Learning Control Applied to a Model Helicopter

by

Ben Julian Weintraub

S.B., Aeronautical and Astronautical Engineering
Massachusetts Institute of Technology, 1992

Submitted to the
Department of Aeronautics and Astronautics in
Partial Fulfillment of the Requirements for the Degree of

Master of Science
in Aeronautical and Astronautical Engineering
at the

Massachusetts Institute of Technology

February 1994

Signature of Author _____
Department of Aeronautics and Astronautics
21 January 1994

Certified by _____
Associate Professor Christopher G. Atkeson
Thesis Supervisor

Certified by _____
Associate Professor John Hansman
Department of Aeronautics and Astronautics Reader

Accepted by _____
Professor Harold Y. Wachman
Chairman, Department Graduate Committee

Learning Control Applied to a Model Helicopter

by

Ben Julian Weintraub

Submitted to the Department of Aeronautics and Astronautics on 21 January, 1994
in partial fulfillment of the requirements for the Degree of
Master of Science in Aeronautical and Astronautical Engineering

ABSTRACT

An iterative learning control algorithm was developed for a model size helicopter. The algorithm allowed the helicopter to improve its tracking of a trajectory that was flown repeatedly. The objective of this learning control algorithm was to iteratively determine the correct feedforward command which would drive the plant over a prespecified trajectory. As part of this project, a helicopter testbed was developed. The helicopter testbed was easy to maintain, could be operated indoors, and its dynamics presented a complex control problem. The helicopter provided an opportunity to obtain valuable experimental data on control algorithms. The development of the trajectory learning algorithm and the testbed are described.

Trajectory learning algorithms have traditionally been associated with research in robot arm control. The helicopter is an inherently different dynamic platform. The particular algorithm used was based on an inverse model of the plant. Having fewer controls than states, helicopter trajectories had to be specified in a manner that did not overdetermine the system. This made it necessary to restrict the learning operator to improve tracking performance of only a limited number of states. Other difficulties that were encountered and overcome involved sensor noise and initial state errors.

Experiments were conducted using planar trajectories. Simulations and results from actual flight experiments demonstrated a dramatic reduction in trajectory errors after only a few learning iterations. Position errors over the trajectory were reduced to near hover accuracy. The helicopter testbed also provided insight into problems with real life application of the trajectory learning algorithm. Experimental data promoted a quick understanding of issues not brought out in simulations.

Thesis Supervisor: Christopher G. Atkeson
Title: Associate Professor, Brain and Cognitive Sciences Department

Acknowledgments

An advisor in many ways, Chris Atkeson has been a great mentor. His style is unusual, to say the least. His wisdom and motivation have pushed me through some very frustrating times. Often, his presence alone instilled the confidence to go on. Chris provides plenty of opportunity to choose your own direction, play with toys, and work on your own concoctions. At the same time, he is always in the background with solid technical advice and a wealth of first-hand experience.

Though my father has been battling bone cancer through the writing of this thesis, both my parents have kept up a front which allowed me to finish my work. Mom and Dad have always been a special source of motivation for me, though they have *always* told me to take it easy, rather than pressuring me. It is their strength and selfless attitude which has driven me so far. I only hope to be granted the time to repay them.

My lab partner, Jackie Moore, and I have been through some tough times with this project. Her dedication is as strong as anyone's I know. Her help with the helicopter experiments was invaluable.

Stefan Schaal, a post-doc, is the next best thing to Chris for advice and talking over ideas. I miss him when he's away on trips to his homeland.

Paul Tradelius, a first class RC and full size pilot, has shared his knowledge, labor, and resources with us to an extent that is difficult to thank. Let's just say that every step we made with the helicopter mechanics - construction, troubleshooting, repairing, and modifying, was performed only after consulting Paul.

The helicopter project was a tremendous undertaking. There are many other people who contributed labor and advice vital to its success. Many thanks go to all these people. To name a few: Anne Wright for her vision system and her enthusiasm, proven by three trips to Georgia. Paul Viola, for his work and advice in the early days of the helicopter project. Jerry Pratt for his knowledge and energy for mass production. Chris Barnhardt for working on a thousand ideas at once.

This thesis describes work done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the research was provided under Air Force Office of Scientific Research grant AFOSR-89-0500, by the Siemens Corporation, and by the ATR Human Information Processing Research Laboratories.

Table of Contents

Chapter 1

Introduction

Intelligent control approaches are increasingly being looked towards as a method of achieving high performance control. In this thesis, the applicability of an iterative trajectory learning algorithm to a flight vehicle was investigated. The premise for this research was that a learning algorithm could improve performance by compensating for errors in the models used in the design of a fixed control system. Through the use of a model size helicopter, it was demonstrated that learning control algorithms could provide improvements on complex, dynamic plants by taking advantage of past performance.

1.1 Motivation For Trajectory Learning

Controllers which have the ability to recognize and adapt to modeling errors or changing plant conditions have already demonstrated their utility in several practical applications [Sofge and White, 1992]. Use of intelligent control, in many cases, can increase plant performance and/or decrease operating, manufacturing, and development costs for the plant and its control system. While the range of potential applications for these control methods is unlimited, experiments are typically conducted on plants or processes with relatively slow and *safe* dynamics. In this context, safe dynamics is meant to describe

plants which will not experience catastrophic failures when used with a poor controller. The primary motivation of this thesis was to explore the uses for learning control in plants with fast and complicated dynamics.

The approach taken in this thesis was to apply trajectory learning algorithms as a first step towards demonstrating the utility of intelligent control methods for high performance control over a flight vehicle. Trajectory learning is a method of improving the performance of a controlled system which repeatedly performs a prespecified and often difficult task or maneuver. Trajectory learning algorithms have been widely researched and successfully applied in the context of industrial robotics. These applications typically demonstrate dramatic performance improvements over non-adaptive control methodologies. Chapter 2 describes the development of the trajectory learning algorithm from previous research. The chapter highlights a simple, yet significant change which was required in order to apply the algorithm to the helicopter.

Trajectory learning control may provide aircraft with maneuverability superior to conventional controllers. The characteristic of this type of controller, however, is that maneuvers must be repeated several times in order for performance to improve. Knowledge gained by performing one trajectory is not recorded in a manner that allows transfer to another trajectory. During execution, trajectories must be performed in their entirety. In a particular situation, the vehicle must select a maneuver from a 'library' of trajectories previously performed. Thus, an important limitation of the trajectory learning algorithm is that desired maneuvers which vary slightly from those practiced can not be performed with improvements over the conventional controller. Traditionally, trajectory learning algorithms show significant improvements after only a few iterations. Therefore, maneuvers need only be practiced three to five times.

The performance that can be achieved by a trajectory learning controller is limited by the level to which disturbances affect the plant. The total trajectory errors depend, in part, on the aggressiveness of the maneuver, the feedback controller, and the environmental disturbances. These factors will induce both repeatable and random errors in the trajectory. As a feedforward controller, learning control algorithms can only compensate for repeatable disturbances. Trajectory learning algorithms are only beneficial for maneuvers where the dominant errors are repeatable. Disturbances, such as random wind and noise, are left to be handled by the feedback controller. These random disturbances induce errors in the feedforward command through the learning algorithm , but only as much as they affect the trajectory errors. Typically a learning rate is employed to reduce the effect of random errors on the learning algorithm's estimate of the feedforward command.

These limitations may not be as serious as they may first seem. Conventional controllers are often adequate for the great majority of a vehicle's operating time. These controllers can sufficiently attenuate random disturbances. There may be a number of high performance maneuvers, however, that are repeatedly performed by an aircraft serving a specific duty. Aggressive maneuvers often push the aircraft into regions where modeling errors are responsible for repeatable disturbances. Many pilots are trained to perform specific maneuvers in reaction to certain situations, such as occurs during 'dogfighting' or special flight procedures. It is reasonable to believe that a limited selection of trajectories could benefit the overall performance of a flight vehicle. Unmanned vehicles are especially likely to perform a set of prespecified maneuvers. The recent growth in unmanned aerial vehicle (UAV) use and production [UAV report, 1993], may open an area for practical application of trajectory learning algorithms.

1.2 Motivation for Helicopter Testbed

Experimental results on the application of learning algorithms to fast dynamic plants are limited. An important reason for the scarcity of experimental results is the lack of stability and robustness guarantees for the learning controllers. For vehicles such as aircraft and rotorcraft, inadequate performance of a controller could be catastrophic. The hazards of the aerospace field often lead to an extensive research and development process which requires great amounts of time and money, and is not readily accessible to a research institution. Extensive simulation and testing is necessary before conducting flight tests of a new control system. At the same time, however, investigation into the uses of learning control could benefit significantly from empirical results. The secondary motivation for this research was to conduct experiments on an actual flight vehicle. By necessity, this vehicle had to be inexpensive, while simultaneously presenting us with a challenging control problem. The helicopter platform developed for this project has proven itself to be a valuable research tool, with potential for applications far beyond the scope of this thesis.

It is hoped that experiments on a testbed such as the MIT helicopter can speed the research and development process. A small test platform can provide several advantages over simulations and full size testing in certain instances. The result may be flight control algorithms which become more developed before implementation on the full size flight vehicle. The helicopter testbed allows results to be gathered at a very early stage in the research process. In this research, flight experiments were conducted inside the laboratory using the same computational hardware and software that were used for simulations. Such flexibility allowed quick and safe assessment of the performance of a flight control algorithm.

A radio controlled model helicopter was chosen as the baseline for the flight test platform. Many considerations led to this choice of platform. Many RC model helicopters are extremely agile vehicles, capable of performing rolls and loops and flips within a two meter spherical boundary. The helicopter presented us with unstable, non-linear, and coupled dynamics. These features provided an interesting dynamic platform for control experimentation and evaluation. The vehicles were also low cost, easy to maintain, and easy to operate. These combined attributes provided for useful flight testing of high performance control algorithms on a daily basis.

Radio Controller helicopters are used by hobbyists around the world. The models are inexpensive (approximately $1000), readily available and are intended for use by people with only basic mechanical skills. The XCell 60 helicopter model chosen for our testbed has a rotor diameter of 1.42 meters, and a ready to fly mass of 5 kg. It can be flown in a variety of environments. The XCell is often used in model aerobatics competitions. Such a popular model has a large following, which includes aftermarket parts for additional reliability and performance, and a wealth of human knowledge.

The complete helicopter testbed included instrumentation, computation hardware, and a stabilizing hover controller. The helicopter was modified with an electric motor, which allowed indoor flights. Instrumentation consisted of attitude rate, attitude, and position measurement systems. The computation hardware processed sensor data and ran the control algorithm to command the helicopter. The hover controller, which was a linear quadratic gaussian design, stabilized the helicopter. Chapter 3 provides a complete description of the testbed as it was configured for these experiments.

1.3 Scope of this Thesis

The experiments performed for this thesis were limited to planar trajectories based on single input linear models. Simulations and flight tests on the helicopter testbed were used to verify the performance of the learning algorithm. These result are described in chapter 4. The planar trajectories were constructed so that the learning algorithm was only applied to the roll and pitch control axis of the helicopter. On each axis, computations for trajectory initialization and the learning operator were made according to a single linear model of the helicopter's dynamics near hover.

Trajectories were initially flown using the best feedforward command computed from the hover models. Planar trajectories were chosen such that initial performance included significant errors, yet the learning operator could improve performance using the hover models. The trajectory learning algorithm improved performance on all of the planar maneuvers attempted, using only the hover models. The validity of the models, and therefore performance of trajectory learning, deteriorate as the helicopter's flight regime expands further from hover. Trajectories such as rolls or loops would likely be beyond the ability of a learning controller that has access to only the hover model. Aggressive maneuvers such as these would require more extensive models or multiple models around the envelope of the trajectory.

This thesis and the trajectory learning experiments were conducted as a first step in applying intelligent control algorithms to the helicopter testbed. The trajectory learning algorithm provided an opportunity to observe aggressive helicopter maneuvers using a relatively simple control algorithm. As a feedforward controller, the algorithm has limited applicability. It is hoped that adaptive or learning feedback controllers, and more general feedforward controllers will be included in future research.

Chapter 2

The Trajectory Learning Algorithm

The goal of any trajectory learning algorithm is to find the correct feedforward command which will drive the plant along a prespecified trajectory. Trajectory learning algorithms are based on an iterative process whereby the controller learns how to track the fixed trajectory. Even with a linear learning operator, the iterative process can result in non-linear corrections. After each trajectory attempt, deviations from the trajectory are recorded and used by the algorithm to update the feedforward command. The new feedforward command is used on the next plant run, hopefully with improved trajectory tracking. As a feedforward controller, plant stability is not a major issue for a trajectory learning algorithm, since plant stabilization is left to a fixed feedback controller. The trajectory learning algorithm assumes that modeling errors are much larger than random sensor and actuator noise.

Every trajectory learning algorithm can be described in three basic steps. The first step is initialization, in which the feedforward command for the first plant run is computed off-line, usually according to a model. The second stage is the execution stage where the actual plant is run using the most recent feedforward command. In the third step, the tracking errors are mapped into feedforward command corrections. Steps two and three

are repeated until satisfactory performance is achieved. Various trajectory learning algorithms differ in the manner in which they perform the mapping. The approach applied here was based on using the inverse plant model [An, et. al., 1988].

In practice, the trajectory learning algorithm was applied to two separate axis of the helicopter independently. This chapter will discuss the specific algorithm used in this thesis. It will begin by highlighting the features of the algorithm and overviewing previous work. The following discussion will give an in depth description of the algorithm and its features.

2.1 Features of the Approach Used in this Thesis

The primary contribution of this thesis was in applying trajectory learning to a complex dynamic plant. Three significant complications were encountered due to the characteristics of the helicopter plant: less than complete actuation, initial state errors, and sensor noise. All of the features of the algorithm used in this thesis are more thoroughly described in later sections.

The first problem was encountered when deciding how to map tracking errors into command updates. For the single input models used for each control axis of the helicopter, only one state could be independently driven over a trajectory by the single command. Therefore, the trajectories could only be specified for one state in each of the axes. For the experiments presented here, the trajectories were specified in x and y positions for the pitch and roll models, respectively. In light of modeling errors, it is not possible to specify the corresponding trajectories of the other states for the feedback controller's reference (since the dependency is determined by the actual plant dynamics and is only approximated by the model). However, a full state feedback controller was

used with the helicopter. Unlike other trajectory learning applications, the controller was left to regulate all states to zero. It was not necesary to provide the feedback controller with desired values for any of the states. By considering the full closed loop model and plant description, with a single input and single output (SISO), it is sufficient to provide the learning operator with the single state trajectory. In this manner the actions of the feedback controller were taken into account, resolving problems created by only specifying a trajectory for one state of a plant. This issue does not appear to have been addressed in research with robot arm control, where typically each joint is actuated and the relationship between joint position and joint velocity is assumed to be known perfectly.

Another problem arose because the helicopter began each trajectory from a hover. This caused the initial position in the trajectory to vary randomly within the limits of the hover controller. These initial errors created large disturbances in the early portions of the feedforward command update. The initial errors were removed in off-line processing before presenting trajectory data to the learning algorithm. This was done by estimating the closed loop plant's behavior for that particular initial error. The closed loop model would predict the 'natural' decay of an initial error, which in turn indicated how it should be removed. Simulations indicated that this processing would be sufficient, but actual implementation required additional filtering of the initial error.

A third problem presented by the helicopter platform was the inherently noisy sensor data. Since measurements of x and y position were limited by the resolution of the position sensing system, (approximately 1 cm) the position data, and therefore, trajectory error, was somewhat jumpy. It was found that simply filtering the jumpy trajectory error before presenting it to the learning algorithm was sufficient for noise attenuation. The

filtered signal had to retain important plant dynamics, while eliminating any unwanted spikes which could have resulted in incorrect feedforward command updates.

2.2 Previous Work

The concept of iterative learning control was originally introduced by Uchiyama (1978). Since then, many researchers have studied various aspects of iterative (also called repetitive or trajectory) learning. Most of this research is concerned with robotic arm movement [Ahn, et al. 1992; An, et al. 1988; Aoyama, et al., 1989; Arimoto, et al., 1984, 1990; Guglielmo, et al., 1989, Moore, 1993].

The main focus of iterative control research is how to map errors from previous movements into improvements to the feedforward command. Typical learning operators make use of plant gradient (Jacobian) information, feedback controller actions, or an inverse plant model. Theoretical derivations of convergence and stability of the learning operator are often presented. Assumptions for these proofs are often linked to the characteristics of robot arms. Sensor noise, initial errors, and disturbances violate conditions for some of these proofs, though work is being conducted to eliminate such assumptions [Arimoto, 1990]. Additionally, some learning operators require low learning rates to consistently improve performance. Therefore, learning requires many trials, which is much more acceptable for robot manipulators than flight vehicles.

The approach taken in this thesis was based on the work of Atkeson (1988), where the feedforward command update came from direct application of the inverse model. Atkeson's work emphasized the performance of the learning operator in terms of speed of convergence and robustness to disturbances and sensor and actuator noise. This algorithm demonstrated dramatic improvements after only three learning trials in actual

experiments with a direct drive robot arm. Atkeson also presented a condition for convergence of the inverse model learning operator, but used a simulation to show that this would not necessarily indicate acceptable performance. One of his points was that an important test for any control algorithm is successful implementation on a real plant.

In Atkeson's experiments two assumptions were made (correctly in the case of a robot arm) concerning initial errors and the actions of the feedback controller. These were not true in the case of the helicopter. Specifically, this thesis directly addressed the closed loop dynamics and initial errors. Sensor noise was handled by Atkeson through filtering. This simple, but effective approach was maintained in this thesis.

2.3 Learning Algorithm

As stated earlier in this chapter, a trajectory learning algorithm consists of three basic steps: Initialization, Execution, and Update. Figure 2-1 shows these steps and the interaction with the plant and inverse model as they were applied in this thesis. In this diagram, the full state vector X should be distinguished from the position state x of the pitch axis system. The following discussions will use the position state x in examples, but the roll axis analysis would simply substitute the corresponding position state y.

In the initialization phase the desired trajectory, which is specified in terms of position $x_d(t)$ or $y_d(t)$ only, is run through the inverse *closed loop* plant *model* $\hat{P}_{CL}^{-1}$ to create the initial feedforward command history $u_{ff}^0(t)$. Care must be taken so that the desired trajectory is physically executable by the plant model, or else the learning process will not be able to eliminate the trajectory error. The initial feedforward command is also the estimate of the ideal feedforward command $\hat{u}_{ff}^*(t)$.

During trajectory execution, the feedforward command $u^i_{ff}(t)$ is fed into the closed loop plant as a function of time (from the start of the trajectory). The feedforward command is summed with the feedback command $u^i_{fb}(t)$. The actual position trajectory is separated from the full state output by the plant output matrix C, and then recorded into memory. The dashed line in the figure outlines the SISO description of the closed loop plant. The trajectory performed by the actual plant will stray from the desired trajectory due to modeling errors, disturbances, and sensor noise (in terms of how the sensors affect the feedback controller).

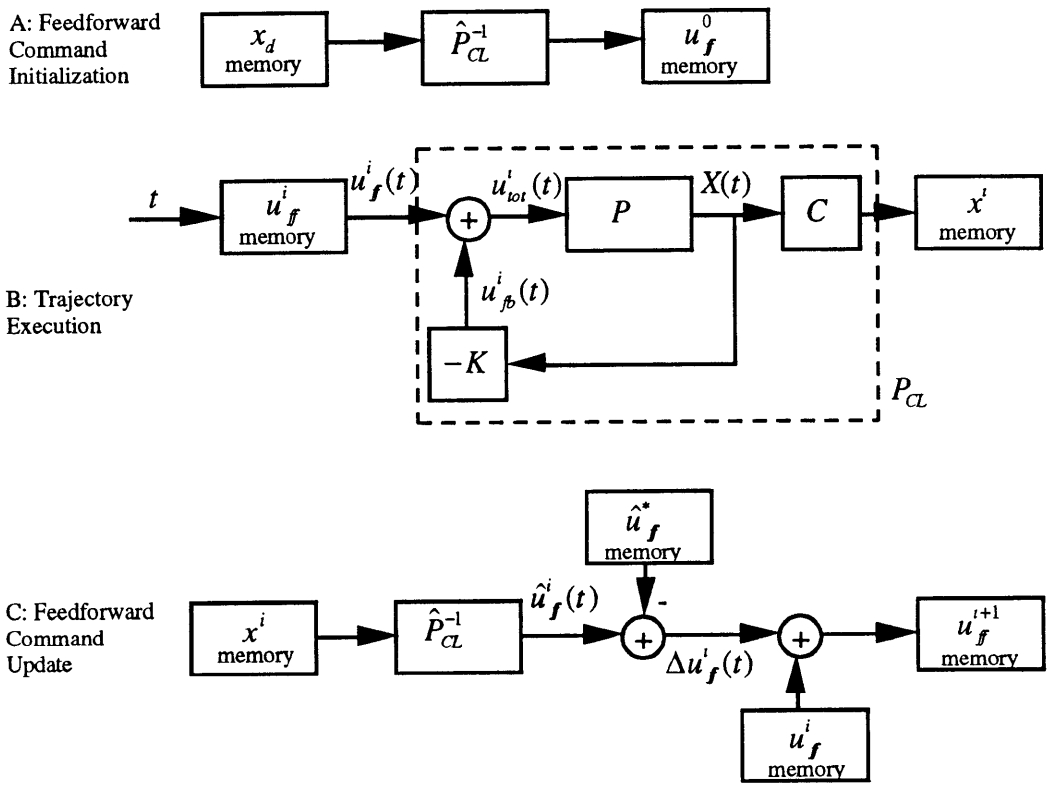

Figure 2-1. Steps of the Trajectory Learning Algorithm

It is the goal of the update phase to create a better estimate of the correct feedforward command for the actual plant dynamics. In the update the inverse closed loop plant model acts as the learning operator which maps position tracking errors into command corrections. A discussion of why the update operation (shown in figure 2-1c) is

appropriate is given below. Not shown in figure 2-1 are filtering and initial error canceling, both of which occur between steps b and c. These will be discussed later.

Steps b and c are performed repeatedly, until acceptable performance is obtained, or successive runs no longer provide improvement.

2.3.1 Initialization

The initial command $u_{ff}^{0}(t)$ is calculated to be the best estimate of the feedforward command which will drive the closed loop plant P_{CL} over the trajectory. An inverse of the closed loop plant model $\hat{P}_{CL}$ is used for this calculation. Since the plant model was developed around hover, errors existed in the initialization according to the model's accuracy around the trajectory. Without a more detailed model, or multiple local models, the hover model provided the best initial estimate for the feedforward command.

The inverse plant model was created by inverting the discrete time transform of the system. In order to get an accurate command from the inverse model, it was necessary to use a desired trajectory which is achievable by the model. The desired trajectory was computed by an optimization routine. The routine tried to fit an arbitrary goal trajectory with an actual plant model trajectory while penalizing control usage. The particular inversion and optimization methods used in this thesis are described below, but it should be clear that the manner in which this was done is independent of the learning algorithm and can therefore be performed in a variety of ways.

2.3.1.1 Creating a Plant Inverse

Inversion of the plant model was achieved by inverting the discrete time z-transform [Siebert, 1986] of $\hat{P}_{CL}$. The z-transform is obtained from a discrete time state space description of the plant. Equation (2-1) is a general SISO state space description of the type used for the model $\hat{P}$ of each of the helicopter axis. In this general description, the matrices A_n, B_n, C_n represent local linearization of the plant dynamics. Given a non-linear model, the procedure of inverting the plant dynamics can be carried out by using matrices which vary with the trajectory timestep according to the current state of the system X_n. For this thesis, however, only the hover models were used, and the state matrices remained constant for the plant inverse calculations. The subscripts indicating the time dependency of the state space equations will, therefore, be dropped in the following discussion.

$$\begin{aligned} X_{n+1} &= A_n X_n + B_n u_n \\ Y_n &= C_n X_n \end{aligned} \qquad (2\text{-}1)$$

In our experiments the trajectories were specified in terms of position, so the output y was selected as the position state for each axis. The z-transform for the plant given by (2-1) is given by equation (2-2) [Franklin, et al., 1990].

$$P(z) = \frac{Y(z)}{u(z)} = C(zI - A)^{-1} B \qquad (2\text{-}2)$$

Instead of taking the inverse of (2-2) we would like to find the transform of the closed loop model. The closed loop dynamics $\hat{P}_{CL}$ are described by equation (2-1b), where K represents the linear full state feedback gains used in the helicopter's feedback controller. Here the output state is specified as the position state x_n. Figure 2-1B shows the block

diagram equivalent for this plant description. The corresponding z-transform is given by equation (2-2b).

$$\begin{aligned} X_{n+1} &= AX_n + Bu_{ff}(n) - BKX_n \\ &= (A - BK)X_n + Bu_{ff}(n) \\ x_n &= CX_n \end{aligned} \qquad \text{(2-1b)}$$

$$\hat{P}_{CL}(z) = \frac{x(z)}{u_{ff}(z)} = C(zI - A + BK)^{-1}B \qquad \text{(2-2b)}$$

In the helicopter experiments, a state estimator was running, but its dynamics were assumed to be significantly faster than the plant dynamics and are not taken into account here.

The transform of the inverse plant $\hat{P}_{CL}^{-1}$ is then obtained by inverting equation (2-2b). This new transfer function describes an acausal system (intuitively, commands u_{ff} must precede outputs x). In order to maintain a proper transfer function, the inverted transfer function is multiplied by z^N, where N is the minimum number necessary to create a proper transfer function, given by (2-3). For the models used in this thesis, N=11, which was the number of discrete sample periods it took a command to affect the position state.

$$\hat{P}_{CL}^{-1}(z) = \frac{u_{ff}(z)}{x(z)} = z^N\left[C(zI - A + BK)^{-1}B\right]^{-1} \qquad \text{(2-3)}$$

Equation (2-3) is a proper inverse transform of the plant, with the exception that the output of (2-3) will be delayed by N timesteps. In order to complete the inversion properly, this delay must be accounted for by shifting the output of the inverse N steps back in time. Since all applications of the inverse plant model occur off-line, this is not a problem.

A more general method of taking the inverse of a plant would be through direct numerical integration. The method described above takes advantage of the linear discrete time properties of our model, and provides a general method of obtaining a useful inverse description (can be directly used in MATLAB™, for instance) for plants that can be described in this manner. The code which runs the inverse dynamics on an input is given in Appendix A.

2.3.1.2 Creating the Desired Trajectory: an Optimization Method

In order for the inverse plant model to accurately calculate the initial feedforward command, the desired trajectory must be physically achievable by the plant model (i.e. steps are an impossible trajectory for the helicopter model, which is similar to a triple integrator). Once again, the method used here took advantage of the linear discrete model of the plant to create a general method of finding feasible trajectories. The method can be phrased as the solution to an optimal control problem with the cost function (2-4).

$$E = \sum_{t=0}^{T} \left[\left(x_g(t) - x_d(t) \right)^2 + w^2 u_{tot}^0(t)^2 \right] \qquad (2\text{-}4)$$

Given any arbitrary goal trajectory $x_g(t)$, the optimization method will find a command history $u_{ff}^0(t)$ which will drive the plant over a feasible trajectory $x_d(t)$ which closely matches the arbitrary original. The feasible trajectory starts the plant from a zero initial state (i.e. in hover). The penalty on control usage w can be adjusted in a trade-off between reducing control usage and following $x_g(t)$ more closely. There are many ways to solve an optimal control problem phrased in this manner [Kirk, 1970], but the exact method used here is based on a regression. The regression takes into account the plant model dynamics, to ensure that the resulting trajectory $x_d(t)$ is physically achievable. The trajectory $x_d(t)$ becomes the desired trajectory for the learning algorithm. This

method allowed us to easily create arbitrary goal trajectories, and then turn them into reasonable ones for the helicopter. The details of this regression are given in Appendix B.

2.3.2 Trajectory Execution

During this phase of trajectory learning, the real plant performs the ith trajectory according to the current feedforward command $u_{ff}^{i}(t)$, stored in memory. This phase is repeated for every learning iteration. On each run, the actual position history $x^{i}(t)$ of the plant is recorded. The characteristics of trajectory learning allow all processing to occur off-line, simply storing and reading items from memory during execution. This can be advantageous in the respect that other processing demands are most likely at their highest during plant operation.

It is important to point out that the feedback controller is not tracking the error between the actual trajectory and the desired one, but instead is still regulating the plant to zero. The result is that as the trajectory drives the plant farther from the origin in the state space, the feedback commands grow tremendously large (easily exceeding the saturation level of the actuators). The command that reaches the plant is the sum of $u_{ff}^{i}(t)$ and $u_{fb}^{i}(t)$. Since the feedforward command $u_{ff}^{i}(t)$ is computed and updated according to the closed loop plant model, the behavior of the feedback controller is accounted for and control saturation does not occur. This is unlike Atkeson's (1988) algorithm in which the feedback command $u_{fb}^{i}(t)$ approaches zero as the plant converges to the desired trajectory. In his approach, the feedback controller operates off of the full state trajectory error $\left(X(t) - X_{d}(t)\right)$ during trajectory execution.

2.3.3 Initial Error Removal and Filtering

Before the position data could be used in the command update operation, it had to be processed to remove the initial error and noise. Considerations of noise are common in the research involving application to real robots [Atkeson, 1988; Guglielmo, 1989]. Typically, however, it is assumed that robots can return to a starting point (within the accuracy of its sensors) at the start of every iteration. This was not true of the helicopter. Helicopter trajectories begin from a hover condition. The feedback controller's ability to maintain this position was limited, and the consequence was that successive trajectory iterations were not initiated from identical positions. Random initial errors cause poor trajectory convergence during learning. The methods described in this section were developed from simulation experiments. Implementation on the actual helicopter, however, required additional work, which is described in Chapter 4 where the data is analyzed.

Some research on initial error handling has been conducted by Arimoto. Arimoto's paper (1990) presents a convergence theorem for an iterative learning scheme that is robust to small initial errors. The paper does not describe a special method of handling initial errors, but rather demonstrates the learning algorithms tolerance to the errors. The proof is based on an analysis of robot arm dynamics, and it restricts the size of initial errors to be small in comparison to the accuracy to which one would like to converge. The helicopter did not meet either of these conditions. The dynamics were significantly different in character, and the aim was for trajectory performance to converge to an accuracy equal to that of the initial error: i.e. achieving hover accuracy, in terms of root mean squared error (rms error), over the trajectory.

In this thesis, initial errors were dealt with by direct removal. It was assumed that the initial error was truly random, and it, therefore, contained no useful trajectory error information and should be ignored. In the deterministic case the response of any linear time invariant system is the sum of a zero state response (ZSR) and zero input response (ZIR) [Siebert, 1986] (equation 2-5). The ZIR is the response of the closed loop plant to its initial condition and no inputs, while the ZSR depends only on the system inputs.

$$Y(z) = P_{CL}(z)u_{ff}(z) + ZIR \qquad (2\text{-}5)$$

Since there was only concern for the ZSR, $P_{CL}(z)u_{ff}(z)$, the plant model was used to estimate the ZIR and subtract it from the total response $Y(z)$. In other words, the history of the initial error decay was estimated by running the initial state vector through the closed loop plant model (with no other input). Once the decay was estimated, it was subtracted from the actual trajectory. If the system was deterministic with no modeling error, this would leave only useful trajectory error information.

A similar methodology can be derived numerically in the time domain for non-linear systems. The idea is the same: Estimate the effect of the initial error on the entire trajectory and then subtract it from the observed trajectory. The non-linear case is more difficult because the property of superposition must be forfeit.

Filtering of the trajectory data was performed in order for the inverse model learning operator to produce a reasonable command update. Just as in creating a desired trajectory, the inverse model will produce an errant command if the position trajectory is not feasible. Noisy position data will result in false command estimates from the inverse model. Noise was removed by filtering the data with a 3rd order, 1.5 Hz, low pass Butterworth filter. The filter was first applied in the forward time direction, and then applied backwards in time. This is equivalent to 6th order filtering, but with zero time

lag. The 1.5 Hz cutoff frequency was determined experimentally such that errors within the dynamic range of the plant were retained, while noise was sufficiently attenuated .

2.3.4 Feedforward Command Update

In this phase, the plant's most recent movement is analyzed and the feedforward command is updated to improve performance on the next movement. The data must be *prepared* by removing the initial error and filtering, as described in the previous section. The learning operator again makes use of the inverse closed loop plant model $\hat{P}_{CL}^{-1}$. In this case, the inverse is used to estimate the error in the feedforward command $\Delta u_{ff}^{i}(t)$ used on the ith iteration. The feedforward command is then updated $u_{ff}^{i+1}(t)$ and stored for use on the next iteration. The key development of the trajectory learning algorithm applied in this thesis was the consistent consideration of the full closed loop plant and corresponding model.

By treating the whole closed loop plant, a SISO system is described with the feedforward command as the input and the position state as an output. This abstraction allows for representation of the dynamic modeling error, which creates the trajectory errors, as an input command disturbance. The learning operator is oblivious to the fact that a feedback controller and other states exist in the system. The command error $\Delta u_{ff}^{i}(t)$ is easily estimated from the SISO plant description.

The feedforward command error is given by equation (2-6) where u_{ff}^{*} is the correct feedforward command for the actual plant.

$$\Delta u_{ff}^{i}(t) = u_{ff}^{i}(t) - u_{ff}^{*}(t) \qquad (2\text{-}6)$$

Given the true inverse closed loop plant dynamics, equation (2-6) is equivalent to

$$\Delta u_{ff}^{i} = P_{CL}^{-1}(x^{i}) - P_{CL}^{-1}(x_{d}) \qquad (2\text{-}7)$$

where x and x_d refer to the actual and desired position states, respectively. Equation (2-7) gives the exact command error. An estimate of the command error is obtained by using the inverse plant model.

$$\begin{aligned} \Delta\hat{u}_{ff}^{i} &= \hat{P}_{CL}^{-1}(x^{i}) - \hat{P}_{CL}^{-1}(x_{d}) \\ &= \hat{P}_{CL}^{-1}(x^{i}) - \hat{u}_{ff}^{*} \end{aligned} \qquad (2\text{-}8)$$

The formulation given here is taken directly from Atkeson's work (1988), except here the closed loop plant dynamics are used, whereas Atkeson considered the open loop plant dynamics. Given the estimate of the command error, equation (2-6) is rearranged to explicitly give the updated feedforward command:

$$\begin{aligned} u_{ff}^{i+1} = \hat{u}_{ff}^{*} &= u_{ff}^{i} - \Delta\hat{u}_{ff}^{i} \\ &= u_{ff}^{i} - \left(\hat{P}_{CL}^{-1}(x^{i}) - \hat{u}_{ff}^{*}\right) \end{aligned} \qquad (2\text{-}9)$$

In the case of a linear closed loop plant model, equation (2-9) can be written as:

$$u_{ff}^{i+1} = u_{ff}^{i} - \hat{P}_{CL}^{-1}\left(x^{i} - x_{d}\right) \qquad (2\text{-}9a)$$

It is useful to introduce a learning rate ρ which is used to reduce the effect of disturbances. The equation for the updated feedforward command $u_{ff}^{i+1}(t)$, is then given by equation (2-9b)

$$\begin{aligned} u_{ff}^{i+1} &= (1-\rho)u_{ff}^{i} + \rho\left(u_{ff}^{i} - \Delta\hat{u}_{ff}^{i}\right) \\ &= u_{ff}^{i} - \rho\hat{P}_{CL}^{-1}\left(x^{i} - x_{d}\right) \\ &= u_{ff}^{i} - \rho\hat{P}_{CL}^{-1}\left(x_{err}^{i}\right) \end{aligned} \qquad (2\text{-}9b)$$

Equation (2-9b) is the update equation that was used in the experiments of this thesis. With the use of a learning rate, the feedforward command is only updated with a fraction of the estimate of the command error. This smoothes out the learning algorithm's response to disturbances, as random errors will not have a significant effect on the feedforward command, while after several iterations, repeatable errors will dominate the majority of the change in $u_{ff}(t)$. In the actual flight experiments, a learning rate of 0.5 was used.

2.3.5 Reason for Consideration of the Closed Loop Dynamics

A key point of this thesis is that it was necessary to take full account of the actions of the feedback controller. Typical trajectory learning methods handle the feedback controller by providing the desired trajectory for all states to the feedback controller. The feedback controller operates according to the full state error between the desired and actual trajectories $\left(X(t) - X_d(t)\right)$. These algorithms anticipate that as the trajectory converges to the correct one, all states will approach the desired states $X_d(t)$ given to the feedback controller, and the feedback command will approach zero.

This assumption is generally true for robot arms, where the state variables (joint angles and derivatives: $\theta, \dot{\theta}, \ddot{\theta}$) have a definitive dependence on one another. In this case, modeling errors occur only between the command (joint torque: τ) and joint acceleration $\ddot{\theta}$, and therefore can be accurately approximated by input command disturbances. Trajectories for robots are often specified in terms of joint angles $\theta(t)$. Despite modeling errors, it is still possible to construct the trajectories of the other states $\dot{\theta}(t), \ddot{\theta}(t)$, because these derivatives have an inherent correspondence. In the case of the helicopter, however, modeling errors occurred *between* several states. In other words, dynamics which were not known exactly existed between states. In light of modeling errors, it was

not possible to accurately construct the trajectories of the other states, given the trajectory of only one state (x / y position in the case of the helicopter). For example, the attitude history of the vehicle was not perfectly known given the translational velocity history of the vehicle.

To date, all of the iterative control work reviewed provides full desired states to the feedback controller. This is not to say that the other approaches ignore the feedback controller. On the contrary, many update methods [Atkeson, 1988; Arimoto, 1990; Ahn et al., 1992] tried to take advantage of the feedback controller by considering the feedback command $u^i_{fb}(t)$ when estimating the feedforward command error $\Delta u^i_{ff}(t)$. The full closed loop dynamics, however, were not considered in these approaches. Constructive use of the feedback command $u^i_{fb}(t)$ was not given up in the method used in this thesis as the command update was made according to the output error, which inherently incorporated the positive effect of the feedback command.

2.3.6 Limitations of this Algorithm

One important limitation of this algorithm is that it inherently requires a SISO description of the plant. The SISO plant must, of course, be controllable and observable. Plants which have a non-unique inverse relationship (e. g. redundant actuators), can not be used with this trajectory learning algorithm. The inverse must be capable of uniquely mapping a trajectory error into a command correction. Though not considered in this thesis, it should be possible to adapt this algorithm to 'square' plants (e. g. two inputs, two outputs), which have a unique inverse. An inverse model which can be constrained so that it provides a unique mapping, is also acceptable.

The update algorithm described in this chapter is applicable to non-linear as well as linear plants. Only the particular methods used to apply the algorithm rely on a linear model. Equivalent methods may be developed for the initialization, and command update steps in the non-linear case.

Chapter 3

Helicopter Testbed

Flight experiments with the helicopter testbed were conducted in an indoor lab facility. The helicopter testbed is intended to provide an inexpensive and accessible platform for the development of various control algorithms over a wide range of maneuvers. This chapter describes the configuration of the entire experimental setup. Figure 3.1 displays a schematic of the laboratory layout.

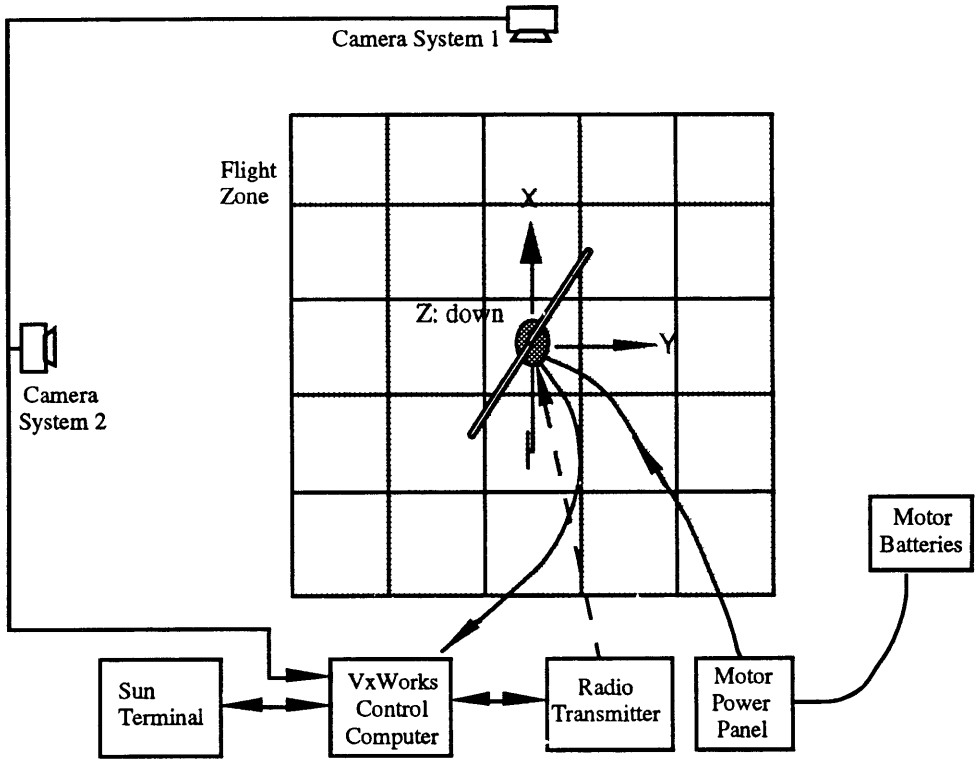

Figure 3.1: Laboratory Layout of the Helicopter Testbed

The entire testbed system consisted of the helicopter chassis with onboard sensors, ground battery power, an external video based positioning system, radio uplink, user interface terminal, and a real time control computer system. A Linear Quadratic Gaussian feedback control system was used to perform state estimation and hover stabilization. The feedback compensator was left unaltered by the feedforward control produced in trajectory learning. The helicopter was a free flight vehicle, with the exceptions of power and data tethers, and a safety leash, which were considered to have a negligible effect on flight characteristics. For the experiments, a three meter square flight zone was sufficient. This area could easily be expanded by changing the cameras' fields of view, and lengthening the tethers. Each of the systems used in the testbed is described in this chapter.

3.1 Helicopter Chassis

The experimental platform was based on a commercially available radio control (RC) helicopter model, a Miniature Aircraft XCell 60. These helicopters are inexpensive (<$1000.00), relatively easy to maintain, operate, and repair. The XCell is an extremely agile vehicle. In the hands of expert human pilots, they are capable of maneuvers such as rolls, loops, inverted flight, spins and flips (rotation about the roll or pitch axis with minimum translation).

It was necessary to modify the helicopter in several aspects. The helicopter was converted to electric propulsion, outfitted with an instrumentation payload, and the standard RC actuators were interfaced to the computer. One of the concerns in performing these modifications was to change the characteristics of the vehicle as little as possible. In this manner, the computer was presented with the same control challenge as

the human pilot. For testing and safety reasons, it was also desirable to retain the capacity for a human to easily take control of the helicopter.

3.1.1 Mechanical Description

The XCell 60 is based on an aluminum and composite structure. Figure 3-2 shows the helicopter in its commercially available form. Main rotor diameter is 1.42 meters. Ready to fly weight in this form is 5 kg. As the photo shows, the helicopter has a standard configuration with a main rotor, tail rotor and main fuselage. The helicopter chassis and motor have a total of five control inputs, which are described below.

Figure 3-2. Miniature Aircraft XCell 60

An important source of dynamics in a helicopter is the rotor head and blades. The XCell 60 is equipped with a two bladed rotor head based on a flybar type Bell/Hiller mixing system. This mixing system incorporates standard cyclic and collective control actuation through a swashplate [Johnson, 1980]. The rotor head is articulated to allow damped teetering and lag motion of the blades. The blades themselves are constructed of composites and have a flat bottom airfoil. It was hoped that this design would exhibit dynamics similar in character to those of full size helicopters.

The main-blade control system was abstracted into three mechanical inputs for the servo actuators. The inputs were roll (or aileron) command, which activated lateral cyclic, pitch (elevator) command, which activated longitudinal cyclic, and collective command, which changed collective blade pitch. A fourth mechanical input varied the pitch of the tail rotor blades. This was referred to as the tail (rudder) command. These controls were configured according to the manufacturer's instructions without modification.

3.1.2 Electric Motor Conversion

The standard powerplant for XCell 60 helicopter is a 0.60 cubic inch two-cycle, internal combustion engine. Typically, these methanol fueled engines produce 2 horsepower (hp), peak, and spin at 15,000 rpm. These engines have an attractive power to weight ratio (including a consideration for fuel). The exhaust and flammable fuel associated with the operation of this engine, however, were incompatible with the confines of the indoor laboratory. This engine was replaced by its closest equivalent in available electric motors.

The motor selected was a HecktoPlett 355. Peak power input to this motor was approximately 2.68 hp (2000 watts). Efficiency of the motor was estimated by the manufacturer to be approximately 83% at these power levels, resulting in a peak output power of 2.2 hp. The motor was a DC brush type motor with neodymium magnets. Motor speed was not controlled directly. A high frequency switching controller allowed adjustment of motor power by varying the duty cycle of current to the motor. This unit was a Zeta Xtra speed controller produced by Product Design, Inc. The Xtra provided the fifth input for the helicopter control system. It gave the user indirect control of motor power (indirect because the duty cycle was not directly related to motor power).

Handling qualities of the electric helicopter were comparable to those of the internal combustion engine version. An approximate payload capacity of 7 kg. was also maintained by the electric version.

In hover, the helicopter motor drew approximately 1400 watts while carrying the full instrumentation payload (2 kg). Four lead acid batteries on the ground provided power for the motor through a power tether. Nicad batteries had successfully been used as an onboard power source, but at the expense of greatly reduced flight times, reduced payload capacity (or reserve power), and time consuming recharge cycles. The lead acid batteries were rated at over 100Ah, and capable of powering the helicopter for over two hours. Three 12 volt and one six volt battery were connected in series, providing a nominal 42 volts to the motor. In actual use the measured voltage at the helicopter was approximately 38 volts, probably due to losses in the tether, ammeter, connectors, and batteries themselves.

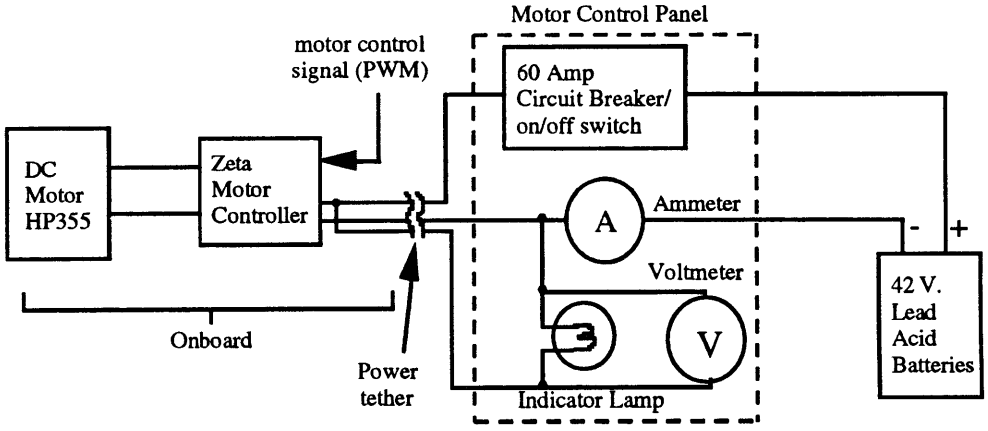

Figure 3-3. Electric Motor Connections

A control panel switched and monitored power to the helicopter motor. Figure 3-3 shows the electrical connections and control panel. The panel provided voltage and current measurements and 60 amp circuit breaker protection. The circuit breaker was also used as a safety cutoff and on/off switch by the operator. Another safety feature which

prevented a runaway helicopter was a 1.25 meter long steel safety cable. The cable served as a leash by connecting a hard point of the helicopter chassis to a fixture secured in the center of the lab floor.

3.2 Helicopter Control Hardware

An overview of the electronic hardware used in the control system of the helicopter is shown in figure 3-4. This section describes the hardware used for measurement, control computation, and actuation. Hardware components were located on the ground as well as carried onboard by the helicopter.

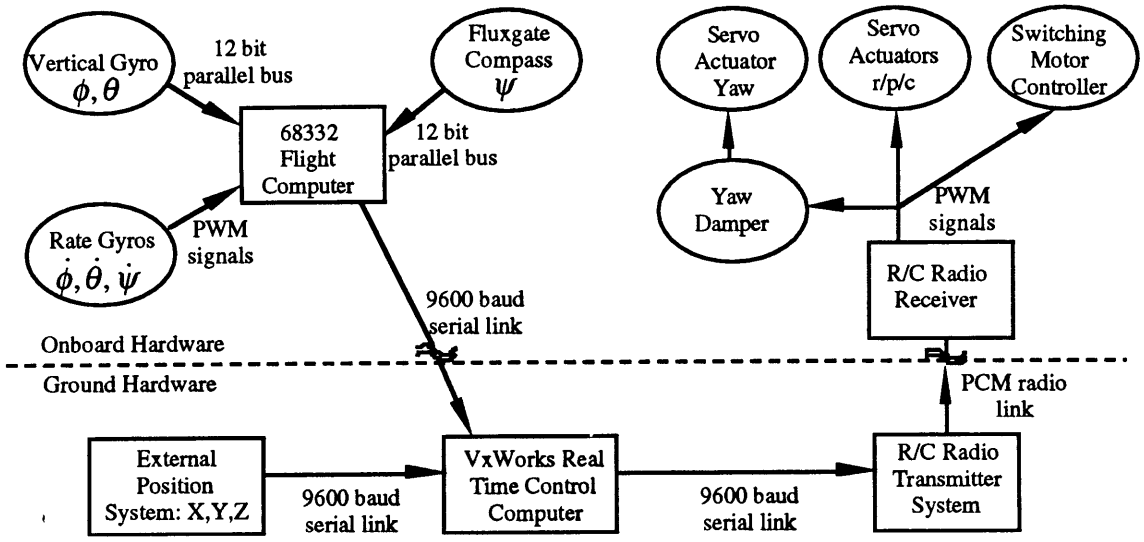

Figure 3-4. Helicopter Hardware Overview

3.2.1 Radio Control System

A commercially available radio control (RC) system (Futaba FPT7-UHPS) was used to translate computer generated commands into actuator movements. The RC system consisted of a radio transmitter, receiver, and electronic servo actuators. The standard RC system operates at a 50Hz servo rate. This is identical to the system used by human

pilots to control the model helicopter. The standard transmitter uses joystick controls for the human to input commands. A modification to the transmitter allowed the computer to send commands to the helicopter in place of the joystick signals. Control through the joysticks could be regained through an override switch located on the transmitter. This configuration was chosen to ease recovery by a human pilot in the event of computer errors.

3.2.1.1 Actuators

The servo actuators are self contained units which incorporate high torque, geared electric motors, position measurement, and an integrated proportional, derivative control system [Ahmad, 1991]. There were four servos on board, connected to the helicopters mechanical linkages for swashplate control (roll, pitch, and collective) and tail rotor pitch control. The servos were commanded by pulse width modulation (PWM) signals which were generated by the radio receiver. Overall, the servos' mechanical motion with the exception of the yaw servo, was directly proportional to the commands generated at the transmitter, either by the joysticks or the computer.

An onboard rate gyro system provided yaw damping independent of our control algorithms. This feature has become standard equipment on RC helicopters, reducing the RC pilot's workload required for yaw control. When the helicopter is in a static condition, the yaw servo's behavior is identical to that of the other servos - its motion is directly proportional to the transmitter commands. When the helicopter yaws, the rate gyro modifies the servo command in a manner that damps the yaw motion. This feature was retained in order to ease human operation of the helicopter as well as to present the computer with the same dynamics as a the human would face.

A fifth channel of the receiver commanded the motor speed controller. This was referred to as the throttle command. The speed controller's functionality was described in the section on the electric motor conversion.

3.2.1.2 Transmitter and Receiver

The radio transmitter sends the control signals from the operator or computer to the receiver on board the helicopter. The system uses a 72MHz FM signal and incorporates pulse coded modulation (PCM) and error checking. The transmitter has two mixing features which are commonly used by R/C helicopter pilots, so they were retained in our setup. The transmitter was modified so that the control computer could send commands to the receiver in place of the human pilot.

Collective pitch mixing caused the collective servo command to be mixed with throttle command. This provided the pilot with a single 'thrust' command (physically one potentiometer of a joystick), which simultaneously adjusts collective pitch and throttle. The mixing curve was ideally set to maintain constant rotor speed. Direct rotor speed regulation had not been used in these experiments, though some experimentation with commercial regulators was conducted. The commercial regulators were not reliable or precise, so it was decided not to use them. The D.C. motor combined with the mixing feature allowed the throttle curve to be set such that constant rotor speed was maintained at all power settings. In these experiments, the mixing function was set to throttle the motor to maintain a constant 1800 rpm throughout the collective pitch range. This setup was validated under static flight conditions.

Revolution mixing is intended to compensate for the change in torque due to throttle changes. This mixing offsets the tail rotor command according to the throttle setting.

The mixing curve was set so that tail rotor thrust balanced main rotor torque at all power settings in steady-state, hovering flight conditions.

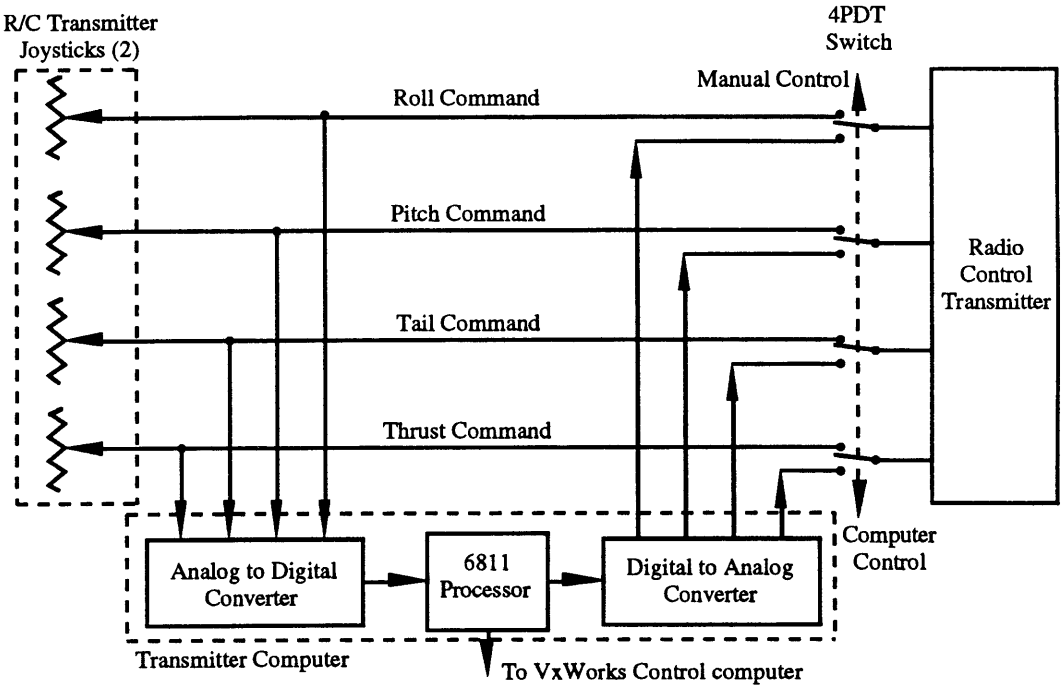

Figure 3-5. Computer/Transmitter Interface

The transmitter was modified by tapping into its four analog joystick signals. Humans typically fly the helicopter through the use of two joysticks (four controls). The modification allowed easy switching between human and computer control. In the human mode, joystick potentiometer signals went to the R/C transmitter's processor in a normal fashion. In the computer control mode, those signals were cut off and analog signals were substituted from our transmitter computer (generated by four digital to analog converters). Figure 3-5 illustrates this setup. Because the computer accesses the transmitter in this manner, all mixing functions were retained, i.e. the computer commands had exactly the same effect as human commands. At all times, the joystick commands were read into the transmitter computer (and subsequently transmitted to the control computer) through analog to digital converters. This allowed for observing the human pilot's control actions. The analog interface with the transmitter was provided by

the transmitter computer, which consisted of a 6811 processor board running four 12 bit A/D and four 12 bit D/A converters. Communication with the main control computer (VxWorks system) was provided via a 9600 baud serial communication line.

3.2.2 Instrumentation

Instrumentation consisted of an offboard positioning system, onboard rate gyros, compass, and vertical gyro. Onboard instruments were mounted on an instrumentation platform, which attached under the chassis, and had a mass of approximately 2 kg with instruments. These instruments provided attitude and attitude rate information. The external position system was based on video image processing and provided 3D position information. Figure 3-4 illustrates the connections between the various systems.

3.2.2.1 Onboard Instruments

Figure 3-7 shows the layout of the instrumentation on the instrument platform. Figure 3-6 is a picture of the helicopter mounted to the platform. The platform was constructed from plywood and attached to the bottom of the helicopter between the chassis and the landing gear. Information from the onboard sensors was collected by the flight computer and transmitted to the control computer via a 9600 baud serial line. In addition to this data tether, a power tether provided electricity from power supplies on the ground to the instrumentation. Batteries could have been used for onboard power, but would have added another kilogram to the airborne payload. The power junction box filtered and distributed power to all of the onboard systems from either onboard batteries or the ground power supply.

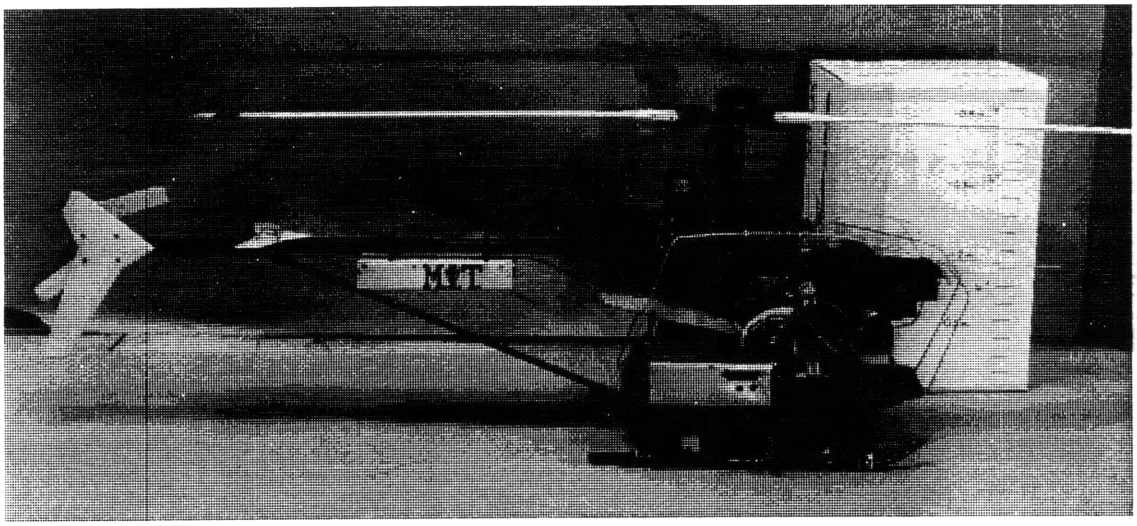

Figure 3-6. Instrumentation Platform Picture

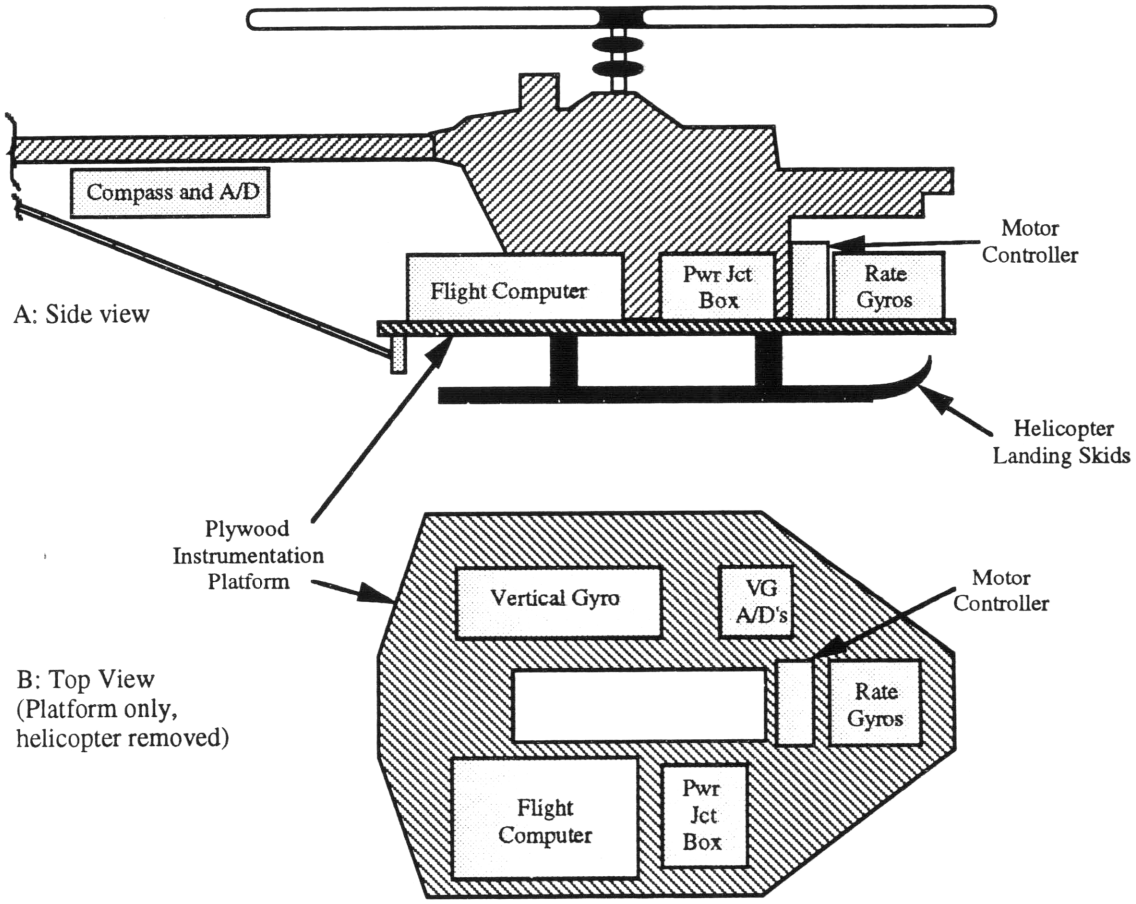

Figure 3-7. Instrumentation Platform Diagram

A mechanical vertical gyro (VG) provided absolute roll and pitch attitude information. The gyro was a Humphrey model VG34-0313-1, which claims a maximum free drift of one degree per minute. Mass of the unit is 0.59 kg. The gyro is equipped with torquers and level sensing devices which improve actual performance well over the free drift specification. Roll and pitch measurements (euler angle) were taken from the VG as analog signals from potentiometers mounted directly on the gimbals. The analog signals were converted to 12 bit digital information by two dedicated A/D converters.

Three mechanical rate gyros measured roll, pitch, and yaw attitude rates. The gyros were R/C hobby gyros similar to the helicopter's built-in yaw damper gyro. The signal from these gyros was generated in a PWM form which was measured by the flight computer. Specifically, JMW model III gyros were used. The JMW gyros are inexpensive and designed to withstand the harsh environment of the helicopter. Specifications on accuracy and dynamic range were not available from the manufacturer. Watson Industries solid state rate gyros (from the Watson AHRS-C300A package) were also tested. Unfortunately, due to the configuration of the piezo element, these gyros are generally susceptible to high frequency translational accelerations (vibrations). Despite attempts to shock mount the Watson gyros, their performance was far worse than that of the JMW's. Solid state rate gyros with improved immunity to vibrations (such as the Murata GyroStar) have recently become available. Foxlin's report (1993) evaluates several available rate gyro models, and tests were performed on some units. Improved gyros would increase the dynamic range and accuracy of the testbed's rate measurement capability.

A KVH C100 fluxgate compass provided magnetic heading information. The analog output of the compass was used. The manufacturer's specifications claim one degree precision when the unit is gravity level. The analog signal was converted to 12 bit digital

information by a dedicated A/D converter. The compass and converter were mounted on the aluminum tail boom in order to minimize magnetic disturbances created by the motor and other onboard systems. The compass does provide a more precise (0.5°) serial data output, but the capability to read an additional serial line was not available. Although accuracy of the compass does degrade rapidly for bank angles exceeding ± 16°, this effect was found to be insignificant for the short periods in which the helicopter exceeded these limits.

The flight computer was based on a 68332 processor which collected data from the onboard sensors and transmitted it to the VxWorks control computer over a 9600 baud serial line. All data was transmitted in a packet format at 50Hz. Vertical gyro and compass data was collected at 50Hz, while rate gyro data was read into the flight computer at 250 Hz. Since the helicopter vibrations had the greatest effect on rate measurements, the rate gyro data was filtered at 15 Hz. using a second order low pass Butterworth filter. The most recent data from all sensors was included in the packet sent to VxWorks.

3.2.2.2 External Position System

A two camera color video tracking system provided 3D (x, y, and z) helicopter position information. The two cameras were positioned on adjacent sides of the square flying area, as shown in figure 3-1. The helicopter was covered with a fluorescent orange sheath, as shown in figure 3-8. The covering made it possible for both camera systems to distinguish the helicopter from the background and track it. Each camera system performed its own real time video processing. The processors in each system transmitted their data via a serial link to the VxWorks control computer. Software running on the

control computer used this information to resolve the 3D position of the helicopter in the room.

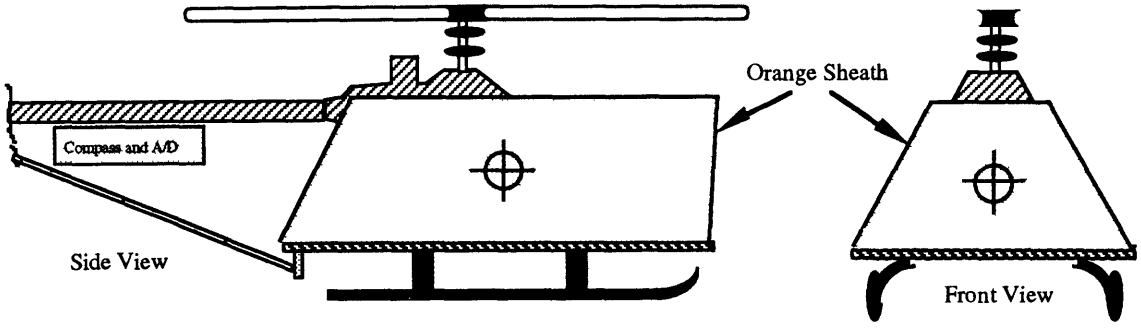

Figure 3-8. Covering on Helicopter for Position Tracking System

Each camera system consists of a color video camera and processing hardware centered around a 68332 CPU [Wright, 1993]. The system operates by examining the camera's video signal as it is generated. The processing scheme is based on a thresholding method which looks for horizontal strings of the image which have a color and intensity within a preselected range. For the helicopter, the system was programmed to recognize bright orange. The position of each string is recorded in terms of the coordinates that it appears in the camera's field of view. All other video information is ignored. The video processor compiles neighboring strings into one object. The centroid of the object is computed in camera coordinates, and transmitted to the control computer. The helicopter covering was arranged so that the centroid of the image (designated by the crosshairs in figure 3-8) approximated the vehicle's center of mass. This relatively simple scheme is analogous to looking for the brightest object in a black and white image.

3.2.2.3 Computation Hardware

The computational structure centered around the VxWorks control computer. The computer ran a VxWorks real time operating system, which allowed fast processing for control. The hardware consisted of two processors and a shared memory. A Sun

workstation provided a user interface and the working environment. Satellite processors provided sensor information and communicated with the helicopter. These processors were the flight computer, transmitter computer, and the two vision processors. All communication occurred via serial links. A block diagram of this arrangement is shown in figure 3-9.

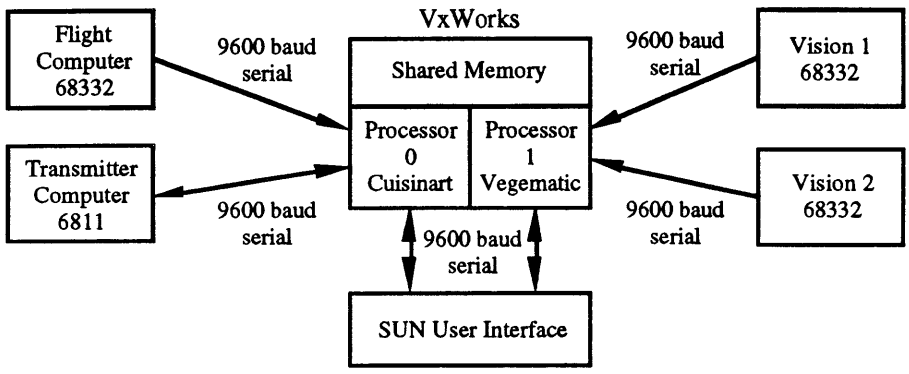

Figure 3-9. Computer Hardware Diagram

The control computer uses two 68040 CPU's and a shared memory in a VME bus configuration. Processor 0, or cuisinart, collected data from the flight computer and the transmitter. This processor sent commands out to the transmitter. Cuisinart was also used to interact with the operator. It displayed updates of the sensors and the controller modes. Commands to start and end a flight, set the controller parameters, and collect and save data, were entered through cuisinart. Processor 1, or vegematic, collected and processed the vision data, and computed the commands. Commands were computed according to all the sensor data, the controller mode (e.g. takeoff, hover, land, run trajectory), and controller parameters, which were set or determined by cuisinart.

The controller cycle operated at 50 Hz. All sensor data was collected at 50 Hz or greater. The servo rate was selected according to the control rate of the standard helicopter radio control system (50Hz). An important comment on this servo rate is that helicopter vibrations are mostly related to the main rotor. Turning at 1800 rpm, a significant signal

was produced at 30Hz. As mentioned earlier, the vibrations appeared mainly in the rate gyro signal, which was therefore sampled at 250 Hz and then filtered. It is believed that this setup defeated any significant problems with aliasing, despite the 50 Hz servo rate.

Though the processors ran continuously, memory restrictions only allowed 1000 points (20 seconds) of data to be collected during a flight. The operator initiated the collection of a data set by issuing a command to cuisinart from the sun terminal. These data sets were stored for post flight analysis. The data set length was therefore used as the length of all trajectories. Feedforward commands were initiated and updated as 1000 point data arrays.

Since the characteristic of trajectory learning is to perform off line processing between repetitions of the trajectory, it was decided to do the learning processing on the Sun workstation. The data from every trajectory iteration was saved on a disk. The Sun used this data with the trajectory learning algorithm to compute an improved feedforward command. The feedforward command was stored in a table format as a function of trajectory time. While cuisinart and vegematic were used to do real time feedback control processing, feedforward commands were simply looked up as a function of time during the flight. When the trajectory was run on a flight, data set recording and feedforward command lookups were performed synchronously.

3.3 Feedback Controller

In order to stabilize the helicopter through the trajectory and fly the helicopter for testing, a feedback controller was designed and used during all computer controlled flights. The controller was based on linear quadratic gaussian (LQG) design techniques. The LQG controller design contains a Kalman state estimator and full state linear feedback gains

(generated from linear quadratic regulator (LQR) techniques). Independent linear models were developed for each of the four control inputs: roll cyclic, pitch cyclic, tail rotor thrust, and main rotor thrust. Through simple mode switching, the same controller successfully performed take-off, hover, and landing. Trajectory maneuvering was achieved by modifying the regulator to accept feedforward commands. Figure 3-10 is a block diagram of the control loop that was used on each of the helicopter's four axis. In the planar trajectories used in this thesis, only the roll and pitch axis made use of the feedforward command input. The yaw and z controllers were always used in their standard regulator mode.

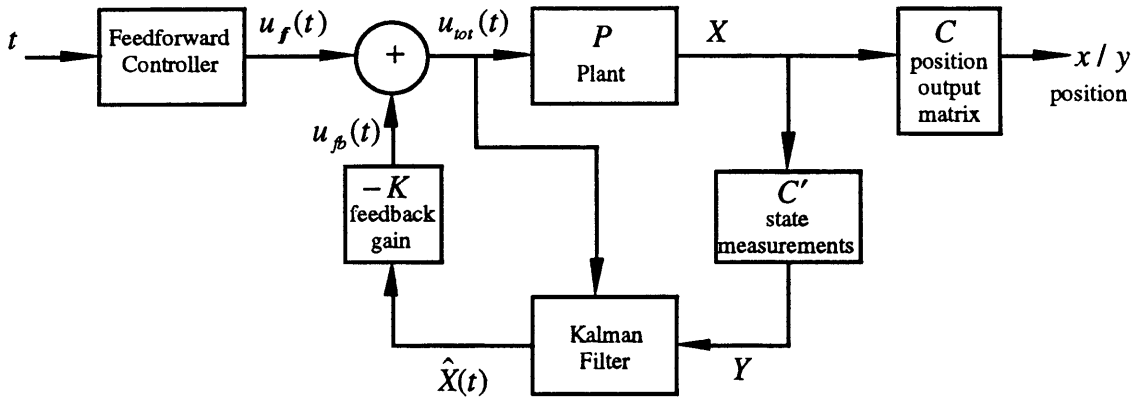

Figure 3-10. Feedback Controller with Feedforward Servo

3.3.1 Linear Modeling

Linear models for hover conditions were obtained through data analysis. Perturbation data used for modeling was gathered by adding filtered white noise to the pilot's commands. Through analysis of the helicopter's dynamics and data regressions, discrete time state space models were obtained for each of the control axis: lateral, longitudinal, vertical, and yaw (heading). Each axis was decoupled from the others and had a single control input. The control inputs and states for each axis is listed in table 3-1.

One significant discovery in the data analysis was the presence of significant delays (0.12 seconds) from the commands to the helicopter response. This could possibly be attributed to delays in communications with the RC receiver and actuator delays. The delays were handled by incorporating extra 'delayed' states in our models [Astrom and Wittenmark 1984]. Though the models used in these tests assumed uncoupled dynamics, helicopters inherently exhibit strong coupling characteristics and it was intended to have the learning controller account for the coupling. However, it should be noted that a controller based on a coupled roll and pitch model was implemented with no significant performance improvement in hover.

Axis:	Pitch	Roll	Yaw	Z
control input	pitch cyclic command	roll cyclic command	tail command	thrust command
states 1-7	delayed pitch commands	delayed roll commands	delayed tail commands	delayed thrust commands
state 8	pitch rate	roll rate	yaw rate	z velocity
state 9	delayed rate	delayed rate	delayed rate	delayed velocity
state 10	pitch	roll	yaw	z
state 11	x velocity	y velocity	n/a	n/a
state 12	x	y	n/a	n/a

Table 3-1. State variables for each control axis

Perturbation data was taken by using a human pilot to control the helicopter. The pilot's goal was to hold the helicopter in hover. Perturbations were constructed off-line by filtering white noise with a 5th order 10 Hz, low pass butterworth filter. The perturbations were stored as 1000 point data arrays. Several different random sets were used throughout the modeling process. During a flight, perturbations were synchronized

with the initiation of a data set. At each timestep during the data set, the pilot's commands were read into the VxWorks control computer and modified by adding in the appropriate perturbation value from the array. The modified pilot command was the actual command sent to the helicopter. The commands and all of the sensor measurements were stored as part of the data set. Perturbation flights were conducted on each axis individually. In this manner, data was collected showing the input / output relationship for each axis.

Linear models were created for each axis by performing regressions on the input/output data. The regressions were used to determine the relationship between the states in each of the models. Selection of state variables was determined through physical insight and regression analysis. The states for each of the axis is listed in table 3-1.

3.3.2 LQG Compensator Design

An LQG compensator was used to stabilize the helicopter in hover. This form of feedback control is based on combining a Kalman Filter with full state LQR feedback gains. The compensator for each axis was designed according to the respective linear state space model.

Linear steady state KF's were implemented to filter measurement noise and estimate translational velocities. The KF parameters were calculated according to estimates of sensor and plant noise, as well as the plant model. The feedback gains were computed from the LQR design process. The LQR design parameters were chosen to heavily penalize deviations in x, y, z, and yaw. Fine tuning of the parameters occurred through actual flight testing.

The LQG control structure provided an excellent base from which to conduct learning experiments. With the complete hover controller (feedback controllers running on each of the four axis) the system provided a root mean squared error (rms error) in position of less than 0.04 m. Adequate performance was demonstrated with respect to wind gusts, weight changes, and control perturbations. If performance improvements became necessary, dynamics such as coupling and ground effect would be taken into account. This system, however, provided sufficient performance and allowed training and flight testing of the learning algorithms to be conducted.

Chapter 4

Trajectory Learning Experiments

In-flight experiments of the trajectory learning algorithm described in chapter 2 were conducted on the helicopter testbed. These experiments involved several different planar trajectories. The learning algorithm was run independently on the roll and pitch axis. Initial feedforward commands were optimally computed according to the linear hover model for each respective axis. The update algorithm utilized the linear closed loop models. The flight experiments demonstrated a dramatic improvement after only a few learning iterations. This chapter concentrates on the results of experiments with one trajectory, but it is representative of all the trajectories performed.

4.1 The Square Trajectory

Trajectories were generated in a two step process. First a goal trajectory was generated simply by specifying a desired shape or pattern. The trajectory used for these results was essentially a square in the helicopter's x-y plane, centered in the helicopter test area. This pattern was created in ignorance of the plant dynamics: i.e. it was not necessarily dynamically possible for the plant model to perform the goal trajectory. The second step was to apply the optimization routine to develop an achievable desired trajectory for the

plant model. The achievable trajectory $x_d(t)$ and associated initial feedforward command $u_{ff}^0(t)$ provided by the optimization method were used for the flight experiments.

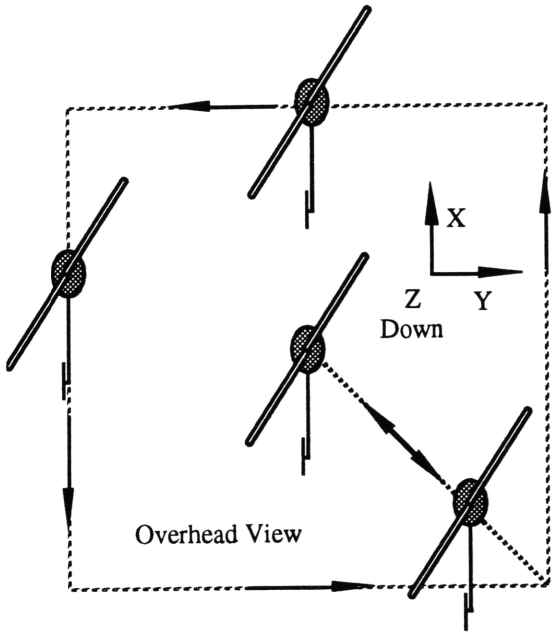

Figure 4-1. Diagram of Square Trajectory Showing Helicopter Orientation and Motion

Figure 4-1 is a diagram of the helicopter's motion and orientation over the trajectory. In the square trajectory, the helicopter started from the center of the test area, moved out along a diagonal to a corner, then moved from corner to corner counterclockwise, with a one second pause at each corner, before moving back to the center along a diagonal after returning to the first corner. Throughout the maneuver, the helicopter maintained the same heading.

Figure 4-2 shows an x-y plot of the trajectory as viewed from above the helicopter. The trajectory specifies the motion of the helicopter as measured by the position system, which approximates the location of the center of mass. The solid line represents the goal trajectory, while the dashed line is the actual desired trajectory computed by the optimization routine. The trajectories are difficult to distinguish (the plots fall on top of one another), but at each corner, one can see a slight overshoot in the desired trajectory

(the goal trajectory is a perfect square). The overhead view does not indicate the time dependency of the trajectory, as is shown in figure 4-3 for the pitch axis. Due to the speed of transitions between corners, the optimization routine allowed the overshoots so as not to use excessive commands (this tradeoff was dictated by the cost function). Note that the x direction is the helicopter's longitudinal (pitch) axis, and the y direction is the helicopter's lateral (roll) axis.

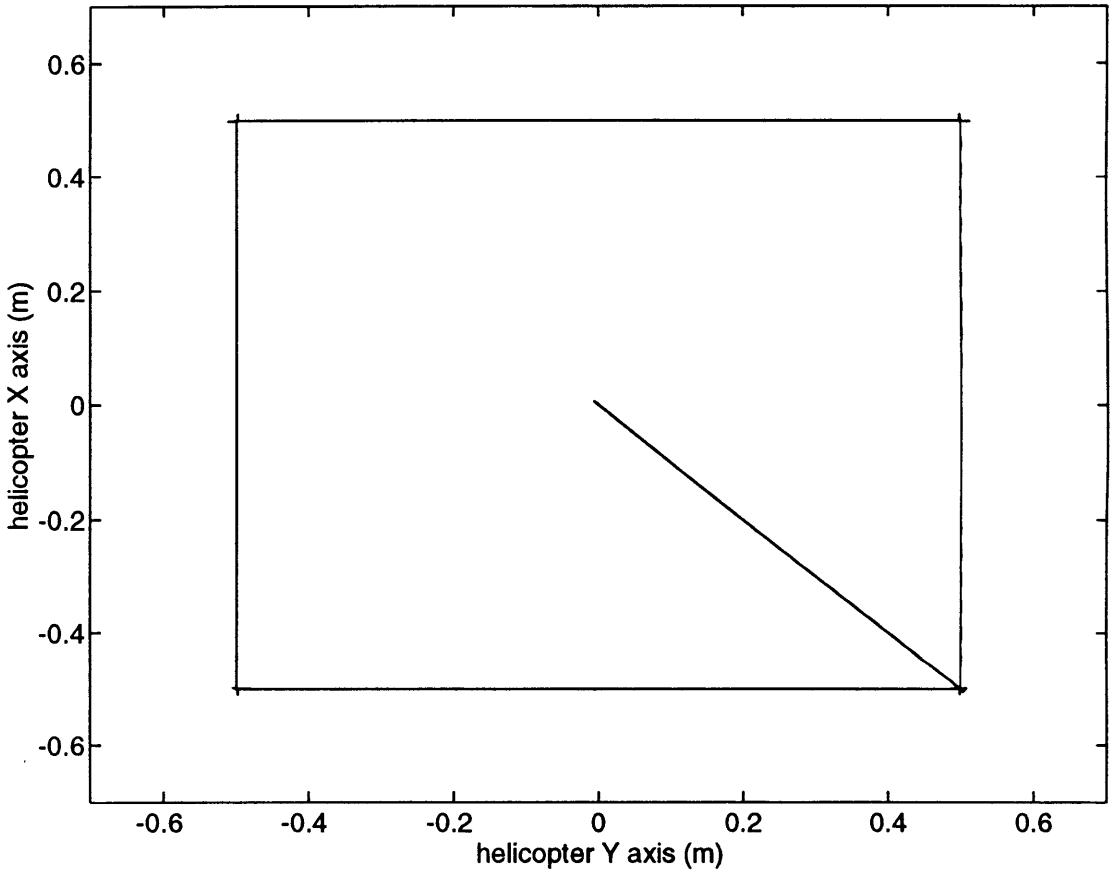

Figure 4-2. Top View of Square Trajectory, goal and desired

Figure 4-3 shows parameters for the pitch axis trajectory over time. The trajectory was created such that transitions between corners took 1.25 seconds. The first plot shows the goal $x_g(t)$ and desired $x_d(t)$ position. Once again the plots practically fall on top of one another. At each corner, the helicopter paused for 1 second. What appears to be longer pauses (i.e. between samples 280 and 420) are actually times when the helicopter was

moving along the y-axis sides of the square. Sine functions were used for the transitions between points so that the optimization routine could more easily match the goal trajectory. A short period of hovering in the center was also included at the beginning and end of the trajectory. The trajectory was created by the matlab code square.m included in Appendix C.

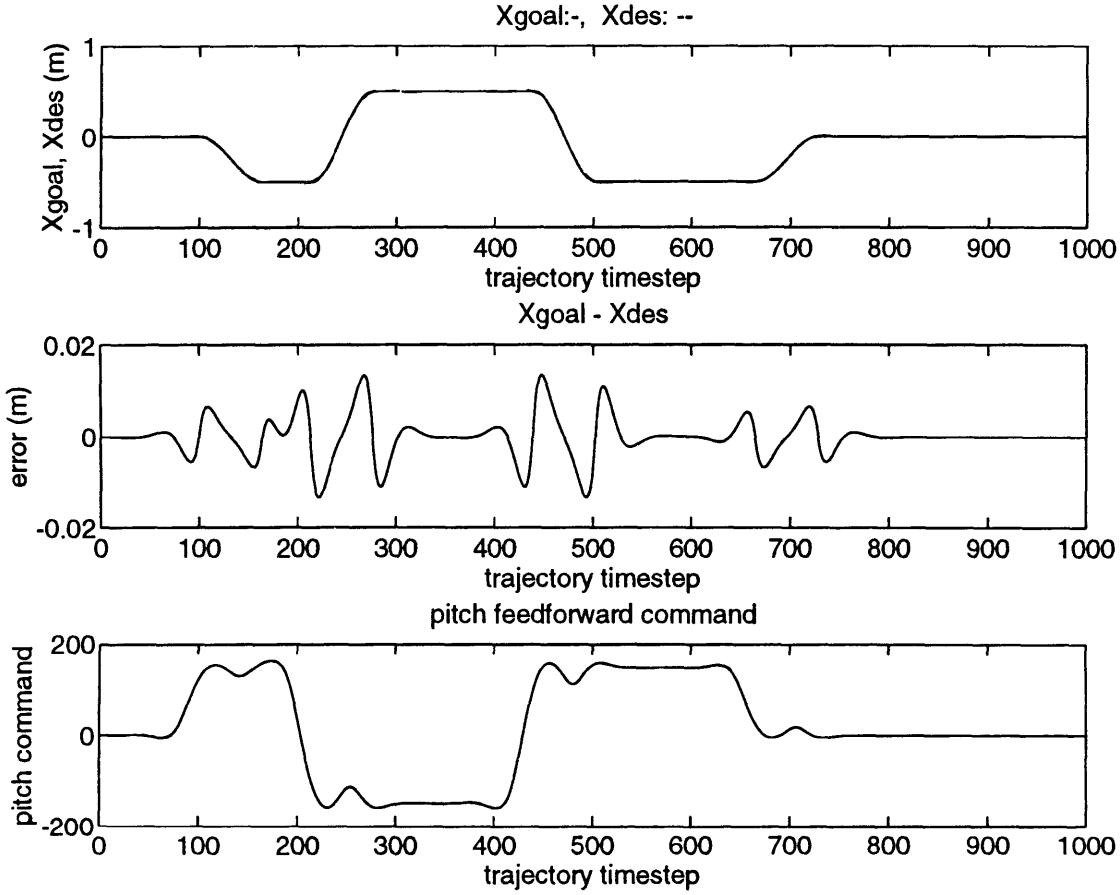

Figure 4-3. Pitch Axis Components of the Initial Trajectory

Once the goal trajectory was created, the optimization method was used to create a physically feasible trajectory. The dashed lines in figures 4-2 and 4-3 show the optimization routine's results. The second plot in figure 4-3 shows the difference between the goal and desired trajectories $\left(x_d(t) - x_g(t)\right)$. Figure 4-3 also shows the initial pitch feedforward command that was computed. The command is scaled in A/D units,

and was used this way in the linear models. The optimization routine was run independently for the roll axis.

4.2 Initial Execution

Trajectories were started after the helicopter took off and settled into a hover under pure feedback control. Once the trajectory was complete, control returned to feedback only, and the helicopter remained in hover for a short period of time. On the first execution, the helicopter produced the results shown in figure 4-4. The dashed line shows the desired trajectory. Figure 4-5 shows the position errors over time for each of the axis. The initial execution was repeated several times to ensure an accurate sampling. The figures show a representative flight. The average root mean squared error (rms error) was 0.142 m. on the x axis (rms_x) and 0.096 m. on the y axis (rms_y). The rms error for the trajectories RMS_{Traj} was calculated according to equation (4-1) where N is the number of samples. In equation (4-1) x signifies the position state of the system.

$$RMS_{Traj} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(x(n) - x_d(n)\right)^2} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(x_{err}(n)\right)^2} \qquad (4\text{-}1)$$

In comparison, the rms error that existed in hover under pure feedback control was rms_x: 0.0177 m.; rms_y: 0.041 m. The hover rms error RMS_{Hover} was calculated in a different manner as specified by equation (4-2). This calculation removed the mean hover position before computing the rms error. Daily bias changes in the instrumentation would often cause mean hover errors to vary significantly. The mean was removed in order to evaluate the true position fluctuations of the helicopter in hover, which would be more comparable to the position errors that occur over a dynamic trajectory.

$$RMS_{Hover} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(x(n) - \frac{1}{N}\sum_{n=1}^{N}x(n)\right)^2} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x(n) - mean(x))^2} \quad (4\text{-}2)$$

The mean hover error could have been removed for the actual flights by carefully adjusting bias parameters in the control computer. The method used here was much less time consuming. Though instrumentation bias errors do affect trajectory performance, the learning algorithm quickly compensated for them, allowing a fair comparison between RMS_{Traj} and RMS_{Hover}.

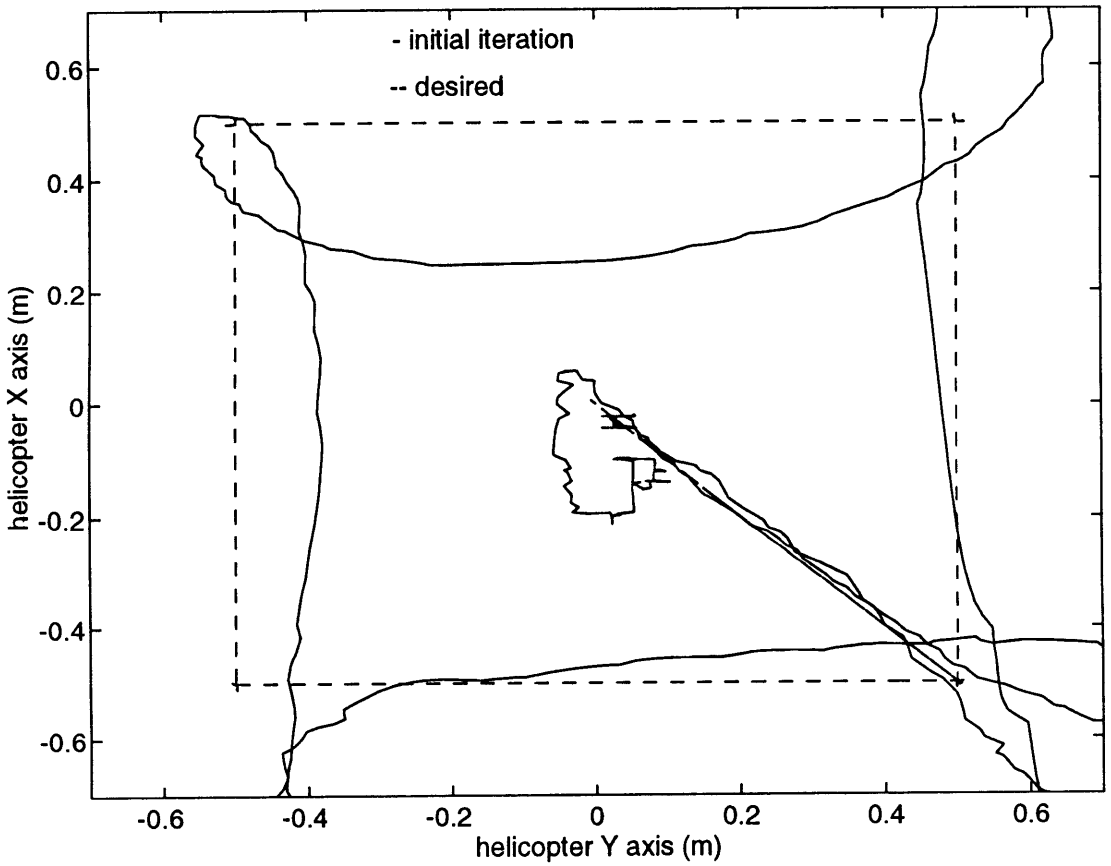

Figure 4-4. Initial Execution of Trajectory

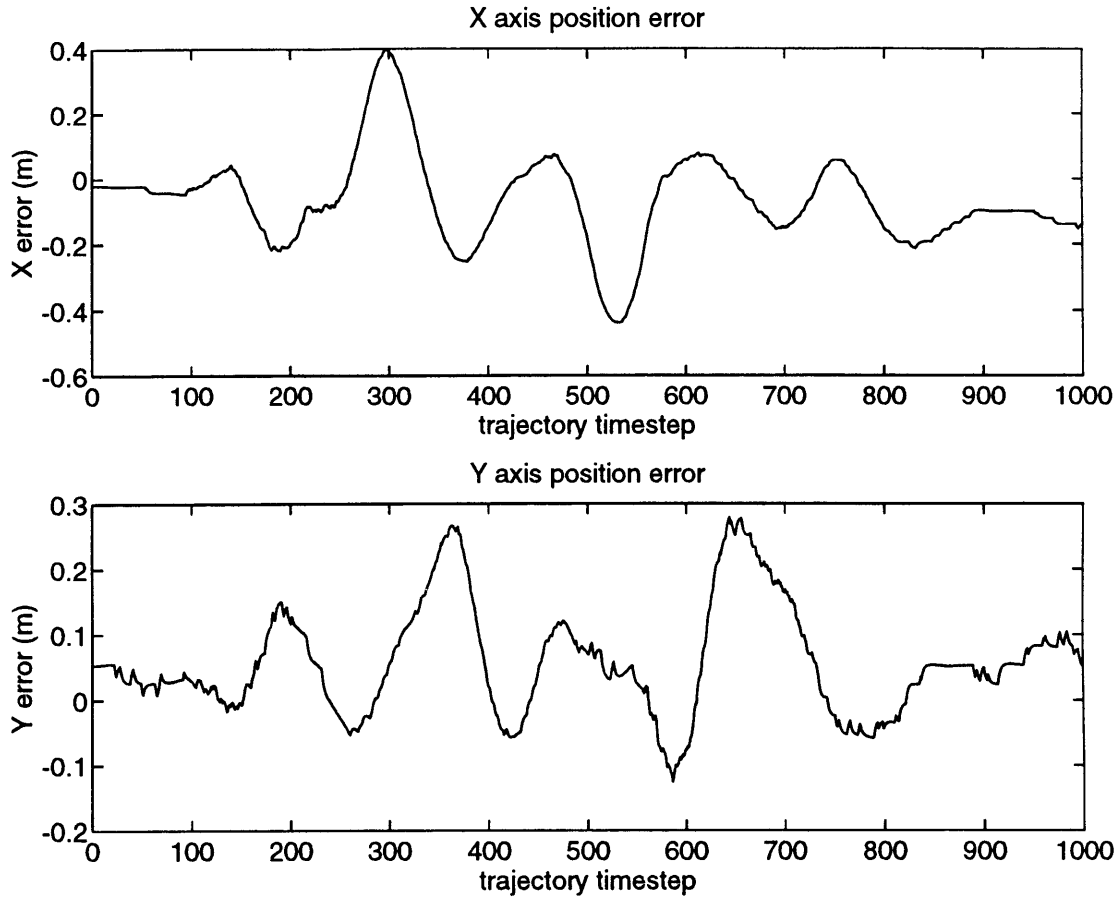

Figure 4-5. Initial Trajectory Errors

4.3 First Feedforward Command Update

The learning algorithm was independently applied to the roll and pitch axis. This section examines the pitch axis. The first feedforward command update $\Delta\hat{u}_{ff}^{0}$ was computed from the original position error history ($x_{err}^{0}(t) = x^{0}(t) - x_{d}(t)$) which is shown in the first plot of figure 4-6 as the solid line. As described in Chapter 2, the data had to be preprocessed in order for the learning operator to produce a reasonable update. Despite the filtering and initial error removal, it was necessary to take several additional steps before updating the feedforward command $u_{ff}^{0}(t)$. These steps were developed as an ad hoc method for handling the problems encountered with the actual flight experiments.

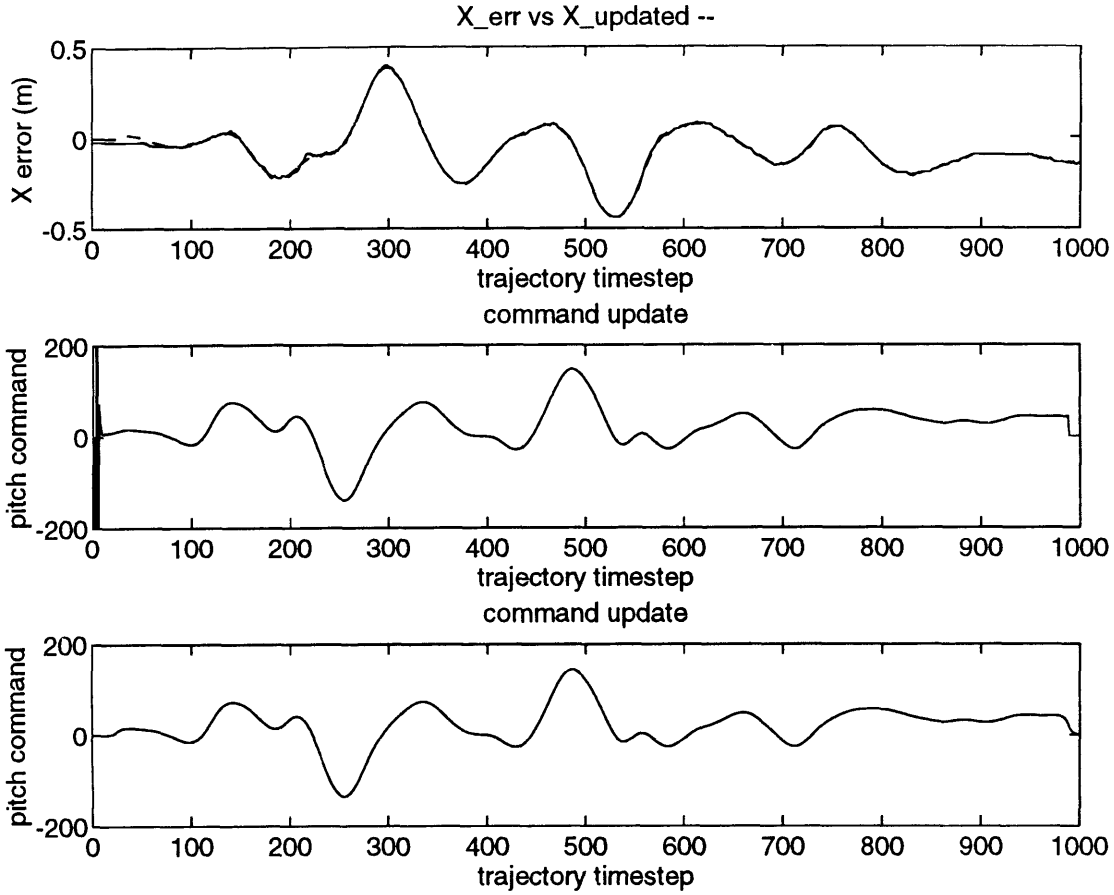

Figure 4-6. X Position Error and the Update results

As the first step in the preprocessing, the position error was filtered as described in chapter 2: using a 6th order, 1.5 Hz low pass , zero lag filter. The next step was to set the first 11 samples of the error to zero. This was necessary because, in the plant model, it took 12 time samples for a command to affect the position state (due to delays and propagation through the other states in the discrete time model). A non-zero error in the first 11 states could not be accounted for by the inverse model, resulting in an erroneous update. From the 12th timestep onward, the initial error was removed as described in chapter 2.

Though this smoothing effort allowed the inverse model to produce an accurate update, the first few timesteps of the feedforward update $\Delta\hat{u}_{ff}^{0}$ still contained spikes as the inverse

model settled. The second plot in figure 4-6 shows the spikes in the early timesteps of the update $\Delta\hat{u}_{ff}^{0}$. The dashed line in the first plot of figure 4-6 shows the result of applying the feedforward command update $\Delta\hat{u}_{ff}^{0}$ to the closed loop plant ($x_{updated}^{i}$), which should have ideally reproduced the original position error. It is shown that even after preprocessing, much of the detail in the trajectory error was retained. This result confirms the operation of the inverse. Removal of the initial error is also demonstrated by this plot.

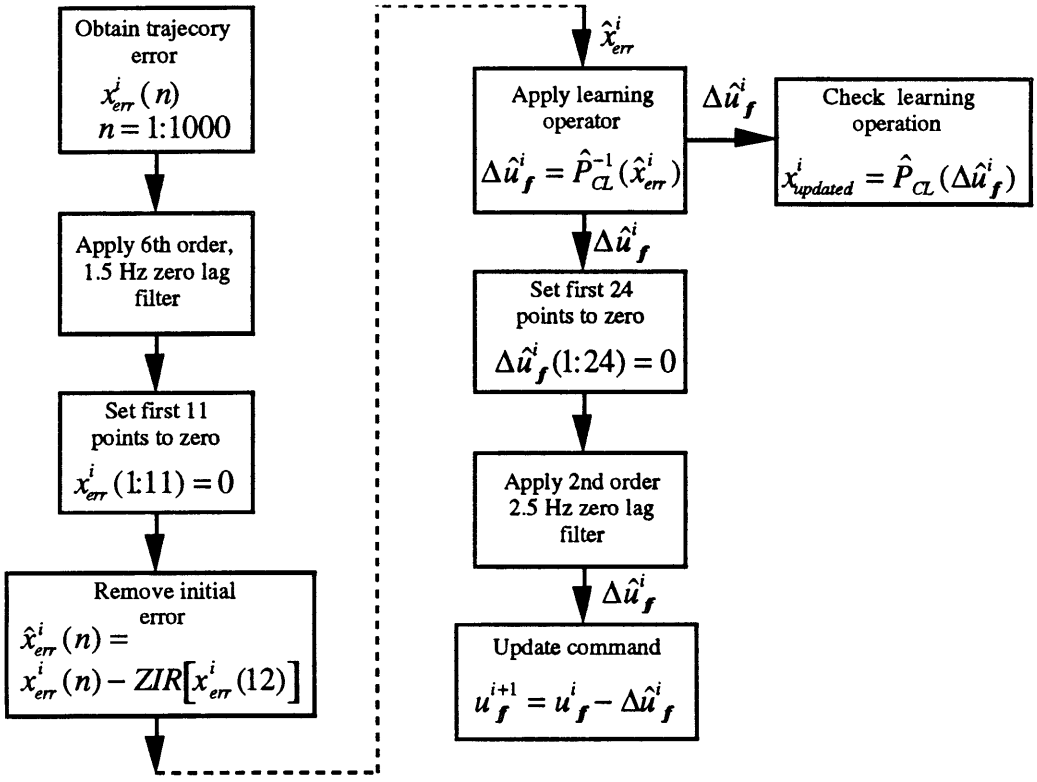

Figure 4-7. Flow Chart of Command Update Procedure

Because the spikes in the first few timesteps of the command update may have saturated the actuators, or caused poor learning performance, the first 24 timesteps of the command were zeroed. A 2nd order, 2.5 Hz, zero lag filter was applied to the update for final smoothing. The actual feedforward command update $\Delta\hat{u}_{ff}^{0}$ is shown in the third plot of figure 4-6. This lengthy process, though not analytically justified, was successfully tested

in simulations as well as the experiments. The flow chart in figure 4-7 re-iterates the steps of the command update process. The actual code used for the update is included in Appendix D for reference.

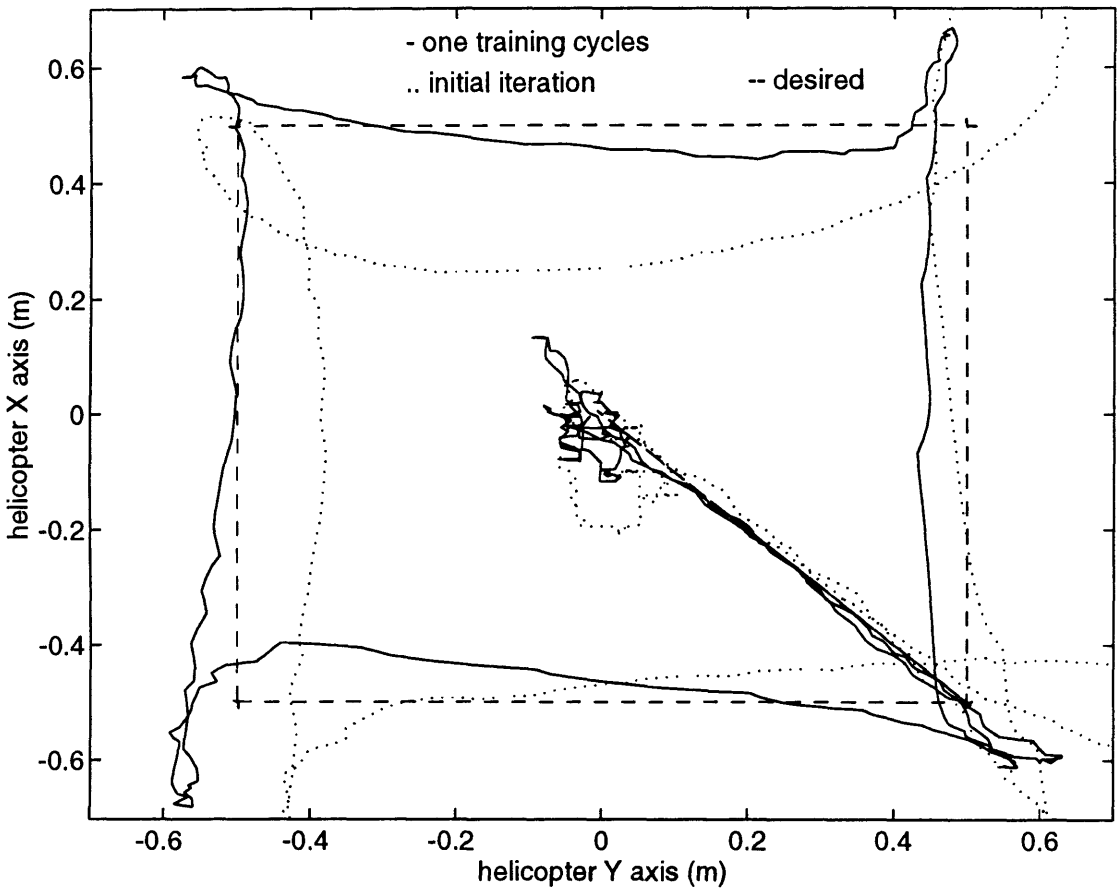

Figure 4-8. Second Trajectory Attempt

The trajectory results of the next flight are shown in figure 4-8, along with the original and desired trajectory. Significant improvement was achieved with only one training run. The errors on this run were rms_x = 0.0656 m., and rms_y = 0.0520 m.

4.4 Successive Trajectory Iterations

The trajectory was repeated for a total of ten training cycles (eleven execution stages). Figure 4-9 plots the progression of the rms error for each of the axis over the iterations.

Typically, improvement settled after the fourth or fifth learning cycle. The flat lines show the hover rms error (average from several flights) for the respective axis. Small variations do occur but, overall, the error appears stabilized at these levels. The plot also shows that performance on any iteration was better than the original execution.

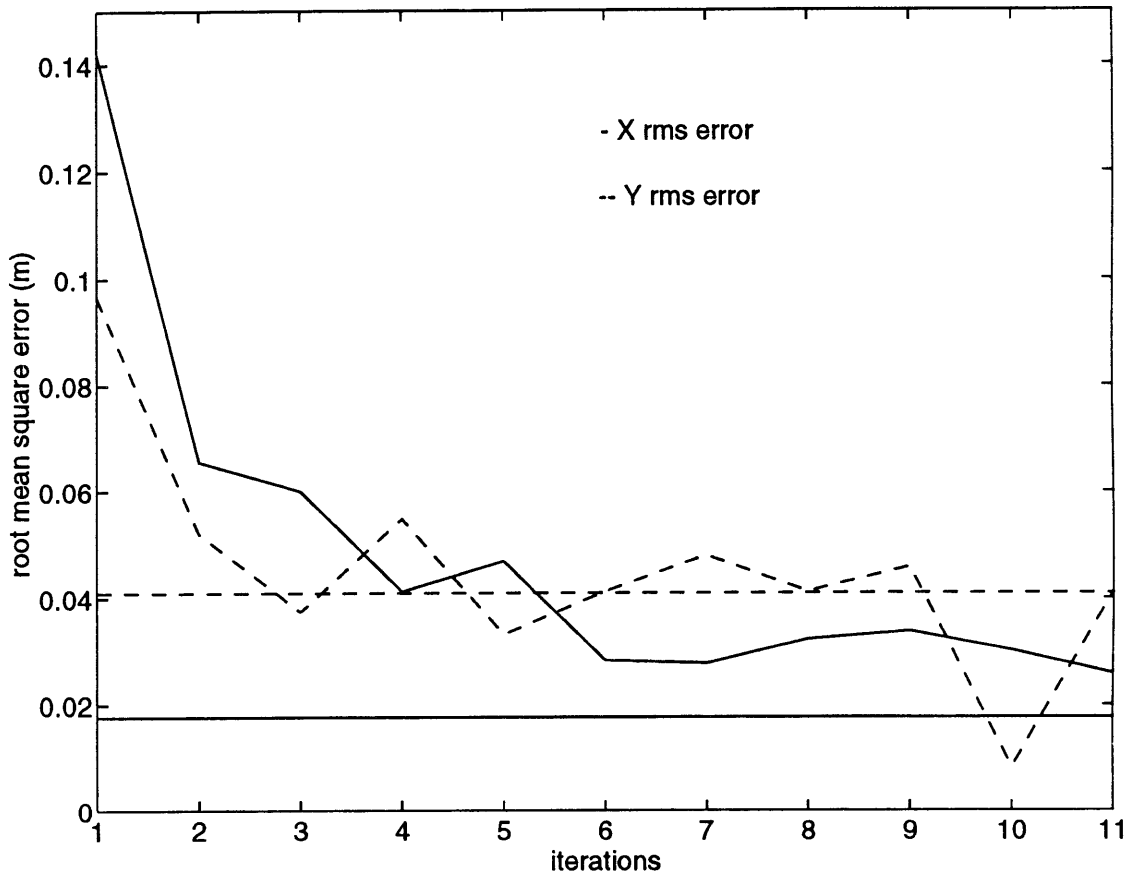

Figure 4-9. Root Mean Square Error vs. Iterations

Figure 4-10 shows a top view of the trajectory after five training cycles. The original trajectory as well as the desired trajectory are plotted for comparison. Figure 4-11 compares the position errors of the individual axis. Again, the original error is shown for comparison.

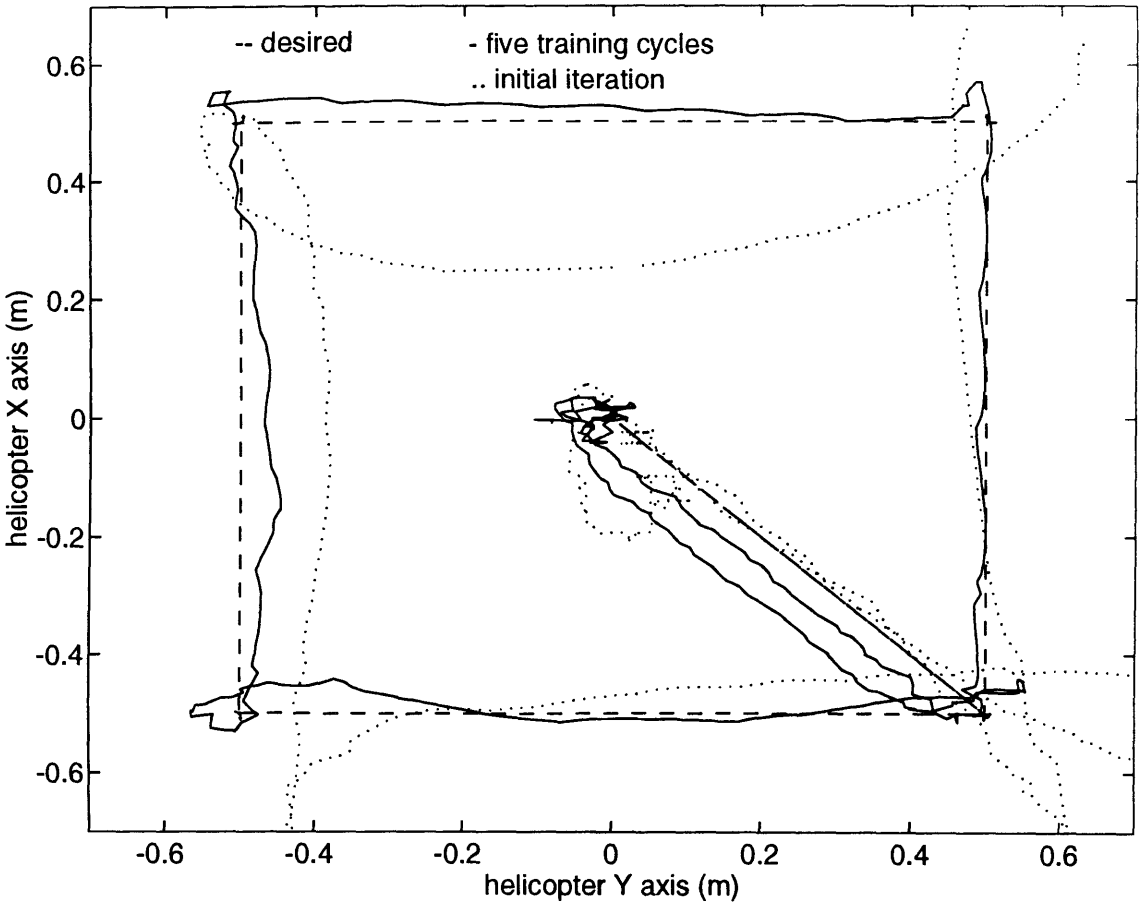

Figure 4-10. Trajectory after Five Training Cycles

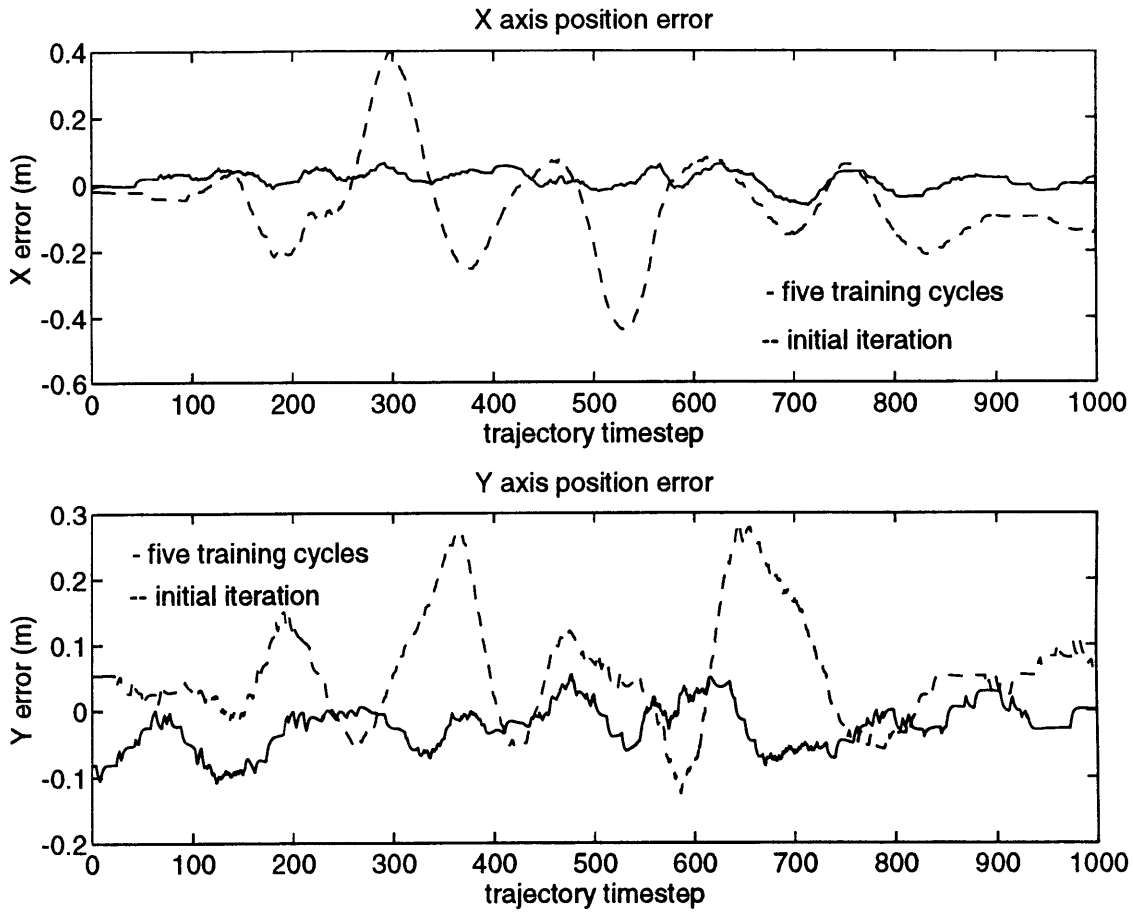

Figure 4-11. Performance on Individual Axis after Five Training Cycles

Figure 4-12 compares the original feedforward command $u_{ff}^{0}(t)$ with the updated feedforward command $u_{ff}^{5}(t)$. This plot shows the large changes that were made to the feedforward command over the learning iterations. The total command $u_{tot}^{5}(t)$ $(=u_{ff}^{5}(t)+u_{fb}^{5}(t))$ applied to the plant is also shown, in comparison with the total command for the original run $u_{tot}^{0}(t)$. In considering the closed loop plant dynamics, the feedback controller was acting on the entire state vector (and not an error from a desired one). Though much of the feedforward command is effectively canceled by the feedback command, distinct differences between the two total commands can still be observed.

By time sample 750 the motion over the square has been completed, so the final seconds of the trajectory place the helicopter in a hover. The feedforward command after the fifth

learning cycle shows a distinct offset over this period of hovering. The offset demonstrates corrections made for mean errors in hover which are caused by instrumentation bias.

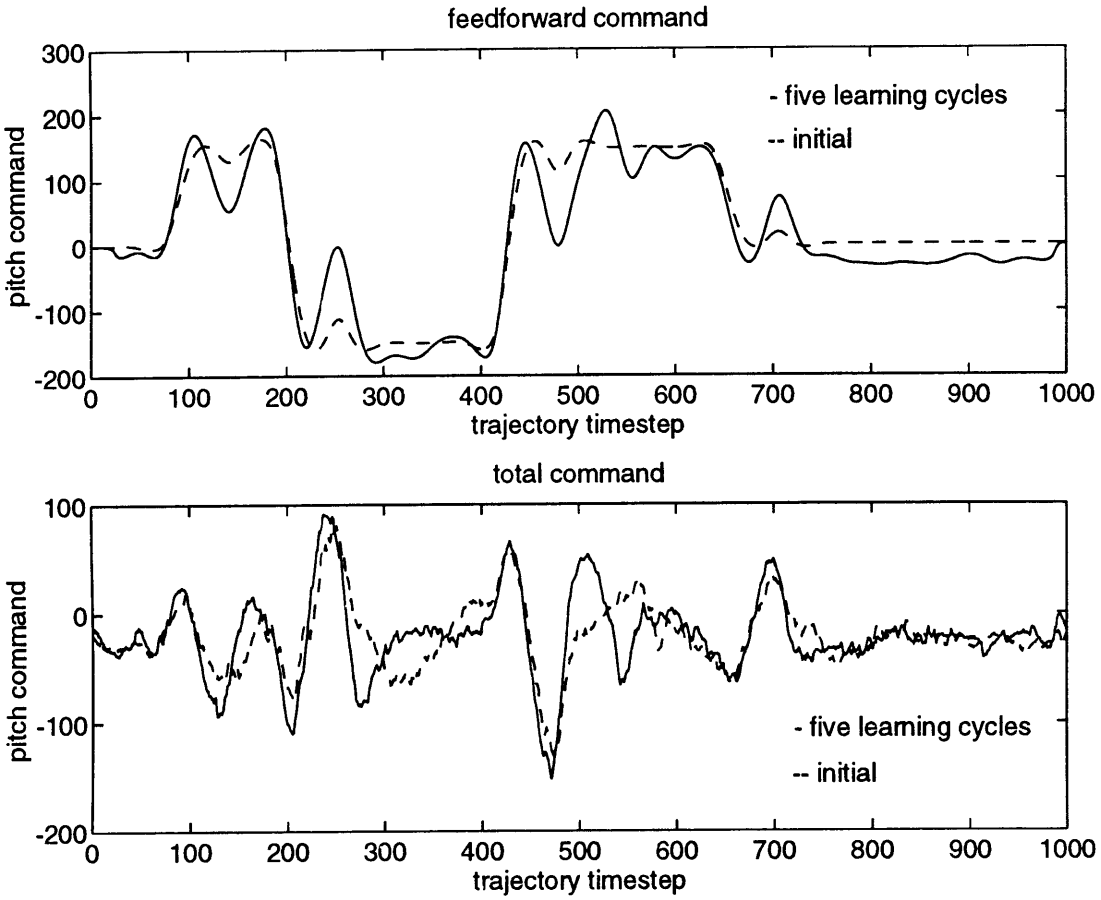

Figure 4-12. Feedforward and Total Commands

4.5 Additional Observations and Experiments

One problem of the trajectory learning algorithm was its tendency to amplify high frequency noise in the feedforward command. Even though performance reached a steady state, the feedforward command continued to grow. Figure 4-13 shows the progression of the root mean squared feedforward command for each of the axes. Due to the relatively quick convergence of the trajectory error, this phenomenon was not

considered a serious problem, as the trajectory learning algorithm was not run beyond 10 iterations. The increase in rms command appears to have had little effect on the position error, which remained stable over the 10 iterations.

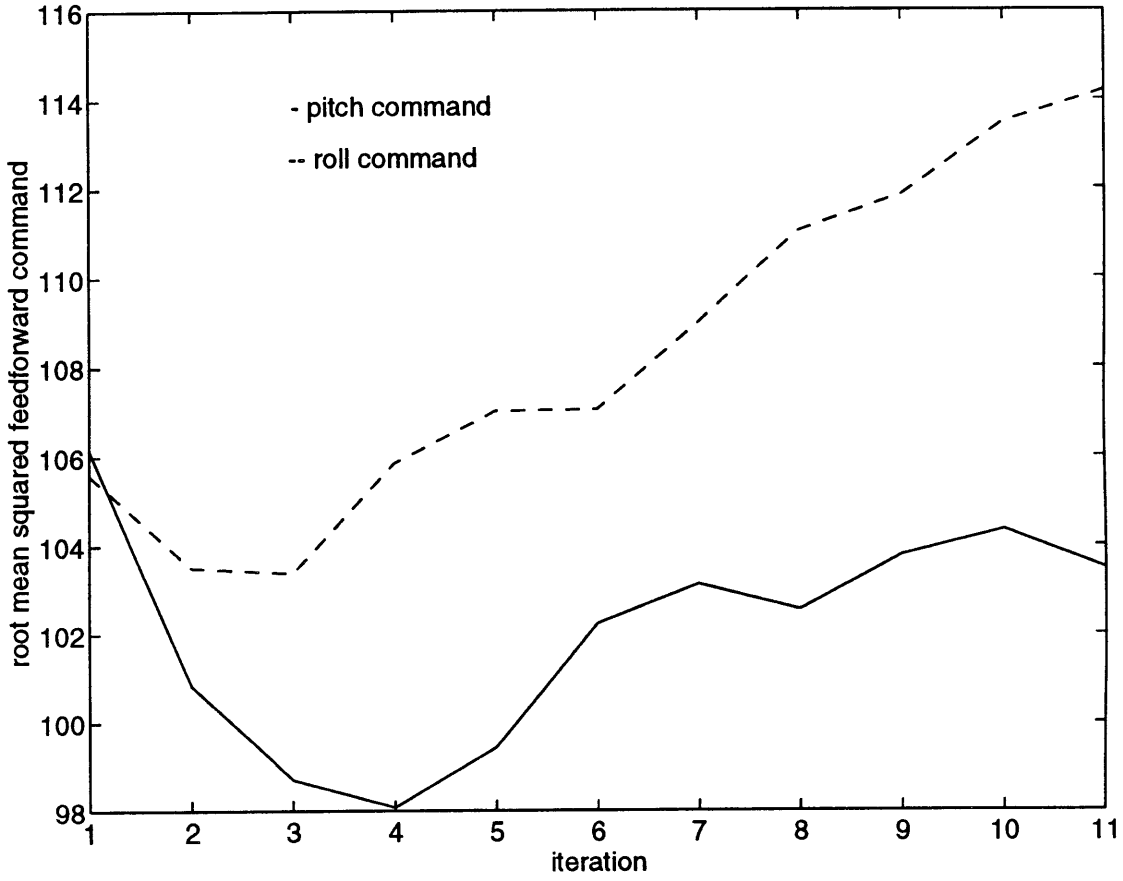

Figure 4-13. Mean Squared Feedforward Command

Figure 4-14 plots the pitch feedforward command after five and ten learning cycles. It appears that the high frequency components of the command are much larger after ten learning cycles. An explanation for this phenomenon could be that higher frequency position errors require many iterations to alter the command. High frequency position errors are slower in affecting the feedforward command since the filtering used in the learning algorithm attenuates these errors. As long as the errors are repeated, however, they will eventually have a significant effect on the feedforward command. Since the effect of modeling errors typically increases with frequency, it is presumed that the

inverse learning operator would not make accurate high frequency corrections to the feedforward command, and further learning iterations would be fruitless.

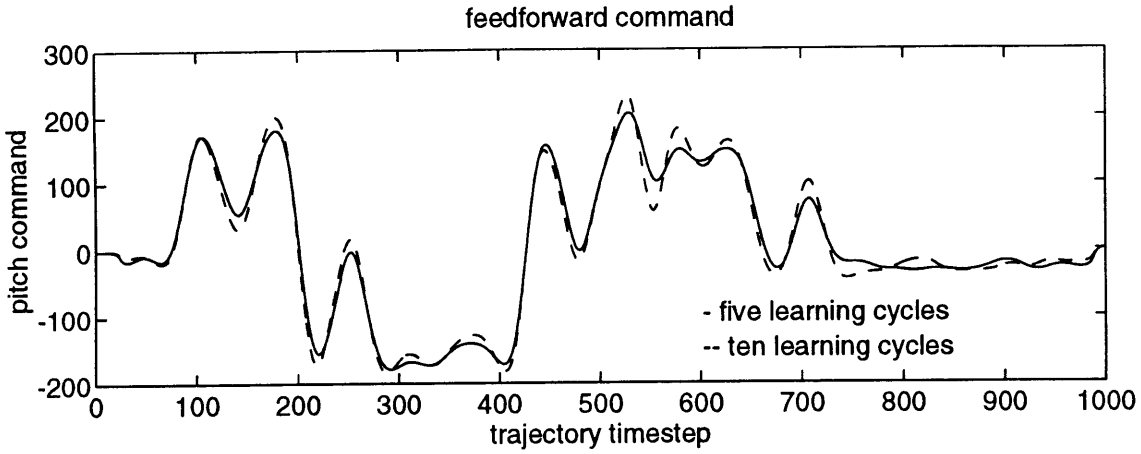

Figure 4-14. Feedforward Command After Five and Ten Learning Cycles

As mentioned earlier in this chapter, these trajectory experiments were conducted on several different x/y planar trajectories. Results of these experiments demonstrated similar performance of the learning algorithm on all trajectories. An additional test that was conducted started the learning iterations without an initial feedforward command (i.e. $u_{ff}^{0}(t) = 0$). Performance did improve to the same level as for the iteration cycles which started with the best initial command estimate, though it naturally required a greater number of iterations (approximately eight).

Chapter 5

Conclusion

The premise investigated by this thesis is that traditional control system performance can be improved by using information from previous experiences. Fixed controllers based on models of plant dynamics are limited by the accuracy of the model. Intelligent control methods can improve performance by either altering the controller or the model after observing previous performance. Work done for this thesis has produced a valuable test platform as well as preliminary results investigating the utility of a trajectory learning algorithm for complex control problems. The trajectory learning algorithm demonstrated dramatic performance improvement through practice on a helicopter testbed.

This thesis has described the helicopter testbed as it was developed for these experiments. The testbed system includes the helicopter chassis, instrumentation, LQG hover controller, and a versatile computation environment to test a variety of control algorithms. The control issues presented by the helicopter's dynamics are representative of a wide range of applications. A majority of research on learning algorithms is restricted to simulations or experiments with a more limited scope. By providing a readily available means of gaining empirical data, the helicopter platform provides a

valuable tool for advancing research in intelligent control. The helicopter testbed is inexpensive to purchase and operate, can be flown indoors, and is highly maneuverable.

The main development of the thesis is the adaptation of an existing trajectory learning algorithm [An, et. al., 1988] to the helicopter testbed. The trajectory learning approach updates a feedforward component of the controller in an iterative process. Several difficulties, which are not normally found in trajectory learning research, were encountered and overcome. A method was developed to handle initial errors through simulation experiments. In actual use it was found that additional processing was required before and after applying the learning operator. The result was a lengthy, yet intuitive strategy for processing the feedforward command update. This was successfully demonstrated in simulations and actual flight experiments. The most important change to the original inverse model approach used by Atkeson (1988) was the consideration of the full closed loop dynamics of the plant. This step was necessary for successful implementation of the algorithm on the helicopter.

Flight experiments demonstrated the ability of the trajectory learning algorithm to improve trajectory following performance after only a few iterations. Initial rms tracking errors for the axes were rms_x = 0.142 m., and rms_y 0.096 m. These errors were reduced to rms_x = 0.0283 m. and rms_y = 0.0412 m., which is comparable to hover rms errors: rms_x = 0.0177 m., and rms_y = 0.0410 m. This thesis asserts that regardless of the accuracy of the plant model, persistent discrepancies in the model will cause errors which can be reduced by the trajectory learning algorithm.

Though the particular implementation used in this thesis relied on the assumptions of a linear plant model and constant gain feedback controller, the trajectory learning algorithm

can be applied much more generally. As long as a full closed loop model is available, and it has a unique inverse, this trajectory learning algorithm may be applied.

There are several practical restrictions which would determine the applicability of a trajectory learning algorithm for a flight vehicle. If the flight profiles of a vehicle involve repeated execution of some maneuvers in similar flight environments, the trajectory learning algorithm can be applied. These maneuvers must have dominant errors which are repeatable. The algorithm also does not allow for any change in command during the execution of the maneuver. In other words, once initiated, the maneuver must be flown through to the end. This is generally not acceptable for a vehicle piloted by a human. At this time, however, the field of unmanned aerial vehicles is quickly expanding. Such vehicles often incorporate semi-autonomous control systems that involve limited human supervision and control. These vehicles often perform preplanned maneuvers. Trajectory learning algorithms might provide these vehicles with improved control system performance, and/or reduced control development costs. It is in these such areas where trajectory learning algorithms could find practical application.

Appendix A

Inverse Plant Operator

As described in section 2.3.1.1, the inversion of the plant dynamics was performed by using a z-transform description of the SISO state space plant model. The matlab code which acts as the inverse plant operator follows. Given the plant dynamics (in a state space description) and an output history, the function produces the plant's corresponding input history.

```
% function [u_update, x_update] = learn5(x, A, B)
% This code calculates the inverse model of the plant given by A and B.
% it assumes the last state as the output state
% the function returns u_update, which is the command that produces x in the
% forward run of the plant
% also provided is x_update, which is the full state history associated with a
% forward run of the plant on u_update

function [u_update, x_update] = learn5(x,A,B)

states = length(A);
C = zeros(1,states);
C(1,states) = 1;
D = [ 0 ];

% obtain z-transform of the plant
[fnum, den] = ss2tf((A-B*kp),B,C,D,1);

% Create the inverse transform u(n-12)/Y to increase the order of the
% numerator so it can become inv_den and still be a proper causal system

% First remove near zero terms from the numerator as these will cause numerical errors
num = 1e-14*round(num*1e14);

inv_num = den;
```

```
inv_den = [ num 0 0 0 0 0 0 0 0 0 0 0 0 0];

% Run the inverse plant
[u_update_delayed,X] = dlsim(inv_num, inv_den, x);

% now fix the time delay, since u_update is actually u_update(n-12):
time = 12:1:1000;
u_update = ([u_update_delayed(time); zeros(11,1)]);

% Check on what u_update actually produces in the system
% x_update is the result of the perfect model running u_update
% x12 is the position (x) output of u_update
[x12, x_update] = dlsim(A,B,C,D,u_update);
```

Appendix B

Creating a Feasible Desired Trajectory

This section describes the process that was used to generate the desired trajectory x_d and initial feedforward command $u_{ff}^0 (= u_{ff}^*)$. The method used is based on a regression which can be described as the solution to an optimal control problem. The regression creates a desired position trajectory x_d which approximates a given goal trajectory x_g. While the goal trajectory can be arbitrarily chosen, the regression takes into account plant model dynamics such that the desired trajectory is feasible. The regression minimizes a functional with quadratic costs on control and the error between the desired and goal trajectories $\left(x_g(t) - x_d(t)\right)$.

$$E = \sum_{n=0}^{N} \left[\left(x_g(n) - x_d(n) \right)^2 + w^2 u_{tot}^0(n)^2 \right] \qquad \text{(B-1)}$$

The summation in equation (B-1) is taken over all 1000 points of the trajectory. The regression procedure is applied to the roll and pitch axis independently.

The regression is formulated as the solution to a system of linear equations. The equations map the controls u_n into the position states x_n over all points in the trajectory. By considering the deterministic plant model the position history is uniquely determined

by the control history. The mapping can be constructed by looking at the discrete time state space equations for the plant model:

$$\begin{aligned} X_{n+1} &= AX_n + Bu_n \\ y_n &= CX_n \end{aligned} \tag{B-2}$$

The output of the system y is the position state x Once again, in the case of a non-linear plant model, this procedure may still be applied by using a local linear state space model in place of the fixed matrices in (B-2).

The state equations can be propagated to show how the position at a particular timestep is related to the control applied at all previous timesteps. Assuming that the initial state X_0 is zero, at the next timestep, the position x_1 is given by:

$$\begin{aligned} x_1 &= C(AX_0 + Bu_0) \\ &= CBu_0 \end{aligned} \tag{B-3}$$

At the second timestep, the position is affected by all previous controls u_0, u_1 as shown by equation (B-4).

$$\begin{aligned} x_2 &= C(AX_1 + Bu_1) \\ &= C(A(AX_0 + Bu_0) + Bu_1 \\ &= CABu_0 + CBu_1 \end{aligned} \tag{B-4}$$

In this manner, a system of equations (N=1000) can be constructed to relate the set of controls to positions. For the entire trajectory, all of the controls and states for every timestep can be assembled into two vectors $\underline{u}$ and $\underline{x}$ respectively.

$$\underline{u} = \begin{bmatrix} u_0 \\ : \\ u_N \end{bmatrix}, \underline{x} = \begin{bmatrix} x_0 \\ : \\ x_1 \end{bmatrix} \tag{B-5}$$

The entire system of equations for the trajectory can be described as a vector-matrix linear equation:

$$A\underline{u} = \underline{x}$$

$$A = \begin{bmatrix} 0 & 0 & : & : & 0 \\ CB & 0 & : & : & 0 \\ CAB & CB & 0 & : & : \\ : & : & : & : & : \\ CA^{N-2}B & : & : & CB & 0 \end{bmatrix} \qquad \text{(B-6)}$$

The form of this equation is well known and since the system is square an exact solution for $\underline{u}$ can be obtained for any $\underline{x}$ that is provided, as long as the singularity of A can be handled. In a discrete approximation, every position trajectory $\underline{x}$ is feasible, but it may require excessive control actions. In order to incorporate the cost on the control vector, equation (B-6) must be modified.

If equation (B-6) is modified so that the system is overdetermined, than the least squares solution to the new system (B-7) will be the vector $\underline{u}$ which minimizes a quadratic function (B-8) [Strang 1986].

$$\hat{A}\underline{u} = \hat{\underline{x}}$$
$$\underline{u} = \left(\hat{A}^T\hat{A}\right)^{-1}\hat{A}^T\hat{\underline{x}} \qquad \text{(B-7)}$$

$$(\hat{A}\underline{u} - \hat{\underline{x}})^T(\hat{A}\underline{u} - \hat{\underline{x}}) \qquad \text{(B-8)}$$

The matrix $\hat{A}$ and vector $\hat{\underline{x}}$ can be constructed so that the minimization of (B-8) is equivalent to minimizing the cost function. The modified system matrix $\hat{A}$ and output vector $\hat{\underline{x}}$ are given by:

$$\hat{A} = \begin{bmatrix} A \\ wI \end{bmatrix}, \hat{\underline{x}} = \begin{bmatrix} \underline{x} \\ \underline{0} \end{bmatrix} \qquad \text{(B-9)}$$

The zero vector $\underline{0}$ and identity matrix I have dimensions of 1000x1 and 1000x1000 respectively. They have the effect of penalizing the control at each timestep by trying to fit $\underline{u}$ to the zero vector.

Given the goal trajectory history $\underline{\hat{x}_g}$ the least squares solution to the system (B-10) will provide the optimal control history $\underline{u}$ which minimize the cost function (B-1) for the dynamic system (B-2). The actual output of the plant model becomes the desired trajectory $\underline{x_d}$, found by applying the system matrix to the control history (B-6). Expanding (B-8) verifies that the least squares solution does in fact minimize the cost function (B-1).

$$\begin{aligned}&(\hat{A}\underline{u}-\underline{\hat{x}_g})^T(\hat{A}\underline{u}-\underline{\hat{x}_g})\\&=\underline{u}^T\hat{A}^T\hat{A}\underline{u}-\underline{u}^T\hat{A}^T\underline{\hat{x}_g}-\underline{\hat{x}_g}^T\hat{A}\underline{u}+\underline{u}^T\underline{u}\end{aligned} \qquad \text{(B-11)}$$

By substituting (B-9) and (B-6) and multiplying, we have:

$$\begin{aligned}&=\underline{x_d}^T\underline{x_d}+w^2\underline{u}^T\underline{u}-\underline{x_d}^T\underline{\hat{x}_g}-\underline{\hat{x}_g}^T\underline{x_d}+\underline{\hat{x}_g}^T\underline{\hat{x}_g}\\&=\left(\underline{x_d}-\underline{\hat{x}_g}\right)^T\left(\underline{x_d}-\underline{\hat{x}_g}\right)+w^2\underline{u}^T\underline{u}\end{aligned} \qquad \text{(B-12)}$$

This is equivalent to the original cost function (B-1).

This method provides a direct method of solving the optimization problem as long as the matrix $\hat{A}$ is well conditioned. The least squares solution provides the initial total command history $\underline{u^0_{tot}}=\underline{u}$ and desired trajectory $\underline{x_d}$ (from (B-6)), but the feedforward command must still be determined. The feedforward command can be found by applying the inverse plant model to the desired trajectory as described in section 2.3.1.1.

The matlab code which performs the regression follows. The code presumes that the state space model has been stored in file m9_roll.mat and that the desired trajectory vector exists as Xg.

```
% Least Squares Optimization Code
load m9_roll

% N = number of time samples in the trajectory
N = 1000;

% Create the A_ls matrix according to the plant dynamics
A_ls = zeros(2*N,N);
for i = 2:N
 i
 A_ls(i,1) = C*A^(i-2)*B;
 for j = 2:N
  A_ls(i,j) = A_ls(i-1,j-1);
 end
end

% Now add the control weight
w = 5e-4
for i = 1:N
 A_ls(i+N,i) = w;
end

% Create the goal trajectory vector
B_ls = zeros(2*N,1);

for i = 1:N;
 B_ls(i) = Xg(i);
 B_ls(N+i) = 0.0;
end

u_tot = inv(A_ls'*A_ls)*A_ls'*B_ls;
X_des = A_ls(1:1000,:)*u_tot;
```

Appendix C

Code to Create Goal Trajectory

The following code creates the square goal trajectory. The user can select parameters such as the duration of the movement between corners, the size of the square, the hovering pause at each corner, and the time before the trajectory begins. The motions are based on a cosine shape. The final goal trajectories for the X and Y axis are X_goal and Y_goal. Note that no information about the plant dynamics are accounted for. After the goal trajectory is create, the regression described in Appendix B turns this into a desired trajectory.

```
% Square.m - code which creates a square goal trajectory
% set N to total number of samples
N = 1000;

% set lead to # of samples before manuever begin
lead = 100;

% fly a planar square (x-y)
% SIMPLY type in the # of samples that each side takes (duration - 50 samples/sec).
% also type in 'wait' which is the pause at each corner of the square
% To get started, it moves the heli diagonally from the center to the
% rear right corner of the suare.
% the reverse is performed for the finish.

% this traj will not necessarily have continuous derivatives (much less
% cont 2nd, 3rd, etc. deriv's)
% The regression will take care of this

% duration is that of each side motion
duration = 63;
```

```
% wait is the pause at each corner
wait = 50;

% width is the length of one side of the squarein meters
width = 1.0;

% find each corners' coordinates
% number CCW from right rear
x1 = -width/2;
y1 = width/2;
x2 = width/2;
y2 = width/2;
x3 = width/2;
y3 = -width/2;
x4 = -width/2;
y4 = -width/2;

% move out from center
mt = 1:1:duration;
move = (-cos(pi*mt/duration) + 1)/2;
move = move';

pause = ones(wait,1);

X = [   x1*move ; x1*pause
        ((x2-x1)*move + x1) ; x2*pause
        ((x3-x2)*move + x2) ; x3*pause
        ((x4-x3)*move + x3) ; x4*pause
        ((x1-x4)*move + x4) ; x1*pause
        ((-x1)*move + x1) ];

Y = [   y1*move ; y1*pause
        ((y2-y1)*move + y1) ; y2*pause
        ((y3-y2)*move + y2) ; y3*pause
        ((y4-y3)*move + y3) ; y4*pause
        ((y1-y4)*move + y4) ; y1*pause
        ((-y1)*move + y1) ];

X_goal = [zeros(lead,1); X; zeros((N-lead-length(X)),1)];
Y_goal = [zeros(lead,1); Y; zeros((N-lead-length(Y)),1)];

% Plot to see what we got
figure(1)
clg
hold off

subplot(311)
plot(Y,X)
axis([ -width width -width width ])
title('top view, what it should do')
xlabel('aircraft y axis')
ylabel('aircraft x axis')
subplot(312)
plot(X_goal)
title('x')
subplot(313)
plot(Y_goal)
title('y')
```

Appendix D

Trajectory Learning Code

The trajectory learning code, rp_learn2.m, is simply the matlab script for the procedure described in the flow chart in figure 4-6. The code calculates and stores the updated feedforward command for the roll and pitch axis for the next trajectory run. This code loads in the data using 'load_dat' which properly reads in and names all of the data vectors from a specified file. The position data is filtered and processed, before applying the learning operator. Plots illustrate performance of the algorithm. All operations are performed on each axis individually.

```
% rp_learn2.m - matlab script which performs the command update step in
% trajectory learning
% Load recent trajectory data
load_dat

% Set up for processing - load in pitch/roll model and feedback gains
load m12_pitch
load p_gain
Ap = A;
Bp = B;
kp = k;

load m12_roll
load r_gain
Ar = A;
Br = B;
%Ar(10,10) = 1.0;
kr = k;
```

```
Y_des_update = Y_des;
Y_err = Y_est - Y_des;
Y_err2 = Y_err;

X_des_update = X_des;
X_err = X_est - X_des;
X_err2 = X_err;

% filtering first
[b, a] = butter(3,1.00/25);
Y_err2(:,12) = filtfilt(b, a, Y_err(:,12));
X_err2(:,12) = filtfilt(b, a, X_err(:,12));


% ***** GET UPDATE's  ************
Y_err2(1:11,:) = zeros(11,12);
X_err2(1:11,:) = zeros(11,12);

% Initial error handling
u = zeros(1000,1);
[Y_init] = plant_cl(Ar,Br,u,Y_err2(12,:)', kr);
[X_init] = plant_cl(Ap,Bp,u,X_err2(12,:)', kp);

for i = 12:1000
  Y_err2(i,:) = Y_err2(i,:) - Y_init(i-11,:);
  X_err2(i,:) = X_err2(i,:) - X_init(i-11,:);
end

Y12_errf2 = Y_err2(:,12);
X12_errf2 = X_err2(:,12);

[ru_junk, Y_junk] = learn5(Y12_errf2, (Ar - Br*kr), Br);
[pu_junk, X_junk] = learn5(X12_errf2, (Ap - Bp*kp), Bp);

raw_pu_junk = pu_junk;
% ****** NOW these updates are likely to have a sharp jump in the beginning
% ***** just zeros the first few, and then do filtering (zero lag) again over
% each state history to make it all smooth.

ru_junk(1:24) = zeros(24,1);
Y_junk(1:24,:) = zeros(24,12);

pu_junk(1:24) = zeros(24,1);
X_junk(1:24,:) = zeros(24,12);

[b2, a2] = butter(1,2.5/25);
ru_update = filtfilt(b2, a2, ru_junk);
pu_update = filtfilt(b2, a2, pu_junk);

% **** COMPUTE THE UPDATED COMMANDS  ***************
% learning rate.  max = 1;
rate = 0.5;

% *************  Update the commands  ***********
r_ff_com = roll_ff_com - rate * ru_update;
p_ff_com = pitch_ff_com - rate * pu_update;

figure(1)
```

```
clg
hold off
subplot(211)
plot(Y_junk(:,12),'--')
hold
plot(Y_err(:,12))
title( 'Y_err vs Y_updated12--' )

subplot(212)
plot(r_ff_com,'--')
hold
plot(roll_ff_com)
title( 'roll_ff_com vs new roll_ff_com --' )

%**************** Pitch plotting
figure(3)
clg
hold off
subplot(211)
plot(X_junk(:,12),'--')
hold
plot(X_err(:,12))
title( 'X_err vs X_updated12--' )

subplot(212)
plot(p_ff_com,'--')
hold
plot(pitch_ff_com)
title( 'pitch_ff_com vs new pitch_ff_com --' )

% Mean Square Errors
Y_mse = mean(Y_err(:,12).^2)
roll_ff_mse = mean(roll_ff_com.^2)
X_mse = mean(X_err(:,12).^2)
pitch_ff_mse = mean(pitch_ff_com.^2)

figure(2)
clg
hold off
plot(Y_des(:,12),X_des(:,12),'--')
hold
plot(Y_est(:,12),X_est(:,12))
axis([-0.701 0.701 -0.701 0.701])
xlabel('helicopter Y axis (m)')
ylabel('helicopter X axis (m)')

[rep] = input('Would you like to save these desireds? ','s');
if rep == 'yes'
 save roll_des Y_des r_ff_com -ascii
 save pitch_des X_des p_ff_com -ascii
end
```

Appendix E

State Space Models

The discrete time state space models for the roll and pitch axis are provided here. The linear models have a matrix form which corresponds to the format [Astrom and Wittenmark, 1984]

$$X_{n+1} = AX_n + Bu_n$$
$$Y_n = CX_n \qquad \text{(E-1)}$$

The feedback commands are generated using full state linear feedback gains given by

$$u_{fb_n} = -KX_n \qquad \text{(E-2)}$$

/************ Pitch Model ************/

A =	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	1	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	0	0	0	0	0
	0	0	0	0	1	0	0	0	0	0	0	0
	0	0	0	0	0	1	0	0	0	0	0	0
	0	0	0	0	0	-4.2e-1	2.1e-1	1.84	-9.1e-1	0	0	0
	0	0	0	0	0	0	0	1	0	0	0	0
	0	0	0	0	0	0	0	-5.2e-5	0	9.9e-1	0	0
	0	0	0	0	0	0	0	0	0	-2.8e-1	9.9e-1	0
	0	0	0	0	0	0	0	0	0	0	2.0e-2	1

B =
1
0
0
0
0
0
0
0
0
0
0
0

C = 0 0 0 0 0 0 0 0 0 0 0 1

K =
1.1e-1 1.2e-1 1.3e-1 1.3e-1 1.4e-1 1.4e-1 -1.4e-1 -7.0e-1 6.0e-1 1.0e3 -1.9e2 -3.0e2

/********** Roll Model **********/

A =

0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1.26	-7.0e-1	1.67	-8.7e-1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	6.0e-5	0	9.9e-1	0	0
0	0	0	0	0	0	0	0	0	3.1e-1	9.9e-1	0
0	0	0	0	0	0	0	0	0	0	2.0e-2	1

B =
1
0
0
0
0
0
0
0
0
0
0
0

C = 0 0 0 0 0 0 0 0 0 0 0 1

K =
1.2e-1 1.3e-1 1.3e-1 1.4e-1 1.5e-1 1.6e-1 -1.9e-1 2.8e-1 -2.3e-1 1.0e3 1.9e2 3.0e2

References

Ahmad, S., T. J. Pallet, "Real-Time Helicopter Flight Control: Modeling and Control by Linearization and Neural Networks", Purdue University Report TR-EE 91-35, August, 1991.

Ahn, Hyun-Sik, Chong-Ho Choi, Tae-Jeong Jang., "Iterative Learning Control in Feedback Systems", Submitted to Automatica, Dec. 1992.

An, C. H., C. G. Atkeson, J. M. Hollerbach, *Model-Based Control of a Robot Manipulator,* (MIT Press, Cambridge, MA, 1988).

Aoyama, T., Y. Konishi, I. Inasaki, "Iterative Learning Control of a Parallel Link Direct-Drive Robot Manipulator", Robotics and Autonomous Systems 5, pp. 127-134, 1989.

Arimoto, S., S. Kawamura, F. Miyazaki, "Bettering Operation of Robots by Learning", Journal of Robotic Systems, 1-2, pp. 123-140, 1984.

Arimoto, S., "Passivity of Robot Dynamics Implies Capability of Motor Program Learning", Advanced Robot Control: Proceedings of the International Workshop on Nonlinear and Adaptive Control: Issues in Robotics, (Grenoble, France, Nov 21-23, 1990).

Ashkenas, I., D. Graham, D. McRuer, *Aircraft Dynamics and Automatic Control*, (Princeton University Press, Princeton, N.J., 1973).

Astrom, K.J., B. Wittenmark, *Computer Controlled Systems*, (Prentice Hall, Englewood Cliffs, N.J., 1984).

Atkeson, C. G., J. McIntyre, "An Application of Adaptive Feedforward Control to Robotics", MIT Artificial Intelligence Laboratory, (Internal Paper, Cambridge, MA, 1987).

Boals, M., W. Kao, R. Horowitz, M. Tomizuka, "Repetitive Control of a Two Degree of Freedom SCARA Manipulator", Proceedings of the Automatic Control Conference, pp. 1484-1490, (June, 1989).

Cauffman, M. G., M. B. Tischler, "Frequency-Response Methods for Rotorcraft System Identification with Applications to the BO-105 Helicopter", Proceedings of the 46th American Helicopter Society Forum, (Washington, D.C., May 1990).

Crawley, E. F., S. R. Hall, *System Identification* , Controlled Structures Short Course, (Summer Course at MIT, 1991).

Department of Defense, "Unmanned Aerial Vehicle (UAV) Master Plan", UAV Joint Project Office Report, (31 March, 1993).

Foxlin, Eric, "Inertial Head Tracking", M. S. Thesis, Electrical Engineering, Massachusetts Institute of Technology, (August 31, 1993).

Franklin, G. F., J. D. Powell, M. L. Workman, *Digital Control of Dynamic Systems*, (Addison-Wesley, New York, N.Y., 1990).

Gomi, H., M. Kawato, "Neural-Network Control for a Closed-Loop System using Feedback-Error-Learning", Advanced Telecommunications Research Institute International Technical Report, (Kyoto, Japan, 1991).

Guglielmo, K., N. Sadegh, "A New Learning Controller for Mechanical Manipulators", Proceedings of the Automatic Control Conference, pp. 2827-2833, (June 1989).

Kawato, M., H. Miyamoto, T. Setoyama, R. Suzuki, "Feedback-Error-Learning Neural Network for Trajectory Control of a Robotic Manipulator", Neural Networks, Vol. 1, pp. 251-265, (1988).

Kirk, D. E., *Optimal Control Theory*, (Prentice Hall, Englewood Cliffs, N.J., 1970).

Johnson, Wayne, *Helicopter Theory*, (Princeton University Press, Princeton, N.J., 1980).

Moore, K. L., M. A. Waddoups, "Neural Networks for Iterative Learning Control", Proceedings of the American Control Conference, pp. 3049-3051, (1992).

Moore, K. L., *Iterative Learning Control for Deterministic Systems*, (Springer-Verlag, Berlin, Germany, 1993).

Strang, G., *Introduction to Applied Mathematics*, (Wellesley-Cambridge Press, Wellesley, MA., 1986).

Siebert, W. M., *Circuits, Signals, and Systems*, (MIT Press, Cambridge, MA., 1986).

Sofge, D. A., D. A. White, *Handbook of Intelligent Control*, (Van Nostrand Reinhold, New York, N.Y., 1992).

Tradelius, Paul, *Basics of Radio Control Helicopters*, (Air Age, Wilton, CT, 1988).

Uchiyama, M., "Formation of High-Speed Motion Pattern of a Mechanical Arm by Trials", Trans. of Society of Instrument and Control Engineers (Japan). 19(5): pp. 706-712 (1978).

Wright, Anne, " A High-Speed Low-Latency Portable Visual Sensing System", SPIE Proceedings, (1993).

3524-21