason for this test setup is as follows. Each battery using the automatic recharging platform takes approximately 70-90 mins to complete a recharge cycle (the five minute cool down period plus battery charging time due to flight use). Since only five vehicles (two Draganflyers and three X-UFOs) were available for the test, each vehicle had no more than 60-70 mins to complete its recharge before having to take-off again to perform surveillance – assuming that there are no vehicle failures or problems during the flight. Therefore, it was decided that the batteries for the two Draganflyer vehicles would be manually changed during the test, while the three X-UFOs would use the automatic recharging stations to recharge their batteries. In addition, the flight times for both Draganflyer vehicles varied based on the condition of the vehicle system. For example, since these vehicles had been flown many times before this test, the number of flight hours on each vehicle varied between 7 and 13 mins with different batteries. Likewise, each X-UFO during testing also showed variations in flight time for new batteries. Therefore, different battery monitors were used for each vehicle and vehicle type.

In this test, the mission manager's primary goal was to maintain one vehicle in the southern end of the room for surveillance purposes at all times. Since the vehicles had varying recharging and flight times, the problem formulation was adjusted to place a lower cost on using Draganflyers when they were available (since the batteries could be changed by the operator) and to always ensure that at least one vehicle was commanded to the surveillance area. For this problem, the state $x \in S$ is defined as the vector $x = (z_1, z_2, z_3, z_4, z_5, h_1, h_2, h_3, h_4, h_5)$, where z_i indicates the task to which each agent is allocated, h_i indicates each agent's maintenance/health state, and S is the state space for the problem. In addition to these states, a "demonstration end" state $\mathcal{O} \in S$ was added to this particular mission management problem setup to ensure that when the flight demonstration was over, the mission manager would command all of the vehicles back to base with probability one. Next, each action $a \in A_x$ is defined as the vector $a = (a_1, a_2, a_3, a_4, a_5)$ where a_i indicates the system's desired allocation for agent in the task space and A_x is the action space associated with the state $x \in S$. Each state x will transition under action a to the future state $y \in S$ with probability $P_a(x, y)$. Note that the agents in this problem can experience failures that cause them to be unavailable. In addition, agents are available for flight operations for a limited period of time (because of fuel, failure and maintenance concerns).

Finally, a local cost for being in state x under action a is defined by the function

$g_a(x) \in \mathbb{R}$. The cost for this problem was setup to penalize the system for not having a vehicle airborne and for having more than one vehicle airborne in the surveillance area at any given time. The idea behind this cost is that when the vehicles are cycling in and out, one vehicle is waiting on the other. Therefore, once the other vehicle is in the surveillance area, the second vehicle will be leaving – and hence, not performing surveillance. In addition, to promote a pro-active health strategy, the system was penalized for having vehicles in the surveillance area in a warning health state. In this problem, there were four health states: good, fair, warning, and maintenance. A good health state was associated with a good battery with at least 3 mins of flight time remaining. A fair health state was associated with a battery with between 3 and 1.5 mins of flight time remaining. A warning health state was associated with under 1.5 mins of battery life and the the maintenance state represented the vehicle in recharge / maintenance. Finally, a cost penalty was added to actions where the system selects a "good" X-UFO over "good" Draganflyer from the base location to fly to the surveillance area. The reason for this penalty was to encourage the system to cycle Draganflyers more often, thereby giving the X-UFOs enough recharge time on the recharge pads in between flights. Using this problem formulation, the basis function generation algorithm in Table (3.1) was used to generate the multipliers that were used by the mission manager to generate the approximation of the cost-to-go function for this problem in real-time.

Using this cost structure with the battery monitors described above, a 1.5 hour persistent surveillance mission with five vehicles was setup in the laboratory. Just as in the previous 1.5 hour test, the mission system was responsible for commanding the vehicles to take-off fly to the surveillance location and back. However, in this test, three of the vehicles were designed to autonomously recharge, while two of the vehicles needed an operator maintenance action upon landing in the same manner as the previous test. The three X-UFO vehicles were fully-autonomous throughout the demonstration, and the Draganflyer vehicles used in the demonstration were fully-autonomous with the exception of the operator action to change out their batteries after landing. Note that in Figure 6-17, a Draganflyer (right) returning from the surveillance area and an X-UFO (left) flying to the surveillance area use their collision avoidance algorithms to avoid one another on their way between the surveillance base areas. The collision avoidance based on a potential function-based method that is briefly described in Ref. [11].

Data from this test is shown in Figures 6-15 and 6-16. During the 90 min test, there were multiple vehicle failures. As shown in Figure 6-15, Vehicle 1x was commanded to

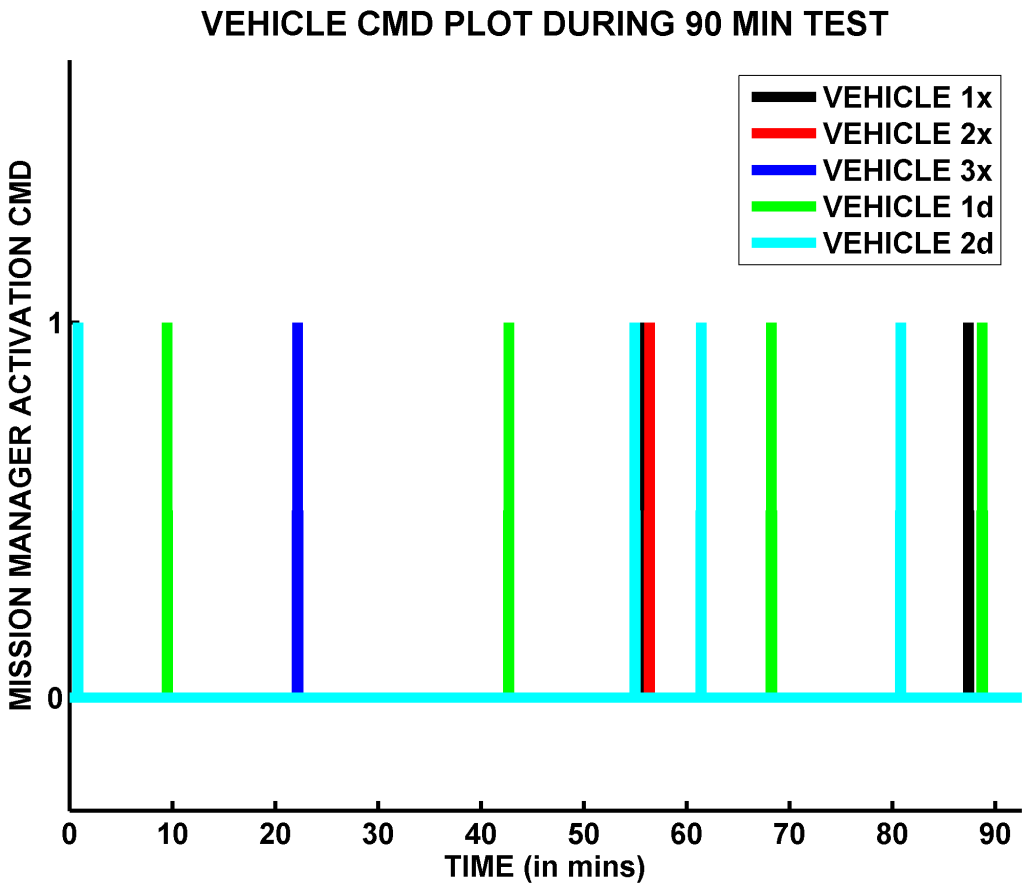


Figure 6-15: Automated 1.5 Hour Persistent Surveillance Mission (PSM) Mission Manager Commands using Mission Manager with Basis Function Generation Algorithm with Five Autonomous Vehicles

take-off twice over the 90 min test, but the vehicle had trouble leaving the recharging station and never was able to take-off during the entire test. The first time Vehicle 1x was commanded to take-off occurred at 55 mins when Vehicle 1x was commanded to take over for Vehicle 2d because it was unresponsive to the mission manager command. Since Vehicle 2d was not re-activated (due to operator error after the vehicle's battery was replaced), the vehicle never left the base area when commanded. After Vehicle 1x failed the first time, , the mission manager commanded Vehicle 2x to take Vehicle 1d's place. However, since the mission manager was proactively commanding vehicles to replace flying vehicles based on their estimated remaining flight time, ut commanded vehicles to take-off early enough such that the vehicle failures at the base station and air vehicle problems due to unexpected flight-related issues only resulted in a 10 sec gap in coverage. At approximately 63 mins into the test, Vehicle 2d was reactivated

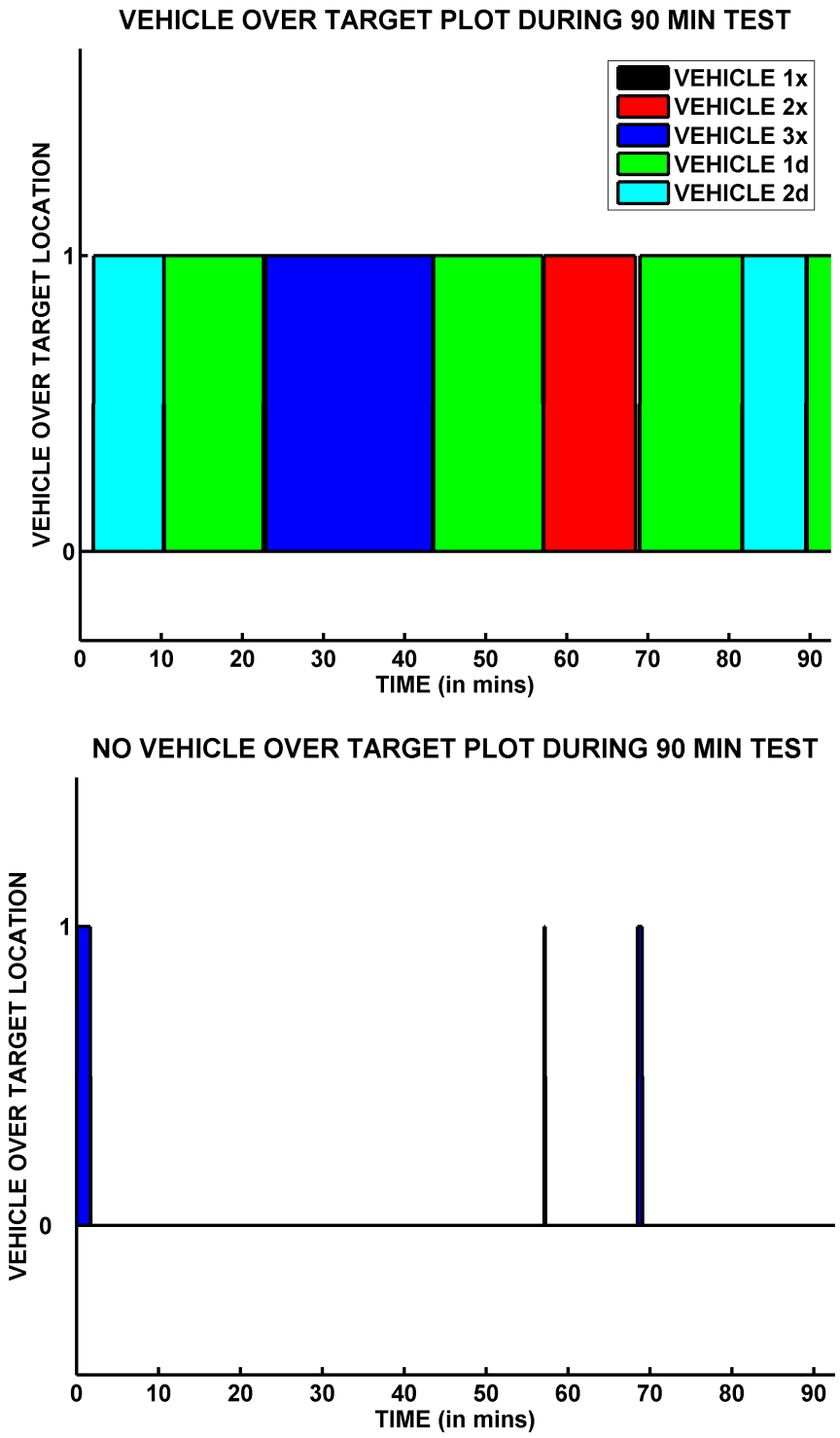


Figure 6-16: Automated 1.5 Hour mission vehicle coverage plot using Mission Manager with Basis Function Generation Algorithm with Five Autonomous Vehicles showing the vehicles flying the mission (top), Lack of Coverage Time History during Test (bottom)

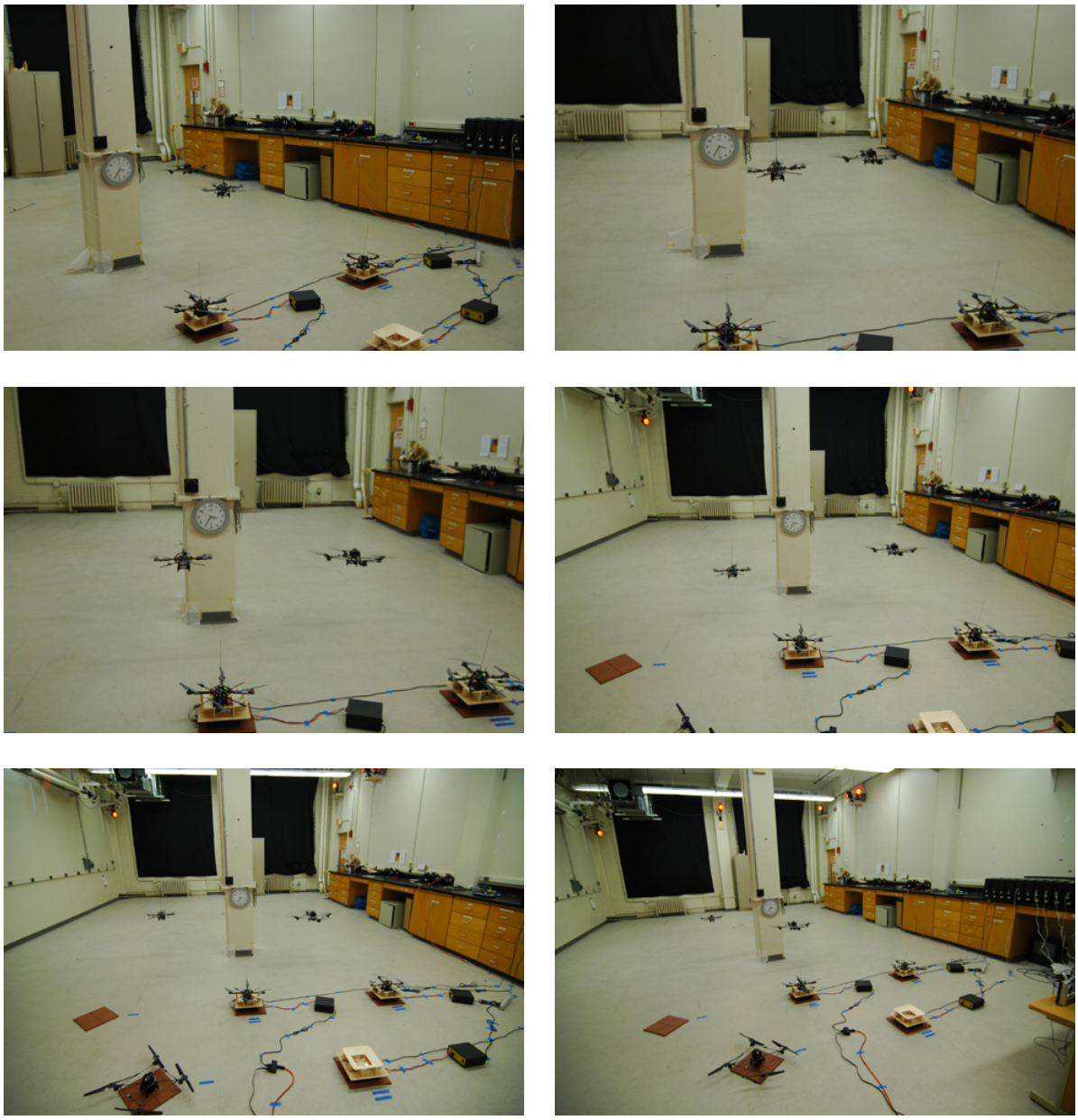


Figure 6-17: Two vehicles (Draganflyer on the right, X-UFO on the left) using collision avoidance detection via Vicon system measurements to ensure that vehicles can move between surveillance region and base region safely

by the operator and tested to make sure that it could take-off using the operator's manual override. Next, at approximately 68 mins into the test, Vehicle 2x's battery alarm detected a low-battery condition causing the vehicle to come back to base earlier than expected. Since this failure was not expected and due to an unexpected vehicle issue, the mission manager reactively caused Vehicle 1d to take-off and replace it in the surveillance area – resulting in a 30 sec gap in surveillance coverage. Finally, at approximately 90 mins Vehicle 1x was commanded a second time to take-off in order to replace Vehicle 2d, however once again, it was unable to clear the ground recharge platform. Hence, the mission manager commanded Vehicle 1d to fly in its place (since its battery was replaced by the operator less than a minute earlier). Since the mission manager was able to command Vehicle 1d before Vehicle 2d's battery alarm reached the warning state, there was no loss in coverage due to this failure.

Overall, even with all of the vehicle issues that occurred during the test, there was approximately a 40 sec gap in coverage after the first vehicle reached the surveillance area, resulting in over 99% coverage during the test. As seen in Figure 6-16, the policy generated using the basis function algorithm's cost-to-go function performed well despite all of the ensuing failures. This test marked the first time in the literature that a basis function generation method has been used to manage resources in a real-time, hardware-based autonomous system.

After performing this 1.5 hr test, the same mission manager was used to manage a 6 hour persistent surveillance mission with same five vehicles in the laboratory. Just as in the 1.5 hour test above, the mission system was responsible for commanding the vehicles to take-off fly to the surveillance location and back. Once again, three of the vehicles were designed to autonomously recharge, while two of the vehicles needed an operator maintenance action upon landing in the same manner as the previous test.

Data from this test is shown in Figures 6-18 and 6-19. Once again, there were multiple vehicle failures during the 375 min test. As shown in Figure 6-18, at approximately 25 mins into the test, Vehicle 3x was commanded to take-off, but the vehicle failed and did not leave the recharging station. In fact, this failure resulted in the loss of the vehicle for the entire test. Therefore, the system commanded vehicle Vehicle 1x to fly in its place. In addition, at about 40 mins into the test, the Vicon positioning system's Tarsus software froze unexpectedly. This caused the hovering Vehicle 2d to fall out of the air and crash. It took the operator approximately 2.6 mins to re-initialize the Vicon system and retrieve the crashed air vehicle. After the system came back up, the mission system tried to command vehicle Vehicle 2d back to the air, but instead commanded Vehicle 1d after Vehicle 2d was deemed unresponsive

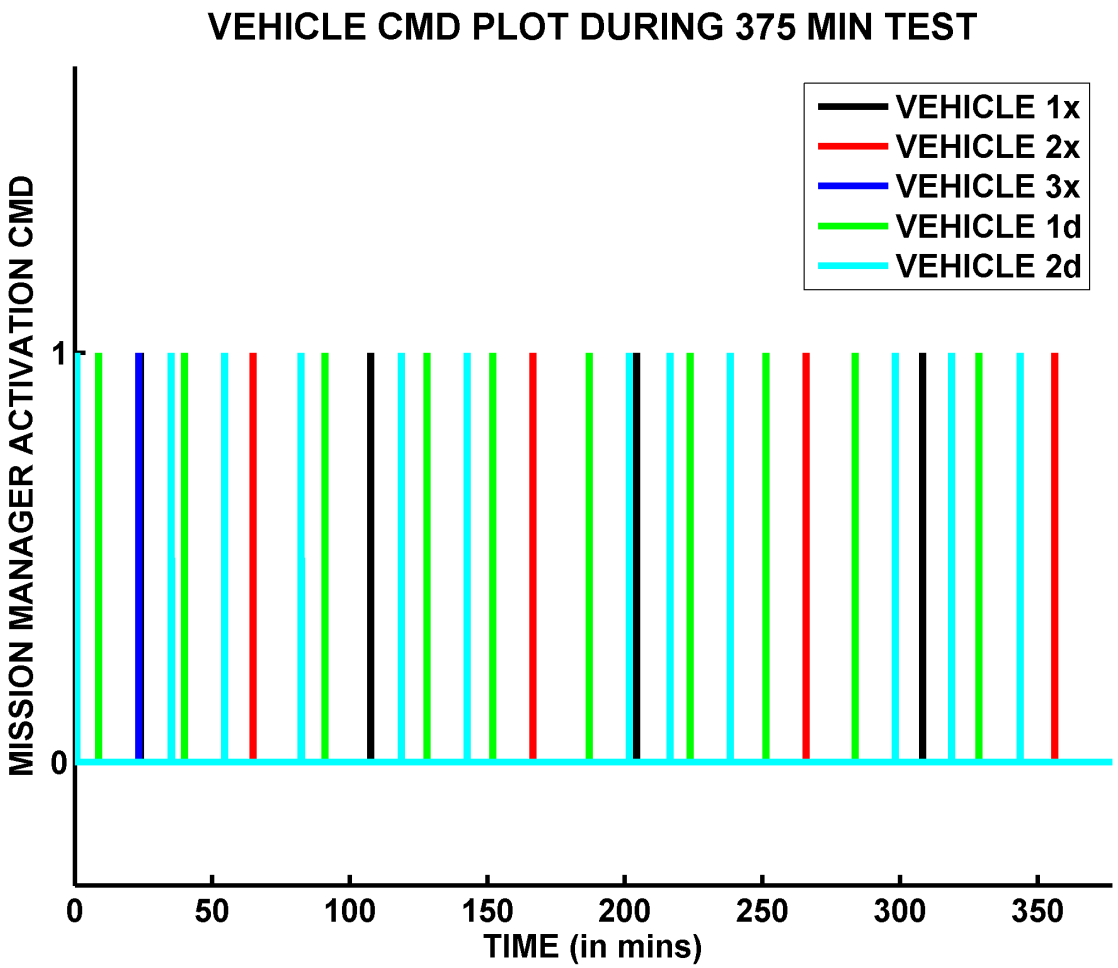


Figure 6-18: Automated 6 Hour Persistent Surveillance Mission (PSM) Mission Manager Commands using Mission Manager with Basis Function Generation Algorithm with Five Autonomous Vehicles

(since the operator was repairing it from the crash). In addition, small gaps in coverage occurred due to battery alarms, other vehicle failures, and air vehicle's taking alternate flight paths to avoid an incoming vehicle to base.

Overall, even with all of the vehicle issues that occurred during this span of time, there was approximately a 5.83 min gap in coverage after the first vehicle reached the surveillance area, resulting in over 98.4% coverage during the 6 hr test. As seen in Figure 6-19, the policy generated using the basis function algorithm's cost-to-go function performed well despite all of the ensuing failures. This test once again showed that the basis function generation method was capable of being used to generate policies that could manage resources in a real-time, hardware-based autonomous system.

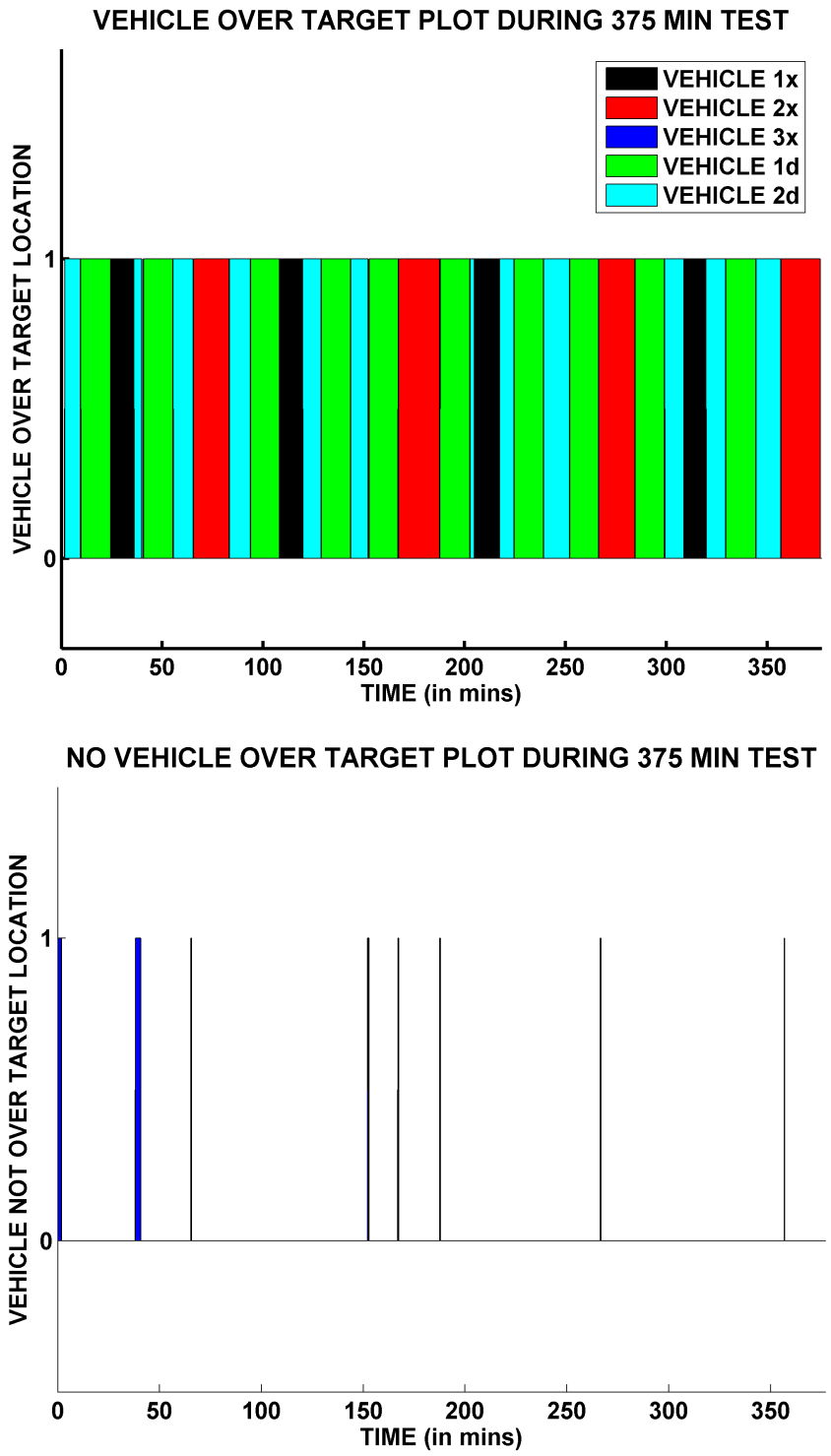


Figure 6-19: Automated 6 Hour mission vehicle coverage plot using Mission Manager with Basis Function Generation Algorithm with Five Autonomous Vehicles showing the vehicles flying the mission (top), Lack of Coverage Time History during Test (bottom)

In addition, to the best of our knowledge this is the longest autonomous multi-vehicle air vehicle mission where an autonomous aerial docking and recharge was used to maintenance vehicles autonomously in the literature. The flight tests in this section along with the large-scale simulation results presented in Section 4.2.3 mark a large step in the development of on-line cost approximation structures used to manage autonomous systems in real-time.

6.6 Summary

In summary, health management techniques can be used to improve mission-level functional reliability through better system self-awareness and adaptive mission planning. The section presents results and examples that demonstrate how health management information is being used to improve the mission system's self-awareness and adapt vehicle, guidance, task and mission plans so that an autonomous mission manager can command and control multiple autonomous UAVs over extended time periods. These algorithms, which determine the health of each mission component in real-time, have been successfully implemented and tested. In addition, these algorithms were used in part with vehicle maintenance hardware to enable the first fully-autonomous flight-recharge 24 hr test by a single vehicle. These, and other health management algorithms for each component, have been shown to improve strategic and tactical level decision making in autonomous mission systems while observing the impact of likely failures and maintenance needs for extended mission scenarios.

Chapter 7

Conclusions and Future Work

In conclusion, this thesis presents the development and implementation of approximate dynamic programming methods used to manage multi-agent systems. The main goal of this thesis is to develop, implement, and test methodologies that can be used in real-world applications to manage teams of autonomous vehicle systems in extended mission operations. To meet these goals, this thesis begins by presenting the real-time, multi-agent mission problem formulation and a system architecture designed to allow an autonomous mission system to manage multi-agent teams in a real-time environment in Chapter 2.

In Chapter 3, this thesis presents a method designed to enable an automated system to generate polices for large-scale problems in real-time. This method is designed to automatically generate an approximate cost structure that can be used to find policies for a given Markov Decision Process (MDP). This is a fundamental question related to use of approximate dynamic programming in real-time systems. As stated in Chapter 3, users normally select basis function parameters for a given MDP based on experience. However, the basis function generation algorithm posed in Table (3.1) implicitly generates basis functions for an approximate linear program by storing only multipliers r_0, r_1, ... r_N. As a result, the algorithm does not require a systems to save large amounts of information to generate the basis functions implicitly. In addition, since new basis functions are generated using sampled trajectories, the algorithm is designed to allow users to distribute computations over networked resources, thereby resulting is a viable algorithm for real-time use. A numerical complexity result, a proof showing the convergence of the algorithm without trajectory sampling, and an error bound comparing the optimal solution to the approximate solution using the basis function generation algorithm for a single update were also presented.

In Chapter 4, this thesis presents an mission management problem formulation that accounts for vehicle failures and system health concerns. This problem formulation allows an autonomous system to implement and solve the multi-agent mission planning problem in real-time using real hardware. In fact, flight test results using the cost structure calculated using the basis function generation algorithm are provided in Section 6.5.1 showing that the algorithm's cost function can be generated and used in real-time with success. Likewise, test results comparing the basis function generation algorithm's cost structure to the optimal cost for a small-scale problem are also provided. In addition, a revised version of the basis function generation algorithm is provided to show the results of the methodology for small-sized problems. These results showed that the method converged faster to the optimal solution than value iteration. Finally, the basis function generation algorithm was used to generate the approximate cost-to-go structure for the centralized multi-agent mission problem that was used to manage long-duration operations in simulation. A large-scale mission management problem formulation with 40 vehicles, 5 task locations and 3 task types was simulated. The results show that the cost function developed by the basis function generation algorithm provides a noticeable improvement in managing resources by making adjustments in the vehicle allocation strategy based on system health feedback information.

Next, to fully investigate questions related to the implementation of such algorithms in real-time, Chapter 5 presents an indoor multi-vehicle testbed that was created to study long-duration mission and to develop health management systems for autonomous multi-agent mission platforms. This thesis presents the architecture and setup of the RAVEN (Real-time indoor Autonomous Vehicle test ENvironment) designed to study long-duration missions in a controlled environment. The RAVEN is designed to allow researchers to *rapidly prototype* UAV technologies by means of flight testing new vehicle configurations and algorithms without redesigning vehicle hardware. The RAVEN's mission system autonomously manages the navigation, control, and tasking of realistic air vehicles during multi-vehicle operations, researchers can focus on high-level tasks. Results are provided showing Draganflyer quadrotor vehicles and an foam airplane performing experiments in the test setup. Using the RAVEN, many other researchers have been able to perform a variety of first-time tests for UAV and other unmanned system technologies implemented in real-time.

Finally, Chapter 6 presents the development and implementation of techniques used to manage autonomous unmanned aerial vehicles (UAVs) performing long-term persistent surveillance operations. Although few papers have suggested the means by

which such a test could be performed, this thesis examines, develops and implements the technologies necessary to achieve this capability. This test represents a large step in the development and implementation of autonomous UAV mission technologies for long-duration missions. This section presents a non-invasive battery health monitoring technique that allows an autonomous system to estimate an electric-powered air vehicle's battery health state in real-time. In addition, a ground maintenance system for recharging the electric-powered quadrotor vehicles is presented and results showing a 24 hr fully-autonomous air vehicle flight-recharge test with no operator interaction was shown. This test marked the first time in the literature that a test such as this had been presented for an air vehicle. In addition, we discuss the additional hardware infrastructure needed to execute an autonomous persistent surveillance operation. Finally, we present results from a fully-autonomous, extended mission test using the technology and monitors developed in this thesis to improve system performance in real-time.

7.1 Future Work

Based on the work presented in this thesis, there are a variety of topics that can be explored for future work. First, based on the system architecture presented in Chapter 2, more research is needed in deciding the best methods to present different types of low-level health information to allow the mission system to interact with human operators during the decision making process. This thesis has explored and presented methods that allow an autonomous mission system to account for vehicle fuel constraints and general failure information from low-levels in the mission system architecture. However, more research is needed to understand how this information can be presented to an operator who is part of the decision making process during an operation. In [82, 100] we presented a natural language interface that allowed a human operator to communicate directly with a vehicle system. More research is needed on how to expand this type of interface to the mission system level, such that an operator can interact with any mission system component while allowing the mission system (and its subcomponents) to carry on with the current mission operations. Currently, research studying the relationship between autonomous systems and operator interfaces is in progress. In addition, we have used open-source voice software to communicate with a vehicle in the RAVEN at a very basic level, however more research is needed to make the human interface components to autonomous mission systems more user-friendly and reliable using the RAVEN and similar test

environments.

In addition, more work is needed to provide error bounds for the basis function generation algorithm in Chapter 3. Currently, research is being done to add sample complexity results to the current error bound provided in this thesis. In addition to the current error bounds, an error bound for multiple iterations of the algorithm is needed to expand on the current results. Likewise, a multi-policy version of this error bound for both a single and multiple iterations is also needed, and more work is needed in examining cases where the initial basis function set are defined by the user. In addition, more research is needed into the best methods for choosing the state relevance weights for a given problem. This research is another fundamental question related to the application of ALP. Currently, the multi-threaded version of this algorithm is currently being developed (to allow multi-processor resources to take advantage of the algorithm's structure). However, another interesting application of the algorithm is in distributed systems. For example, since the multipliers generated by one resource can be sent to other resources in a system, distributed vehicle and mission systems can use the multipliers generated by other mission systems in their decision making processes. More research can be done to better understand how to use this capability in a real-time system framework.

Third, a variety of mission-level swarm-related techniques and ideas have been provided in the literature. Using this research as a basis, there are many open questions related to task- and mission-level health management architectures and how to apply them to general problems that can be addressed in the RAVEN architecture. Specific questions such as how to detect and compensate for sensor failures (e.g, camera failures based on vision stream or packet dropouts due to an obstruction) without compromising mission level goals and how to demonstrate distribute mission processing tasks between resources. In addition, new and innovative health monitors can be developed and used in the RAVEN for monitoring motor and other failure types in real-time. More research must be done in the area of non-invasive health monitoring of electric air vehicles. Since UAVs continue to get smaller, simple yet non-invasive monitoring techniques can help autonomous systems evaluate a vehicle's future performance without modifying hardware designs. For example, monitors can be used to detect changes in actuator performance by observing the flight capabilities of a vehicle in set flight conditions, thereby providing a vehicle's hardware monitors with more information in detecting future failures.

Finally, more work is needed in developing an active landing and recharge systems that enable the fast recharge of battery technology. Currently, battery balancers

exist that allow a charger to interface with a battery to syphon off extra voltage on individual battery cells, insuring a safer method for charging batteries above 1C. Since most R/C batteries designs use plastic connectors to fasten batteries to the vehicles and chargers, a revised charging station must offer a "good" battery-charger connection that provides a reliable connection with very little resistance. In general, it is difficult to land a hovering air vehicle precisely. Therefore, active maintenance systems are necessary to provide method that either moves the vehicle to a desired location or moves the maintenance system to the vehicle, thus ensuring the proper connections are adequate for fast charging purposes. In addition, more work is needed in performing longer missions to truly be able to investigate and discover the limits of persistent surveillance missions using vehicle hardware. The RAVEN offers an excellent environment for such testing and more testing should be done to better define relationships between battery charging practices, flight activities and battery life over extended (i.e., 100+ consecutive flights over a week) periods of time.

Appendix A

Basis Function Generation Algorithm Formulation

Given an MDP (S, A, P, g) with multi-policies of the form $u : S \mapsto A$, discount factor α, state relevance weights c, and basis functions Φ, we can find an approximation of the optimal cost $J^* = \sum_{j=0}^{\infty}(\alpha P_{u^*})^j g_{u^*}$ under the optimal policy u^* of the form Φr using:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g_a(x) + \alpha \sum_{y \in S} P_a(x,y)\Phi(y)r \geq \Phi(x)r \quad \forall (x,a) \in D \end{array} \tag{A.1}$$

where $D \subseteq S \times A$. Here, let $\Phi\tilde{r}$ represent the approximate cost found using Eq. (A.1). To reduce the number of calculations needed to make a "good" approximation of J^* with guarantees on the error, one can sample both the constraints and the future states considered in the ALP. In doing so, the revised problem statement from Eq. (A.1) becomes:

$$\begin{array}{ll} \max_r & c^T \Phi r \\ \text{subj. to} & g_a(x) + \frac{\alpha}{M} \sum_{i=1}^{M} \Phi(y_i)r \geq \Phi(x)r \quad \forall (x,a) \in C \end{array} \tag{A.2}$$

where $C \subset D$ defines the set of constraints being used and for each state $x \in S$, a set of M future state $\{y_1, y_2, ..., y_M\}$ are used to approximate the quantity

$$\sum_{y \in S} P_a(x,y)\Phi(y) \quad \approx \quad \frac{1}{M} \sum_{i=1}^{M} \Phi(y_i) \tag{A.3}$$

Let $\Phi\hat{r}$ represent the approximate cost found using Eq. (A.2). This solution gives us a computationally efficient approximation of the cost J using $\Phi\hat{r}$. Next, after finding r_k, given Φ_k and a policy $u : S \mapsto A$, we can calculate each additional basis function at each state $x \in S$ using the Bellman update via

$$\phi_{k+1}(x) \;=\; g_a(x) + \alpha \sum_{y \in S} P_a(x,y)\Phi_k(y)r_k$$

where $a = u(x)$. However, we can approximate ϕ_1 using:

$$\phi_1(x) \;\approx\; \frac{1}{M}\sum_{k=1}^{M}\left(g_{a_0}(x_0^k) + \alpha\Phi_0(x_1^k)r_0\right)$$

by sampling future states $\{x_1^k\}$ for $k \in \{1,...,M\}$. Likewise, using trajectories using M trajectories of the form $\{x_0^k, x_1^k, x_2^k, ..., x_{N+1}^k\}$ for $k \in \{1,...,M\}$, we can also generate an approximate the future cost-to-go. Let $r_{00} = r_0 \in \mathbb{R}^{1\times K}$, $r_N = [r_{N0}\ r_{N1}\ ...\ r_{NN}]^T$ such that $r_{N0} \in \mathbb{R}^{1\times K}$ and $r_{Ni} \in \mathbb{R}$ for $i \in \{1,...,N\}$. Notice that for each trajectory, we can calculate each basic function in the following way:

$$\begin{aligned}
\phi_1(x_0) &= g_{a_0}(x_0) + \alpha\Phi_0(x_1)r_{00} \\
\phi_2(x_0) &= g_{a_0}(x_0) + \alpha\left[\Phi_0(x_1)r_{10} + \phi_1(x_1)r_{11}\right] \\
&= g_{a_0}(x_0) + \alpha\Phi_0(x_1)r_{10} + \alpha\left(g_{a_1}(x_1) + \alpha\Phi_0(x_2)r_{00}\right)r_{11} \\
&= g_{a_0}(x_0) + \alpha\Phi_0(x_1)r_{10} + \alpha g_{a_1}(x_1)r_{11} + \alpha^2\Phi_0(x_2)r_{00}r_{11} \\
\phi_3(x_0) &= g_{a_0}(x_0) + \alpha\left[\Phi_0(x_1)r_{20} + \phi_1(x_1)r_{21} + \phi_2(x_1)r_{22}\right] \\
&= g_{a_0}(x_0) + \alpha\Phi_0(x_1)r_{20} + \alpha g_{a_1}(x_1)r_{21} + \alpha^2\Phi_0(x_2)r_{00}r_{21} \\
&\qquad + \alpha g_{a_1}(x_1)r_{22} + \alpha^2\Phi_0(x_2)r_{10}r_{22} + \alpha^2 g_{a_2}(x_2)r_{11}r_{22} + \alpha^3\Phi_0(x_3)r_{00}r_{11}r_{22}
\end{aligned}$$

so on. Notice that for the $(N+1)$th basis function, the terms are organized as follows:

$$\begin{array}{rcccc}
\phi_{N+1}(x_0) &=& g_{a_0}(x_0) &+& \alpha\Phi_0(x_1)r_{N0} \\
&+& \alpha g_{a_1}(x_1)\sum_{i=1}^{N} r_{Ni} &+& \alpha^2\Phi_0(x_2)\sum_{i=1}^{N} r_{(i-1)0}r_{Ni} \\
&+& \alpha^2 g_{a_2}(x_2)\sum_{i=1}^{N-1}\sum_{j=1}^{i} r_{ij}r_{N(i+1)} &+& \alpha^3\Phi_0(x_3)\sum_{i=1}^{N-1}\sum_{j=1}^{i} r_{(j-1)0}r_{ij}r_{N(i+1)} \\
&+& \vdots &+& \vdots \\
&+& \alpha^N g_{a_N}(x_N)r_{11}r_{22}\cdots r_{NN} &+& \alpha^{N+1}\Phi_0(x_{N+1})r_{00}r_{11}r_{22}\cdots r_{NN}
\end{array}$$

Here, the mth row can be written as:

$$+ \quad \alpha^m g_{a_m}(x_m) \sum_{i_1=1}^{N-m+1} \cdots \sum_{i_m=1}^{i_{m-1}} r_{i_{m-1}i_m} r_{(i_{m-2}+1)(i_{m-1}+1)} \cdots r_{(i_1+m-2)(i_2+m-2)} r_{N(i_1+m-1)}$$

$$+ \quad \alpha^{m+1} \Phi_0(x_{m+1}) \sum_{i_1=1}^{N-m+1} \cdots \sum_{i_m=1}^{i_{m-1}} r_{(i_m-1)0} r_{i_{m-1}i_m} r_{(i_{m-2}+1)(i_{m-1}+1)} \cdots r_{(i_1+m-2)(i_2+m-2)} r_{N(i_1+m-1)}$$

Here, we have a pattern. Essentially, if we can compute the multipliers, they can be used for the computation for any trajectory. However, notice the following:

$$\begin{array}{rcccc}
\phi_1(x_0) & = & g_{a_0}(x_0) & + & \alpha\Phi_0(x_1) r_{00} \\
\phi_2(x_0) & = & g_{a_0}(x_0) & + & \alpha\Phi_0(x_1) r_{10} \\
 & + & \alpha g_{a_1}(x_1) r_{11} & + & \alpha^2\Phi_0(x_2) r_{00} r_{11} \\
\phi_3(x_0) & = & g_{a_0}(x_0) & + & \alpha\Phi_0(x_1) r_{20} \\
 & + & \alpha g_{a_1}(x_1)[r_{21} + r_{22}] & + & \alpha^2\Phi_0(x_2)[r_{00} r_{21} + r_{10} r_{22}] \\
 & + & \alpha^2 g_{a_2}(x_2)[r_{11} r_{22}] & + & \alpha^3\Phi_0(x_3)[r_{00} r_{11} r_{22}] \\
\phi_4(x_0) & = & g_{a_0}(x_0) & + & \alpha\Phi_0(x_1) r_{30} \\
 & + & \alpha g_{a_1}(x_1)[r_{31} + r_{32} + r_{33}] & + & \alpha^2\Phi_0(x_2)[r_{00} r_{31} + r_{10} r_{32} + r_{20} r_{33}] \\
 & + & \alpha^2 g_{a_2}(x_2)[r_{11} r_{32} + (r_{21} + r_{22}) r_{33}] & + & \alpha^3\Phi_0(x_3)[r_{00} r_{11} r_{32} + (r_{00} r_{21} + r_{10} r_{22}) r_{33}] \\
 & + & \alpha^3 g_{a_3}(x_3)[r_{11} r_{22} r_{33}] & + & \alpha^4\Phi_0(x_4)[r_{00} r_{11} r_{22} r_{33}]
\end{array}$$

and so on. By looking at the above, one will notice that there is a pattern with the multipliers from one iteration to the next. Therefore, if we save these multipliers they can be used to calculate the subsequent stages of multipliers and speed up the calculation of the basis functions for on-line computations. Again, notice the following:

$$\begin{aligned}
\phi_1(x_0) &= g_{a_0}(x_0) + \alpha\Phi_0(x_1) r_{00} \\
&= \begin{bmatrix} g_{a_0}(x_0) & \Phi_0(x_1) \end{bmatrix} \begin{bmatrix} 1 \\ \alpha r_{00} \end{bmatrix} \\
&= [g_{a_0}(x_0) \quad \Phi_0(x_1)] m_{00}
\end{aligned}$$

where $m_{00} = [1 \;\; \alpha r_{00}]^T$. Likewise, we can write:

$$\begin{aligned}
\phi_2(x_0) &= g_{a_0}(x_0) + \alpha\Phi_0(x_1)r_{10} + \alpha g_{a_1}(x_1)r_{11} + \alpha^2\Phi_0(x_2)r_{00}r_{11} \\
&= [g_{a_0}(x_0) \;\; \Phi_0(x_1)] \begin{bmatrix} 1 \\ \alpha r_{10} \end{bmatrix} + [g_{a_1}(x_1) \;\; \Phi_0(x_2)] \begin{bmatrix} 1 \\ \alpha r_{00} \end{bmatrix} \alpha r_{11} \\
&= [g_{a_0}(x_0) \;\; \Phi_0(x_1)]m_{10} + [g_{a_1}(x_1) \;\; \Phi_0(x_2)]m_{11}
\end{aligned}$$

where $m_{10} = [1 \;\; \alpha r_{10}]^T$ and $m_{11} = \alpha m_{00} r_{11}$. Then:

$$\begin{aligned}
\phi_3(x_0) &= g_{a_0}(x_0) + \alpha\Phi_0(x_1)r_{20} + \alpha g_{a_1}(x_1)[r_{21} + r_{22}] + \alpha^2\Phi_0(x_2)[r_{00}r_{21} + r_{10}r_{22}] \\
&\quad + \alpha^2 g_{a_2}(x_2)[r_{11}r_{22}] + \alpha^3\Phi_0(x_3)[r_{00}r_{11}r_{22}] \\
&= [g_{a_0}(x_0) \;\; \Phi_0(x_1)] \begin{bmatrix} 1 \\ \alpha r_{20} \end{bmatrix} + [g_{a_1}(x_1) \;\; \Phi_0(x_2)] \left(\alpha \begin{bmatrix} 1 \\ \alpha r_{00} \end{bmatrix} r_{21} + \alpha \begin{bmatrix} 1 \\ \alpha r_{10} \end{bmatrix} r_{22} \right) \\
&\quad + [g_{a_2}(x_2) \;\; \Phi_0(x_3)]\alpha \begin{bmatrix} \alpha r_{11} \\ \alpha^2 r_{00} r_{11} \end{bmatrix} r_{22} \\
&= \sum_{i=0}^{2} [g_{a_i}(x_i) \;\; \Phi_0(x_{i+1})]m_{2i}
\end{aligned}$$

where $m_{20} = [1 \;\; \alpha r_{20}]^T$, $m_{21} = \alpha \sum_{i=0}^{1} m_{i0} r_{2(i+1)}$ and $m_{22} = \alpha m_{11} r_{22}$.

Therefore, the $(N+1)$th basis function is generated using M trajectories of the form $\{x_0^k, x_1^k, x_2^k, ..., x_{N+1}^k\}$ for $k \in \{1, ..., M\}$ such that:

$$\phi_{(N+1)}(x_0) \;\; \approx \;\; \frac{1}{M} \sum_{k=0}^{M} \sum_{i=0}^{N} \left[g_{a_i^k}(x_i^k) \;\; \Phi_0(x_{i+1}^k) \right] m_{Ni}$$

where:

$$m_{Ni} \;\; = \;\; \begin{cases} \alpha \sum_{j=(i-1)}^{N-1} m_{j(i-1)} r_{N(j+1)} & \text{when } \; i > 0 \\ [1 \;\; \alpha r_{N0}]^T & \text{when } \; i = 0 \end{cases}$$

and $r_{00} = r_0 \in \mathbb{R}^{1\times K}$, $r_N = [r_{N0} \;\; r_{N1} \;\; ... \;\; r_{NN}]^T$ such that $r_{N0} \in \mathbb{R}^{1\times K}$ and $r_{Ni} \in \mathbb{R}$ for $i \in \{1, ..., N\}$.

This means that to generate the $(N+1)$th basis function, we must save $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$ values of m_{ij}. Since $|m_{ij}| = |r_{00}| + 1$, we must store $\frac{N(N+1)}{2}(|r_{00}| + 1)$ values to generate each $\phi_1(x_0)$,...,$\phi_{N+1}(x_0)$ given $g_{a_0}(x_0)$,...,$g_{a_N}(x_N)$ and $\phi_0(x_1)$,...,$\phi_0(x_{N+1})$.

Bibliography

[1] Aerospace Controls Laboratory at MIT. UAV SWARM Health Management Project Website. Available at http://vertol.mit.edu, July 2006.

[2] O. Amedi, T. Kanade, and K. Fujita. A visual odometer for autonomous helicopter flight. *Robotics and Autonomous Systems*, 28:185–193, 1999.

[3] 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.

[4] 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.

[5] K. C. Becker, C. S. Byington, N. A. Forbes, and G. W. Nickerson. Predicting and Preventing Machine Failures. *The Industrial Physicist*, 4(4):20–23, December 1998.

[6] J. Bellingham, M. Tillerson, M. Alighanbari, and J. How. Cooperative Path Planning for Multiple UAVs in Dynamic and Uncertain Environments. In *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, December 2002.

[7] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2000.

[8] D. P. Bertsekas and D. A. Castanon. Rollout Algorithms for Stochastic Scheduling. *Journal of Heuristics*, 5:89–108, 1999.

[9] D. P. Bertsekas and J. N. Tsitiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[10] D. Bertsimas and S. S. Patterson. The Air Traffic Flow Management Problem with Enroute Capacities. *Operations Research*, 46(3):406–422, May-June 1998.

[11] B. Bethke. Persistent Vision-Based Search and Tracking Using Multiple UAVs. Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, May 2007.

[12] S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004.

[13] F. Carr, A. Evans, J-P. Clarke, and E. Feron. Modelling and Control of Airport Queuing Dynamics under Severe Constraints. In *Proceedings of 2002 American Control Conference*, volume 2, pages 1314–1319, Providence, RI, May 2002.

[14] D. W. Casbeer, R. W. Beard, T. W. McLain, S-M. Li, and R. K. Mehra. Forest Fire Monitoring With Multiple Small UAVs. In *Proceedings of 2005 American Control Conference*, Portland, OR, June 2005.

[15] D. W. Casbeer, D. B. Kingston, R. L. Beard, T. W. McLain, S-M. Li, and R. K. Mehra. Cooperative Forest Fire Surveillance Using a Team of Small Unmanned Air Vehicles. Submitted to the International Journal of Systems Science, January 2005.

[16] P. R. Chandler, M. Pachter, D. Swaroop, J. M. Fowler, J. K. Howlett, S. Rasmussen, C. Schumacher, and K. Nygard. Complexity in UAV cooperative control. In *Proceedings of the American Control Conference*, Anchorage, AK, May 2002.

[17] C.-C. Chang and C.-J. Lin. *LIBSVM: A Library for Support Vector Machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[18] L. Clarke, E. Johnson, G. Nemhauser, and Z. Zhu. The Aircraft Rotation Problem. *Annuls of Operations Research*, 69:33–46, 1997.

[19] D. Cruz, J. McClintock, B. Perteet, O. Orqueda, Y. Cao, and R. Fierro. Decentralized Cooperative Control: A multivehicle platform for research in networked embedded systems. *IEEE Control Systems Magazine*, 27(2), April 2007.

[20] K. Culligan, M. Valenti, Y. Kuwata, and J. P. How. Three-dimensional flight experiments using on-line mixed-integer linear programming trajectory optimization. In *Accepted to the American Control Conference*, New York, NY, June 2007.

[21] D. P. de Farias. *The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application*. PhD thesis, Stanford University, June 2002.

[22] D. P. de Farias and B. Van Roy. The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research*, 51(6):850–856, 2003.

[23] D. P. de Farias and B. Van Roy. On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming. *Mathematics of Operations Research*, 29:462–478, 2004.

[24] Draganfly Innovations Inc. Draganfly V Ti Pro Website. Available at http://www.rctoys.com/draganflyer5tipro.php, January 2006.

[25] C. Edwards and I. Postlethwaite. Anti-windup and bumpless transfer schemes. In *Proceedings of the UKACC International Conference on Control 1996 (Conf. Publ. No. 427)*, pages 394–399, Exeter, England, September 1996.

[26] J. Evans, G. Inalhan, J. S. Jang, R. Teo, and C. Tomlin. DragonFly: A Versatile UAV Platform for the Advancement of Aircraft Navigation and Control. In *In the Proceedings of the 20th IEEE Digital Avionics Systems Conference*, October 2001.

[27] R. Franz, M. Milam, and J. Hauser. Applied receding horizon control of the Caltech ducted fan. In *Proceedings of the 2002 American Control Conference*, pages 3735–3740, Anchorage, MA, May 2002.

[28] M. Freed, R. Harris, and M. G. Shafto. Human vs. Autonomous Control of UAV Surveillance. In *Proceedings of the AIAA 1st Intelligent Systems Technical Conference*, Chicago, IL, September 2004.

[29] E. Frew, S. Spry, T. McGee, X. Xiao, J. K. Hedrick, and R. Sengupta. Flight Demonstrations of Self-Directed Collaborative Navigation of Small Unmanned Aircraft. In *Proceedings of the AIAA 3rd Unmanned Unlimited Technical Conference, Workshop, and Exhibit*, Chicago, IL, September 2004.

[30] M. Fudge, T. Stagliano, and S. Tsiao. Non-Traditional Flight Safety Systems and Integrated Vehicle Health Management Systems. Produced for the Federal Aviation Administration, ITT Industries, Advanced Engineering and Sciences Division, 2560 Alexandria Drive, Alexandria, VA 22303, August 2003.

[31] 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.

[32] V. Gavrilets, E. Frazzoli, B. Mettler, M. Piedmonte, and E. Feron. Aggressive Maneuvering of Small Helicopters: A Human Centered Approach. *International Journal of Robotics Research*, 20:705–807, October 2001.

[33] V. Gavrilets, I. Martinos, B. Mettler, and E. Feron. Flight Test and Simulation Results for an Autonomous Acrobatic Helicopter. In *Proceedings of the AIAA/IEEE Digital Avionics Systems Conference*, Irvine, CA, October 2002.

[34] R. Glaubius and W. D. Smart. Manifold Representations for Value-Function Approximation. In Daniela Pucci de Farias, Shie Mannor, Doina Precup and Georgios Theocharous (editors), editor, *Learning and Planning in Markov Processes – Advances and Challenges: Papers from the 2004 AAAI Workshop*, pages 13–18, 2004.

[35] G. Goebel. In the Public Domain: Unmanned Aerial Vehicles. Available at http://www.vectorsite.net/twuav.html, January 2005.

[36] R. Gopalan and K. Talluri. The Aircraft Maintenance Routing Problem. *Operations Research*, 46(2):260–271, March-April 1998.

[37] Gumstix, Inc. Gumstix.com Website. Available at http://www.gumstix.com, December 2006.

[38] S. R. Gunn. Support Vector Machines for Classification and Regression. Technical report, University of Southhampton, May 1998.

[39] D. Gurdan, J. C. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus. Energy-Efficient Autonomous Four-Rotor flying Robot Controlled at 1 Khz. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA '07)*, Rome, Italy, April 2007.

[40] Y. Hada and S. Yuta. A First-Stage Experiment of Long Term Activity of Autonomous Mobile Robot - Result of Respective Base-Docking Over a Week. *Lecture Notes in Control and Information Sciences: Experimental Robotics VII*, 271:229–238, 2001.

[41] T. Hansen and C.-J. Wang. Support vector based battery state of charge estimator. *Journal of Power Sources*, 141:351–358, 2005.

[42] J. C. Hartman and J. Ban. The series-parallel replacement problem. *Robotics and Computer Integrated Manufacturing*, 18:215–221, 2002.

[43] J. Higle and A. Johnson. Flight Schedule Planning with Maintenance Considerations. Submitted to OMEGA, The International Journal of Management Science, 2005.

[44] Hobbico, Inc. DuraTrax Mini Quake Website. Available at http://www.duratrax.com/cars/dtxd11.html, December 2006.

[45] A. Hobbs and S. R. Herwitz. Human Factors in the Maintenance of Unmanned Aircraft. Technical report, Office of the Chief Scientist for Human Factors, Unmanned Aerial Vehicles Human Factors: Program Review FY05, 2005. Available at http://www.hf.faa.gov/docs/508/docs/UAV05.pdf.

[46] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. Tomlin. The Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC). In *In the Proceedings of the 23rd Digital Avionics Systems Conference*, Salt Lake City, UT, November 2004.

[47] O. Holland, J. Woods, R. De Nardi, and A. Clark. Beyond Swarm Intellegence: The UltraSwarm. In *Proceedings of the 2005 IEEE Swarm Intellegence Symposium*, Pasadena, CA, June 2005.

[48] J. P. How. Lecture Notes: Aircraft Stability and Control (16.333): Lectures 3 and 4. Available at http://ocw.mit.edu/OcwWeb/Aeronautics-and-Astronautics/16-333Fall-2004/LectureNotes/index.htm, September 2004.

[49] J. P. How, E. King, and Y. Kuwata. Flight Demonstrations of Cooperative Control for UAV Teams. In *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, Chicago, IL, September 2004.

[50] Z. Jin, S. Waydo, E. B. Wildanger, M. Lammers, H. Scholze, P. Foley, D. Held, and R. M. Murray. MVWT-11: The second generation Caltech multi-vehicle wireless testbed. In *Proceeding of the 2004 American Control Conference*, pages 5321–5326, Boston, MA, June 2004.

[51] C. Johnson. Inverting the Control Ratio: Human Control of Large Autonomous Teams. In *International Conference on Autonomous Agents and Multiagent Systems: Workshop on Humans and Multi-Agent Systems*, Melbourne, Australia, July 2003.

[52] E. N. Johnson and D. P. Schrage. System integration and operation of a research unmanned aerial vehicle. *AIAA Journal of Aerospace Computing, Information, and Communication*, 1:5–18, 2004.

[53] E. Kadioglu and N. Papanikolopoulos. A method for transporting a team of miniature robots. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2297–2302, Las Vegas, NV, October 2003.

[54] T. Kalmar-Nagy, R. D'Andrea, and P. Ganguly. Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle. *Robotics and Autonomous Systems*, 46:47–64, 2004.

[55] P. Keller, S. Mannor, and D. Precup. Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, June 2006.

[56] E. King, M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control experiments on a multi-vehicle testbed. In *Proceedings of the 2004 American Control Conference*, Boston, MA, June 2004.

[57] N. Knoebel, S. Osborne, D. Snyder, T. Mclain, R. Beard, and A. Eldredge. Preliminary modeling, control, and trajectory design for miniature autonomous tailsitters. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, August 2006.

[58] T. John Koo. Vanderbilt Embedded Computing Platform for Autonomous Vehicles (VECPAV). Available at http://www.vuse.vanderbilt.edu/~kootj/Projects/VECPAV/, July 2006.

[59] Y. Kuwata. *Trajectory Planning for Unmanned Vehicles using Robust Receding Horizon Control*. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, February 2007.

[60] B. Kveton and M. Hauskrecht. Learning Basis Functions in Hybrid Domains. In *Proceedings of the 2006 National Conference on Artificial Intelligence (AAAI)*, Boston, MA, July 2006.

[61] M. G. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[62] Z. Liu, M. S. Squillante, and J. L. Wolf. On Maximizing Service-Level-Agreement Profits. Working paper, IBM T.J. Watson Research Center, 2000.

[63] Robin Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, January 2003.

[64] M. Maggioni and S. Mahadevan. Fast Direct Policy Evaluation using Multiscale Analysis of Markov Diffusion Processes. In *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, June 2006.

[65] S. Mahadevan. Samuel Meets Amarel: Automating Value Function Approximation using Global State Space Analysis. In *Proceedings of the 2005 National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, PA, July 2005.

[66] S. Mahadevan and M. Maggioni. Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. Technical report, University of Massachusetts, July 2006.

[67] S. Mahadevan and M. Maggioni. Value Function Approximation using Diffusion Wavelets and Laplacian Eigenfunctions. In *Neural Information Processing Systems (2006*, MIT Press, 2006.

[68] J. S. McCarley and C. D. Wickens. Human Factors in UAV Flight. Produced for the Federal Aviation Administration, Human Factors Research and Engineering Division – 2004 Annual Report, Institute of Aviation, Aviation Human Factors Division, University of Illinois at Urbana-Champaign, 2560 Alexandria Drive, Alexandria, VA 22303, 2004.

[69] I. Menache, S. Mannor, and N. Shimkin. Basis Function Adaptation in Temporal Difference Reinforcement Learning. *Annals of Operations Research*, 134:215238, 2005.

[70] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard. Vector Field Path Following for Small Unmanned Air Vehicles. In *Proceedings of the 2006 American Control Conference*, Minneapolis, MN, June 2006.

[71] Department of Defense Budget for Fiscal Year 2003. *Program Acquisition Costs by Weapon System*. Department of Defense, February 2002.

[72] Office of Management and Budget. *Budget of the United States Government – Fiscal Year 2005*. The Executive Office of the President, 2005.

[73] R. Olfati-Saber, W. B. Dunbar, and R. M. Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *Proceedings of the American Control Conference*, Denver, CO, June 2003.

[74] H. Paruanak, S. Brueckner, and J. Odell. Swarming Coordination of Multiple UAV's for Collaborative Sensing. In *Proceedings of the 2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations Aerospace Conference*, San Diego, CA, September 2003.

[75] R. Patrascu. *Linear Approximations For Factored Markov Decision Processes*. PhD dissertation, University of Waterloo, Department of Computer Science, February 2004.

[76] R. Patrascu, P. Poupart, D. Schuurmans, C. Boutilier, and C. Guestrin. Greed Linear Value-Approximation for Factored Markov Decisions Processes. In *Proceedings of the National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada, Jul 28-Aug 1 2002.

[77] G. L. Plett. Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs. *Journal of Power Sources*, 134:277–292, 2004.

[78] A. Prieditis, M. Dalal, A. Arcilla, B. Groel, M. Van Der Bock, and R. Kong. SmartSwarms: Distributed UAVs that Think. In *2004 Department of Defense Command and Control Research and Technology Symposium*, San Diego, CA, 2004.

[79] P. Rong and M. Pedram. An Analytical Model for Predicting the Remaining Battery Capacity of Lithium-Ion Batteries. *IEEE Transactions on VLSI Systems*, 14(5):441–451, 2006.

[80] J. Rosenberger, E. Johnson, and G. Nemhauser. Rerouting Aircraft for Airline Recovery. *Transportation Science*, 37(4):408–421, November 2003.

[81] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[82] T. Schouwenaars, M. Valenti, E. Feron, J. How, and E. Roche. Linear Programming and Language Processing for Human-Unmanned Aerial-Vehicle Team Mission Planning and Execution . *AIAA Journal of Guidance, Control and Dynamics*, 29(2):303–313, March–April 2006.

[83] P. J. Schweitzer and A. Seidmann. Generalized Polynomial Approximations in Markovian Decision Processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.

[84] D. Serena. Adaptive hierarchical swarming in persistent surveillance applications. In D. A. Trevisani and A. F. Sisti, editors, *Enabling Technologies for Simulation Science IX. Edited by Trevisani, Dawn A.; Sisti, Alex F. Proceedings of the SPIE, Volume 5805, pp. 299-308 (2005).*, pages 299–308, May 2005.

[85] D. Shim, H. Chung, H. J. Kim, and S. Sastry. Autonomous Exploration in Unknown Urban Environments for Unmanned Aerial Vehicles. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, August 2005.

[86] T. Shima, S. J. Rasmussen, and A. G. Sparks. UAV Cooperative Multiple Task Assignments using Genetic Algorithms. In *Proceedings of the 2006 American Control Conference*, Portland, OR, June 2005.

[87] W. D. Smart. Explicit Manifold Representations for Value-Functions in Reinforcement Learning. In *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*, 2004.

[88] A. J. Smola and B. Schölkopf. A Tutorial on Support Vector Regression. Produced as part of the ESPRIT Working Group in Neural and Computational Learning II, NeuroCOLT2, October 1998.

[89] B. L. Stevens and F. L. Lewis. *Aircraft Control and Simulation, Second Edition*. J. W. Wiley and Sons, 2003.

[90] R. Teo, J. S. Jang, and C. J. Tomlin. Automated multiple UAV flight the Stanford DragonFly UAV program. In *43rd IEEE Conference on Decision and Control*, pages 4268–4273, Paradise Island, Bahamas, December 2004.

[91] The MathWorks. MATLAB, 2006. Available at http://www.mathworks.com/products/matlab/.

[92] Thunder Power Batteries. Thunder Power Batteries Website. Available at http://www.thunderpower-batteries.com/, September 2006.

[93] R. Tiron. War Urgency Drives Decisions on U.S. Deployment of Drones. *National Defense*, 86(577):32–33, December 2001.

[94] G. Tournier, M. Valenti, J. P. How, and E. Feron. Estimation and control of a quadrotor vehicle using monocular vision and moire patterns. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, August 2006.

[95] M. A. Trick and S. E. Zin. Spline Approximations to Value Functions. *Macroeconomic Dynamics*, 1:255–277, January 1997.

[96] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J. P. How, and J. Vian. The MIT Indoor Multi-Vehicle Flight Testbed. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA '07)*, Rome, Italy, April 2007. Video Submission.

[97] M. Valenti, B. Bethke, G. Fiore, J. How, and E. Feron. Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, August 2006.

[98] M. Valenti, B. Bethke, J. How, D. P. de Farias, and J. Vian. Embedding Health Management into Mission Tasking for UAV Teams. In *Accepted to the 2007 American Control Conference*, New York, NY, July 2007.

[99] M. Valenti, D. Dale, J. How, and J. Vian. Mission Health Management for 24/7 Persistent Surveillance Operations. In *Accepted to the AIAA Guidance,*

Navigation, and Control Conference and Exhibit, Myrtle Beach, SC, August 2007.

[100] M. Valenti, T. Schouwenaars, Y. Kuwata, E. Feron, J. How, and J. Paunicka. Implementation of a Manned Vehicle-UAV Mission System. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, August 2004.

[101] V. Vapnik. *Statistical Learning Methods*. J. W. Wiley and Sons, 1998.

[102] Vicon. Vicon MX Systems. Available at http://www.vicon.com/products/viconmx.html, July 2006.

[103] R. Vidal, O. Shakernia, H. J. Kim, H. Shim, and S. Sastry. Multi-Agent Probabilistic Pursuit-Evasion Games with Unmanned Ground and Aerial Vehicles. *IEEE Transactions on Robotics and Automation*, 18(5):662–669, 2002.

[104] V. Vladimerouy, A. Stubbs, J. Rubel, A. Fulford, J. Strick, and G. Dullerud. A Hovercraft Testbed for Decentralized and Cooperative Control. In *Proceedings of the 2004 American Control Conference*, Boston, MA, July 2004.

[105] J.M. Wohletz, D.A. Castanon, and M.L. Curry. Closed-Loop Control for Joint Air Operations. In *Proceedings from the 2001 American Control Conference*, Arlington, VA, June 2001.

[106] O. Ziv and N. Shimkin. Multigrid Algorithms for Temporal Difference Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning: Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, August 2005.

[107] O. Ziv and N. Shimkin. Multigrid Methods for Policy Evaluation and Reinforcement Learning. In *Proceedings of the 2005 IEEE International Symposium on Intelligent Control*, Limassol, Cyprus, June 2005.