DOI:10.1145/1538788.1538812

Apprenticeship Learning for Helicopter Control

By Adam Coates, Pieter Abbeel, and Andrew Y. Ng

Abstract

Autonomous helicopter flight is widely regarded to be a highly challenging control problem. As helicopters are highly unstable and exhibit complicated dynamical behavior, it is particularly difficult to design controllers that achieve high performance over a broad flight regime.

While these aircraft are notoriously difficult to control, there are expert human pilots who are nonetheless capable of demonstrating a wide variety of maneuvers, including aerobatic maneuvers at the edge of the helicopter's performance envelope. In this paper, we present algorithms for modeling and control that leverage these demonstrations to build high-performance control systems for autonomous helicopters. More specifically, we detail our experiences with the Stanford Autonomous Helicopter, which is now capable of extreme aerobatic flight meeting or exceeding the performance of our own expert pilot.

1. INTRODUCTION

Autonomous helicopter flight represents a challenging control problem with high-dimensional, asymmetric, noisy, nonlinear, nonminimum phase dynamics. Helicopters are widely regarded to be significantly harder to control than fixed-wing aircraft. (See, e.g., Leishman,[18] Seddon.[31]) At the same time, helicopters provide unique capabilities, such as in-place hover and low-speed flight, important for many applications. The control of autonomous helicopters thus provides a challenging and important test bed for learning and control algorithms.

There is a considerable body of research concerning control of autonomous (RC) helicopters in the typical "upright flight regime." This has allowed autonomous helicopters to reliably perform many practical maneuvers, such as sustained hover, low-speed horizontal flight, and autonomous landing.[9, 16, 17, 24, 28, 30]

In contrast, autonomous flight achievements in other flight regimes have been limited. Gavrilets et al.[14] performed some of the first autonomous aerobatic maneuvers: a stall-turn, a split-S, and an axial roll. Ng et al.[23] achieved sustained autonomous inverted hover. While these results significantly expanded the potential capabilities of autonomous helicopters, it has remained difficult to design control systems capable of performing arbitrary aerobatic maneuvers at a performance level comparable to human experts.

In this paper, we describe our line of autonomous helicopter research. Our work covers a broad approach to autonomous helicopter control based on "apprenticeship learning" that achieves expert-level performance on a vast array of maneuvers, including extreme aerobatics and autonomous autorotation landings.[1, 2, 12, 23] (Refer footnote a.)

In apprenticeship learning, we assume that an expert is available who is capable of performing the desired maneuvers. We then leverage these demonstrations to learn all of the necessary components for our control system. In particular, the demonstrations allow us to learn a model of the helicopter dynamics, as well as appropriate choices of target trajectories and reward parameters for input into a reinforcement learning or optimal control algorithm.

The remainder of this paper is organized as follows: Section 2 briefly overviews related work in the robotics literature that is similar in spirit to our approach. Section 3 describes our basic modeling approach, where we develop a model of the helicopter dynamics from data collected under human control, and subsequently improve this model using data from autonomous flights. Section 4 presents an apprenticeship-based trajectory learning algorithm that learns idealized trajectories of the maneuvers we wish to fly. This algorithm also provides a mechanism for improving our model of the helicopter dynamics along the desired trajectory. Section 5 describes our control algorithm, which is based on differential dynamic programming (DDP).[15] Section 6 describes our helicopter platform and presents our experimental results.

2. RELATED WORK

Although no prior works span our entire setting of apprenticeship learning for control, there are separate pieces of work that relate to various components of our approach.

Atkeson and Schaal,[8] for instance, use multiple demonstrations to learn a model for a robot arm, and then find an optimal controller in their simulator, initializing their optimal control algorithm with one of the demonstrations.

The work of Calinon et al.[11] considered learning trajectories and constraints from demonstrations for robotic tasks. There, however, they do not consider the system's dynamics or provide a clear mechanism for the inclusion of prior knowledge, which will be a key component of our approach as detailed in Section 4. Our formulation will present a principled, joint optimization which takes into account the multiple demonstrations, as well as the (complex) system dynamics.

Among others, An et al.[6] and Abbeel et al.[5] have exploited the idea of trajectory-specific model learning for control.

[a] Autorotation is an emergency maneuver that allows a trained pilot to descend and land the helicopter without engine power.

A previous version of this paper, entitled "Learning for Control from Multiple Demonstrations" was published in *Proceedings of the 26th International Conference of Machine Learning*, (ICML 2008), 144–151.

In contrast to our setting, though, their algorithms do not coherently integrate data from multiple (suboptimal) demonstrations by experts. We will nonetheless use similar ideas in our trajectory learning algorithm.

Our work also has strong connections with recent work on inverse reinforcement learning, which extracts a reward function from expert demonstrations. See, e.g., Abbeel,[4] Neu,[22] Ng, Ramachandran, Ratliff,[25–27] Syed.[32] We will describe a methodology roughly corresponding to the inverse RL algorithm of Abbeel[4] to tune reward weights in Section 5.2.

3. MODELING

The helicopter state s comprises its position (x, y, z), orientation (expressed as a unit quaternion q), velocity $(\dot{x}, \dot{y}, \dot{z})$, and angular velocity $(\omega_x, \omega_y, \omega_z)$. The pitch angle of a blade is changed by rotating it around its long axis changing the amount of thrust the blade generates. The helicopter is controlled via a four-dimensional action space:

1. u_1 and u_2: The lateral (left–right) and longitudinal (front–back) cyclic pitch controls cause the helicopter to roll left or right, and pitch forward or backward, respectively.
2. u_3: The tail rotor pitch control changes tail rotor thrust, controlling the rotation of the helicopter about its vertical axis.
3. u_4: The main rotor collective pitch control changes the pitch angle of the main rotor's blades, by rotating the blades around an axis that runs along the length of the blade. The resulting amount of upward thrust (generally) increases with this pitch angle; thus this control affects the main rotor's thrust.

By using the cyclic pitch and tail rotor controls, the pilot can rotate the helicopter into any orientation. This allows the pilot to direct the thrust of the main rotor in any particular direction (and thus fly in any particular direction) by rotating the helicopter appropriately.

Following our approach from Abbeel,[3] we learn a model from flight data that predicts accelerations as a function of the current state and inputs. Accelerations are then integrated to obtain the state changes over time. To take advantage of symmetry of the helicopter, we predict linear and angular accelerations in a "body-coordinate frame" (a coordinate frame attached to the helicopter). In this body-coordinate frame, the x-axis always points forward, the y-axis always points to the right, and z-axis always points down with respect to the helicopter.

In particular, we use the following model:

$$\begin{aligned}
\ddot{x}^b &= A_x\dot{x}^b + g_x^b + w_x,\\
\ddot{y}^b &= A_y\dot{y}^b + g_y^b + D_0 + w_y,\\
\ddot{x}^b &= A_z\dot{z}^b + g_z^b + C_4u_4 + D_4 + w_z,\\
\dot{\omega}_x^b &= B_x\omega_x^b + C_1u_1 + D_1 + w_{\omega_x},\\
\dot{\omega}_y^b &= B_y\omega_y^b + C_2u_2 + D_2 + w_{\omega_y},\\
\dot{\omega}_z^b &= B_z\omega_z^b + C_3u_3 + D_3 + w_{\omega_z}.
\end{aligned}$$

By our convention, the superscripts b indicate that we are using body coordinates. We note our model explicitly encodes the dependence on the gravity vector (g_x^b, g_y^b, g_z^b) and has a sparse dependence of the accelerations on the current velocities, angular rates, and inputs. The terms w_x, w_y, w_z, w_{ω_x}, w_{ω_y}, and w_{ω_z} are zero mean Gaussian random variables, which represent the perturbation of the accelerations due to noise (or unmodeled effects).

To learn the coefficients, we record data while the helicopter is being flown by our expert pilot. We typically ask our pilot to fly the helicopter through the flight regimes we would like to model. For instance, to build a model for hovering, the pilot places the helicopter in a stable hover and sweeps the control sticks back and forth at varying frequencies to demonstrate the response of the helicopter to different inputs while hovering. Once we have collected this data, the coefficients (e.g., A_x, B_x, C_1, etc.) are estimated using linear regression.

When we want to perform a new maneuver, we can collect data from the flight regimes specific to this maneuver and build a new model. For aerobatic maneuvers, this involves having our pilot repeatedly demonstrate the desired maneuver.

It turns out that, in practice, these models generalize reasonably well and can be used as a "crude" starting point for performing aerobatic maneuvers. In previous work,[2] we demonstrated that models of the above form are sufficient for performing several maneuvers including "funnels" (fast sideways flight in a circle) and in-place flips and rolls. With a "crude" model trained from demonstrations of these maneuvers, we can attempt the maneuver autonomously. If the helicopter does not complete the maneuver successfully, the model can be re-estimated, incorporating the data obtained during the failed trial. This new model more accurately captures the dynamics in the flight regimes actually encountered during the autonomous flight and hence can be used to achieve improved performance during subsequent attempts.

The observation that we can leverage pilot demonstrations to safely obtain "reasonable" models of the helicopter dynamics is the key to our approach. While these models may not be perfect at first, we can often obtain a good approximation to the true dynamics provided we attempt to model only a small portion of the flight envelope. This model can then, optionally, be improved by incorporating new data obtained from autonomous flights. Our trajectory learning algorithm (Section 4) exploits this same observation to achieve expert-level performance on an even broader range of maneuvers.

4. TRAJECTORY LEARNING

Once we are equipped with a (rudimentary) model of the helicopter dynamics, we need to specify the desired trajectory to be flown. Specifying the trajectory by hand, while tedious, can yield reasonable results. Indeed, much of our own previous work used hand-coded target trajectories.[2] Unfortunately these trajectories usually do not obey the system dynamics—that is, the hand-specified trajectory is infeasible, and cannot actually be flown in reality. This results in a somewhat more difficult control problem since the control algorithm must determine an appropriate tradeoff between the errors it must inevitably make. As well, it complicates our modeling process because we do not know, a priori, the trajectory that the controller will attempt to fly, and hence cannot focus our data collection in that region of state space.

One solution to these problems is to leverage expert demonstrations. By using a trajectory acquired from a demonstration aboard the real helicopter as the target trajectory we are guaranteed that our target is a feasible trajectory. Moreover, our data collection will already be focused on the proper flight regime, provided that our expert demonstrations cover roughly the same parts of state space each time. Thus, we expect that our model of the dynamics along the demonstrated trajectory will be reasonably accurate. This approach has been used successfully to perform autonomous autorotation landings with our helicopter.[1]

While the autorotation maneuver can be demonstrated relatively consistently by a skilled pilot,[b] it may be difficult or impossible to obtain a perfect demonstration that is suitable for use as a target trajectory when the maneuver does not include a steady-state regime, or involves complicated adjustments over long periods of time. For example, when our expert pilot attempts to demonstrate an in-place flip, the helicopter position often drifts away from its starting point unintentionally. Thus, when using this demonstration as our desired trajectory, the helicopter will repeat the pilot's errors. However, repeated expert demonstrations are often suboptimal in different ways, suggesting that a large number of demonstrations could implicitly encode the ideal trajectory that the (suboptimal) expert is trying to demonstrate.

In Coates,[12] we proposed an algorithm that approximately extracts this implicitly encoded optimal demonstration from multiple suboptimal expert demonstrations. This algorithm also allows us to build an improved, time-varying model of the dynamics along the resulting trajectory suitable for high-performance control. In doing so, the algorithm allows the helicopter to not only mimic the behavior of the expert but even perform significantly better.

Properly extracting the underlying ideal trajectory from a set of suboptimal trajectories requires a significantly more sophisticated approach than merely averaging the states observed at each time step. A simple arithmetic average of the states would result in a trajectory that does not obey the constraints of the dynamics model. Also, in practice, each of the demonstrations will occur at different rates so that attempting to combine states from the same time step in each trajectory will not work properly.

Following Coates,[12] we propose a generative model that describes the expert demonstrations as noisy observations of the unobserved, intended target trajectory, where each demonstration is possibly warped along the time axis. We use an expectation–maximization (EM) algorithm to both infer the unobserved, intended target trajectory and a time-alignment of all the demonstrations. The time-aligned demonstrations provide the appropriate data to learn good local models in the vicinity of the trajectory—such trajectory-specific local models tend to greatly improve control performance.

4.1. Basic generative model

From our expert pilot we obtain M demonstration trajectories of length N^k, for $k = 0..M - 1$. Each trajectory is a sequence of states, s_j^k, and control inputs, u_j^k, composed into a single state vector:

$$y_j^k = \begin{bmatrix} s_j^k \\ u_j^k \end{bmatrix} \text{ for } j = 0..N^k - 1,\ k = 0..M - 1.$$

Our goal is to estimate a "hidden" target trajectory of length H, denoted similarly:

$$z_t = \begin{bmatrix} s_t^\star \\ u_t^\star \end{bmatrix} \text{ for } t = 0..H.$$

We use the following notation: $y = \{y_j^k \mid j = 0..N^k - 1, k = 0..M - 1\}$, $\mathbf{z} = \{z_t | t = 0..H\}$, and similarly for other indexed variables.

The generative model for the ideal trajectory is given by an initial state distribution $z_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ and an approximate model of the dynamics

$$z_{t+1} = f(z_t) + w_t^{(z)}, \quad w_t^{(z)} \sim \mathcal{N}(0, \Sigma^{(z)}). \qquad (1)$$

The dynamics model does not need to be particularly accurate. In fact, in our experiments, this model is of the form described in Section 3, trained on a large corpus of data that is not even specific to the trajectory we want to fly.[c] In our experiments (Section 6) we provide some concrete examples showing how accurately the generic model captures the true dynamics for our helicopter.

Our generative model represents each demonstration as a set of independent "observations" of the hidden, ideal trajectory $\mathbf{z}$. Specifically, our model assumes

$$y_j^k = z_{\tau_j^k} + w_j^{(y)}, \quad w_j^{(y)} \sim \mathcal{N}(0, \Sigma^{(y)}). \qquad (2)$$

Here τ_j^k is the time index in the hidden trajectory to which the observation y_j^k is mapped. The noise term in the observation equation captures both inaccuracies in estimating the observed trajectories from sensor data, as well as errors in the maneuver that are the result of the human pilot's imperfect demonstration.[d]

[b] The autorotation maneuver consists of a steady-state "glide" followed by a short (several second) "flare" before landing. Though the maneuver is not easy to learn, these components tend not to vary much from one demonstration to the next.

[c] The state transition model also predicts the controls as a function of the previous state and controls. In our experiments we predict $u_{t+1}^\star$ as $u_t^\star$ plus Gaussian noise.

[d] Even though our observations, $\mathbf{y}$, are correlated over time with each other due to the dynamics governing the observed trajectory, our model assumes that the observations y_j^k are independent for all $j = 0..N^k - 1$ and $k = 0..M - 1$.

The time indices τ_j^k are unobserved, and our model assumes the following distribution with parameters d_i^k:

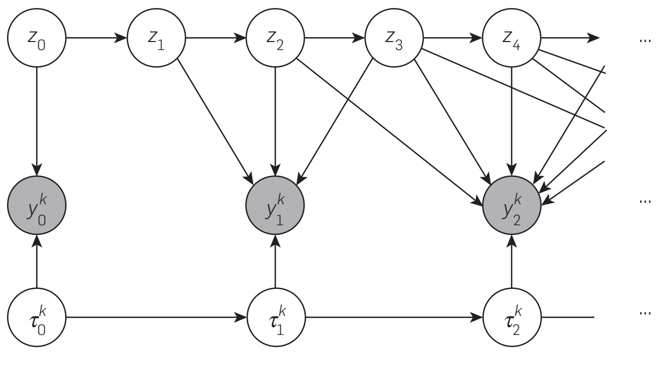
$$\mathbb{P}(\tau_{j+1}^k \mid \tau_j^k) = \begin{cases} d_1^k & \text{if } \tau_{j+1}^k - \tau_j^k = 1, \\ d_2^k & \text{if } \tau_{j+1}^k - \tau_j^k = 2, \\ d_3^k & \text{if } \tau_{j+1}^k - \tau_j^k = 3, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

$$\tau_0^k \equiv 0. \tag{4}$$

To accommodate small, gradual shifts in time between the hidden and observed trajectories, our model assumes the observed trajectories are subsampled versions of the hidden trajectory. We found that having a hidden trajectory length equal to twice the average length of the demonstrations, i.e., $H = 2\,(\frac{1}{M}\sum_{k=0}^{M-1} N^k)$, gives sufficient resolution.

Figure 1 depicts the graphical model corresponding to our basic generative model. Note that each observation y_j^k depends on the hidden trajectory's state at time τ_j^k, which means that for τ_j^k unobserved, y_j^k depends on all states in the hidden trajectory with which it could potentially be associated.

Figure 1: Graphical model representing our trajectory assumptions. (Shaded nodes are observed.)



4.2. Extensions to the generative model

We have assumed, thus far, that the expert demonstrations are misaligned copies of the ideal trajectory merely corrupted by Gaussian noise. Listgarten et al. have used this same basic generative model (for the case where $f(\cdot)$ is the identity function) to align speech signals and biological data.[19, 20] In our application to autonomous helicopter flight, we can augment the basic model described above to account for other sources of error that are important for modeling and control.

Learning Local Model Parameters: We can substantially improve our modeling accuracy by using a time-varying model $f_t(\cdot)$ that is specific to the vicinity of the intended trajectory at each time t.

We express f_t as our "crude" model (from Section 3), f, augmented with a bias term,[e] $\beta_t^\star$:

$$z_{t+1} = f_t(z_t) + w_t^{(z)} \equiv f(z_t) + \beta_t^\star + w_t^{(z)}.$$

To regularize our model, we assume that $\beta_t^\star$ changes only slowly over time. Specifically $\beta_{t+1}^\star \sim \mathcal{N}(\beta_t^\star, \Sigma^{(\beta)})$.

We incorporate the bias into our observation model by computing the observed bias $\beta_j^k = y_j^k - f(y_{j-1}^k)$ for each of the observed state transitions, and modeling this as a direct observation of the "true" model bias corrupted by Gaussian noise.

The result of this modification is that the ideal trajectory must not only look similar to the demonstration trajectories, but it must also obey a dynamics model which includes those modeling errors consistently observed in the demonstrations.

Factoring Out Demonstration Drift: It is often difficult, even for an expert pilot, during aerobatic maneuvers to keep the helicopter centered around a fixed position. The recorded position trajectory will often drift around unintentionally. Since these position errors are highly correlated, they are not explained well by the Gaussian noise term in the observation model. The basic dynamics model is easily augmented with "drift" terms to model these errors, allowing us to infer the drift included in each demonstration and remove it from the final result (see Coates[12] for details).

Incorporating Prior Knowledge: Even though it might be hard to specify the complete ideal trajectory in state space, we might still have prior knowledge about the trajectory. For example, for the case of a helicopter performing an in-place flip, our expert pilot can tell us that the helicopter should stay at a fixed position while it is flipping. We show in Coates[12] that these bits of knowledge can be incorporated into our model as additional noisy observations of the hidden states, where the variance of the noise expresses our confidence in the accuracy of the expert's advice. In the case of the flip, the variance expresses our knowledge that it is, in fact, impossible to flip perfectly in place and that the actual position of the helicopter may vary slightly from the position given by the expert.

4.3. Trajectory learning algorithm

Our learning algorithm will automatically find the time-alignment indices $\boldsymbol{\tau}$, the time-index transition probabilities $\mathbf{d}$, and the covariance matrices $\Sigma^{(\cdot)}$ by (approximately) maximizing the joint likelihood of the observed trajectories $\mathbf{y}$ and the observed prior knowledge about the ideal trajectory, $\boldsymbol{\rho}$, while marginalizing out over the unobserved, intended trajectory $\mathbf{z}$. Concretely, our algorithm (approximately) solves

$$\max_{\tau, \Sigma^{(\cdot)}, \mathbf{d}} \log \mathbb{P}(\mathrm{y}, \rho, \tau; \Sigma^{(\cdot)}, \mathbf{d}). \tag{5}$$

Then, once our algorithm has found $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$, it finds the most likely hidden trajectory, namely the trajectory $\mathbf{z}$ that maximizes the joint likelihood of the observed trajectories $\mathbf{y}$ and the observed prior knowledge about the ideal trajectory for the learned parameters $\boldsymbol{\tau}$, $\mathbf{d}$, $\Sigma^{(\cdot)}$. The joint optimization in

[e] Our generative model can incorporate richer local models. We discuss our choice of merely using biases in Coates.[12] We also show there how to estimate richer models post hoc using the output of our trajectory learning algorithm.

Equation 5 is difficult because (as can be seen in Figure 1) the lack of knowledge of the time-alignment index variables $\boldsymbol{\tau}$ introduces a very large set of dependencies between all the variables. However, when $\boldsymbol{\tau}$ is known, the optimization problem in Equation 5 greatly simplifies thanks to context specific independencies.[10] For instance, knowledge that $\tau_1^k = 3$ tells us that y_1^k depends only on z_3. Thus, when all of the $\boldsymbol{\tau}$ are fixed, we obtain a simplified model such as the one shown in Figure 2. In this model we can directly estimate the multinomial parameters $\mathbf{d}$ in closed form; and we have a standard HMM parameter learning problem for the covariances $\Sigma^{(\cdot)}$, which can be solved using the EM algorithm[13]—often referred to as Baum–Welch in the context of HMMs. Concretely, for our setting, the EM algorithm's E-step computes the pairwise marginals over sequential hidden state variables by running a (extended) Kalman smoother; the M-step then uses these marginals to update the covariances $\Sigma^{(\cdot)}$.

To also optimize over the time-indexing variables $\boldsymbol{\tau}$, we propose an alternating optimization procedure. For fixed $\Sigma^{(\cdot)}$ and $\mathbf{d}$, and for fixed $\mathbf{z}$, we can find the optimal time-indexing variables $\boldsymbol{\tau}$ using dynamic programming over the time-index assignments for each demonstration independently. The dynamic programming algorithm to find τ is known in the speech recognition literature as dynamic time warping[29] and in the biological sequence alignment literature as the Needleman–Wunsch algorithm.[21] The fixed $\mathbf{z}$ we use is the one that maximizes the likelihood of the observations for the current setting of parameters τ, $\mathbf{d}$, $\Sigma^{(\cdot)}$.[f]

In practice, rather than alternating between complete optimizations over $\Sigma^{(\cdot)}$, $\mathbf{d}$ and τ, we only partially optimize over $\Sigma^{(\cdot)}$, running only one iteration of the EM algorithm.

Complete details of the algorithm are provided in Coates.[12]

5. CONTROLLER DESIGN

Using the methods of Sections 3 and 4, we can obtain a good target trajectory and a high-accuracy dynamics model for this trajectory using pilot demonstrations. It remains to develop an adequate feedback controller that will allow the helicopter to fly this trajectory in reality. Our solution is based on the DDP algorithm, which we have used in previous work.[1, 2]

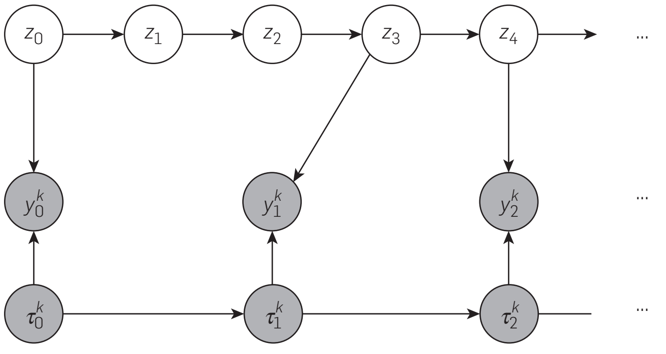
5.1. Reinforcement learning formalism and DDP

A reinforcement learning problem (or optimal control problem) can be described by a quintuple $(S, \mathcal{A}, \mathcal{T}, H, s_0, R)$, which is also referred to as a Markov decision process (MDP). Here S is the set of states; $\mathcal{A}$ is the set of actions or inputs; $\mathcal{T}$ is the dynamics model, which is a set of probability distributions $\{P^t_{su}\}$ ($P^t_{su}(s' \mid s, u)$ is the probability of being in state s' at time $t+1$ given the state and action at time t are s and u); H is the horizon or number of time steps of interest; $s_0 \in S$ is the initial state; $R: S \times \mathcal{A} \to \mathbb{R}$ is the reward function.

A policy $\pi = (\mu_0, \mu_1, \dots, \mu_H)$ is a tuple of mappings from states S to actions $\mathcal{A}$, one mapping for each time $t = 0, \dots, H$.

[f] Fixing $\mathbf{z}$ means the dynamic time warping step only approximately optimizes the original objective. Unfortunately, without fixing $\mathbf{z}$, the independencies required to obtain an efficient dynamic programming algorithm do not hold. In practice we find our approximation works very well.

Figure 2: Example of graphical model when τ is known. (Shaded nodes are observed.)



The expected sum of rewards when acting according to a policy π is given by: $\mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t) \mid \pi]$. The optimal policy $\pi^\star$ for an MDP $(S, \mathcal{A}, \mathcal{T}, H, s_0, R)$ is the policy that maximizes the expected sum of rewards. In particular, the optimal policy is given by

$$\pi^\star = \arg\max_{\pi} \mathrm{E}\left[\sum_{t=0}^{H} R(s_t, u_t) \mid \pi\right].$$

The linear quadratic regulator (LQR) control problem is a special class of MDP, for which the optimal policy can be computed efficiently. In LQR the set of states $S = \mathbb{R}^n$, the set of actions/inputs $\mathcal{A} = \mathbb{R}^p$, the dynamics model is given by

$$s_{t+1} = A_t s_t + B_t u_t + w_t,$$

where for all $t = 0, \dots, H$ we have that $A_t \in \mathbb{R}^{n\times n}$, $B_t \in \mathbb{R}^{n\times p}$ and w_t is a mean zero random variable (with finite variance). The reward for being in state s_t and taking action/input u_t is given by

$$-s_t^{\top} Q_t s_t - u_t^{\top} R_t u_t.$$

Here Q_t, R_t are positive semidefinite matrices which parameterize the reward function. It is well known that the optimal policy for the LQR control problem is a time-varying linear feedback controller, which can be efficiently computed using dynamic programming. (See, e.g., Anderson[7] for details on linear quadratic methods.)

The linear quadratic methods, which in their standard form as given above drive the state to zero, are easily extended to the task of tracking the desired trajectory $s_0^\star, \dots, s_H^\star$ learned in Section 4. The standard formulation (which we use) expresses the dynamics and reward function as a function of the error state $e_t = s_t - s_t^\star$ rather than the actual state s_t. (See, e.g., Anderson.[7])

DDP approximately solves general continuous state-space MDP's by iteratively approximating them by LQR problems. In particular, DDP solves an optimal control problem by iterating the following steps:

1. Around the trajectory obtained from running the current policy, compute: (i) a linear approximation to the

(nonlinear) error state dynamics and (ii) a quadratic approximation to the reward function.
2. Compute the optimal policy for the LQR problem obtained in Step 2 and set the current policy equal to the optimal policy for the LQR problem.
3. Simulate a trial starting from, s_0, under the current policy and store the resulting trajectory.

In our experiments, we have a quadratic reward function, thus the only approximation made in the algorithm is the linearization of the dynamics. To bootstrap the process (i.e., to obtain an initial trajectory), we linearize around the target trajectory in the first iteration.

The result of DDP is a sequence of linear feedback controllers that are executed in order. Since these controllers were computed under the assumption of linear dynamics, they will generally fail if executed from a state that is far from the linearization point. For aerobatic maneuvers that involve large changes in orientation, it is often difficult to remain sufficiently close to the linearization point throughout the maneuver. Our system, thus, uses DDP in a "receding horizon" fashion. Specifically, we rerun DDP *online*, beginning from the current state of the helicopter, over a horizon that extends 2 s into the future.[g] The resulting feedback controller obtained from this process is always linearized around the current state and, thus, allows the control system to continue flying even when it ventures briefly away from the intended trajectory.

5.2. Learning reward function parameters

Our quadratic reward is a function of 21 features (which are functions of the state and controls), consisting of the squared error state variables, the squared inputs, and squared change in inputs. Choosing the parameters for the reward function (i.e., choosing the entries of the matrices Q_t, R_t used by DDP) is difficult and tedious to do by hand. Intuitively, the reward parameters tell DDP how to "trade off" between the various errors. Selecting this trade-off improperly can result in some errors becoming too large (allowing the helicopter to veer off into poorly modeled parts of the state space), or other errors being regulated too aggressively (resulting in large, unsafe control outputs).

This problem is more troublesome when using infeasible target trajectories. For instance, for the aerobatic flips and rolls performed previously in Abbeel,[2] a hand-coded target trajectory was used. That trajectory was not feasible, since it assumed that the helicopter could remain exactly fixed in space during the flip. Thus, there is always a (large) nonzero error during the maneuver. In this case, the particular choice of reward parameters becomes critical, since they specify how the controller should balance errors throughout the flight.

Trajectories learned from demonstration using the methods presented in Section 4, however, are generally quite close to feasible for the real helicopter. Thus, in contrast to our prior work, the choice of trade-offs is less crucial when using these learned trajectories. Indeed, in our recent experiments it appears that a wide range of parameters work well with trajectories learned from demonstration.[h] Nonetheless, when the need to make adjustments to these parameters arises, it is useful to be able to learn the necessary parameters, rather than tune them by mere trial and error.

Since we have expert demonstrations of the desired behavior (namely, following the trajectory) we can alleviate the tuning problem by employing the apprenticeship learning via inverse reinforcement learning algorithm[4] to select appropriate parameters for our quadratic reward function. In practice, in early iterations (before convergence) this algorithm tends to generate parameters that are dangerous to use on the real helicopter. Instead, we adjust the reward weights by hand following the philosophy, but not the strict formulation of the inverse RL algorithm. In particular: we select the feature (state error) that differed most between our autonomous flights and the expert demonstrations, and then increase or decrease the corresponding quadratic penalties to bring the autonomous performance closer to that of the expert with each iteration.[i] Using this procedure, we obtain a good reward function in a small number of trials in practice.

We used this methodology to successfully select reward parameters to perform the flips and rolls in Abbeel,[2] and continue to use this methodology as a guide in selecting reward parameters.

6. EXPERIMENTAL RESULTS

6.1. Experimental setup

For our experiments we have used two different autonomous helicopters. The experiments presented here were performed with an XCell Tempest helicopter (Figure 3), but we have also conducted autonomous aerobatic flights using a Synergy N9. Both of these helicopters are capable of professional, competition-level maneuvers. We instrumented our helicopters with a Microstrain 3DM-GX1 orientation sensor. A ground-based camera system measures the helicopter's position. A Kalman filter uses these measurements to track the helicopter's position, velocity, orientation, and angular rate.

We collected multiple demonstrations from our expert for a variety of aerobatic trajectories: continuous in-place flips and rolls, a continuous tail-down "tic toc," and an airshow, which consists of the following maneuvers in rapid sequence: split-S, snap roll, stall-turn, loop, loop with pirouette, stall-turn with pirouette, "hurricane" (fast backward funnel), knife-edge, flips and rolls, tic-toc, and inverted hover.

We use a large, previously collected corpus of hovering, horizontal flight, and mixed aerobatic flight data to build a crude dynamics model using the method of Section 3. This model and the pilot demonstrations are then provided to the trajectory learning algorithm of Section 4. Our trajectory

[g] The 2 s horizon is a limitation imposed by available computing power. Our receding horizon DDP controller executes at 20 Hz.

[h] It is often sufficient to simply choose parameters that rescale the various reward features to have approximately the same magnitude.

[i] For example, if our controller consistently uses larger controls than the expert but achieves lower position error, we would increase the control penalty and decrease the position penalty.

Figure 3: Our XCell Tempest autonomous helicopter.

learning algorithm includes bias terms, β_t^*, for each of the predicted accelerations, and hence will learn a time-dependent acceleration that is added to the crude base model. We also include terms to model position drift in the pilot demonstrations, and incorporate our prior knowledge that flips and rolls should remain roughly in place, and that maneuvers like loops should be flown in a plane (i.e., they should look flat when viewed from the top).[12]

6.2. Trajectory learning results

Figure 4(a) shows the horizontal and vertical position of the helicopter during the two loops flown during the airshow performed by our pilot. The colored lines show the expert pilot's demonstrations. The black dotted line shows the inferred ideal path produced by our algorithm. The loops are more rounded and more consistent in the inferred ideal path. We did not incorporate any prior knowledge to this effect. Figure 4(b) shows a top-down view of the same demonstrations and inferred trajectory. This view shows that the algorithm successfully inferred a trajectory that lies in a vertical plane, while obeying the system dynamics, as a result of the included prior knowledge.

Figure 4(c) shows one of the bias terms, namely the prediction errors made by our crude model for the z-axis acceleration of the helicopter for each of the demonstrations (plotted as a function of time). Figure 4(d) shows the result after alignment (in color) as well as the inferred acceleration error (black dotted). We see that the bias measurements allude to errors approximately in the −1G to −2G range for the first 40 s of the airshow (a period that involves high-G maneuvering that is not predicted accurately by the "crude" model). However, only the aligned biases precisely show the magnitudes and locations of these errors along the trajectory. The alignment allows us to build our ideal trajectory based upon a much more accurate model that is tailored to match the dynamics observed in the demonstrations.

6.3. Flight results

After constructing the idealized trajectories and models using our algorithms, we attempted to fly the trajectories on the actual helicopter. As described in Section 5, we use a receding-horizon DDP controller.[15] Our trajectory learning algorithm provides us with desired state and control trajectories, as well as an accurate, time-varying dynamics model tailored to the trajectory. These are provided to our DDP implementation along with quadratic reward weights chosen previously using the method described in Section 5.2. The quadratic reward function penalizes deviation from the target trajectory, s_t^*, as well as deviation from the desired controls, u_t^*, and the desired control velocities, $u_{t+1}^* - u_t^*$.

We compare the result of this procedure first with the former state of the art in aerobatic helicopter flight, namely the in-place rolls and flips of Abbeel.[2] That work used a single crude model, developed using the method of Section 3, along with hand-specified target trajectories, and reward weights tuned using the methodology in Section 5.2.

Figure 5(a) shows the Y–Z position[j] and the collective (thrust) control inputs for the in-place rolls performed by the controller in Abbeel[2] and our controller using receding-horizon DDP and the outputs of our trajectory learning algorithm. Our new controller achieves (i) better position performance and (ii) lower overall collective control values (which roughly represents the amount of energy being used to fly the maneuver).

Similarly, Figure 5(b) shows the X–Z position and the collective control inputs for the in-place flips for both controllers. Like for the rolls, we see that our controller significantly outperforms the previous approach, both in position accuracy and in control energy expended.

[j] These are the position coordinates projected into a plane orthogonal to the axis of rotation.

Figure 4: Colored lines: demonstrations. Black dotted line: trajectory inferred by our algorithm. (See text for details.)

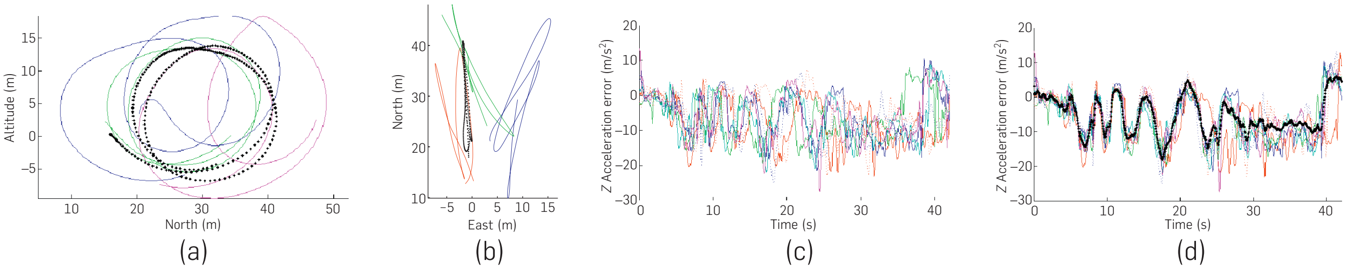
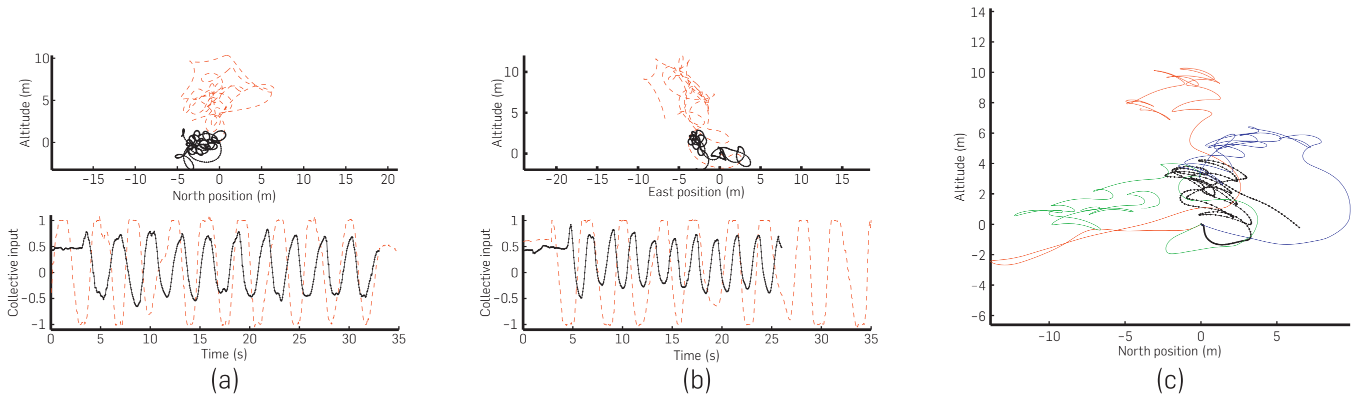


Figure 5: Flight results. (a, b) Solid black: results with trajectory learning algorithm. Dashed red: results with hand-coded trajectory from Abbeel.[2] (c) Dotted black: autonomous tic-toc. Solid colored: expert demonstrations.



Besides flips and rolls, we also performed autonomous "tic tocs"—widely considered to be an even more challenging aerobatic maneuver. During the (tail-down) tic-toc maneuver the helicopter pitches quickly backward and forward in place with the tail pointed toward the ground (resembling an inverted clock pendulum). The complex relationship between pitch angle, horizontal motion, vertical motion, and thrust makes it extremely difficult to create a feasible tic-toc trajectory by hand. Our attempts to use such a hand-coded trajectory, following the previous approach in Abbeel,[2] failed repeatedly. By contrast, the trajectory learning algorithm readily yields an excellent feasible trajectory that was successfully flown on the first attempt. Figure 5(c) shows the expert trajectories (in color), and the autonomously flown tic-toc (black dotted). Our controller significantly outperforms the expert's demonstrations.

We also applied our algorithm to successfully fly a complete aerobatic airshow, as described in Section 6.1.

The trajectory-specific models typically capture the dynamics well enough to fly all the aforementioned maneuvers reliably. Since our computer controller flies the trajectory very consistently, however, this allows us to repeatedly acquire data from the same vicinity of the target trajectory on the real helicopter. Thus, we can incorporate this flight data into our model, allowing us to improve flight accuracy even further. For example, during the first autonomous airshow our controller achieves an RMS position error of 3.29 m, and this procedure improved performance to 1.75 m RMS position error.

Videos of all our flights are available at: http://heli.stanford.edu

7. CONCLUSION

We have presented learning algorithms that take advantage of expert demonstrations to successfully fly autonomous helicopters at the level of an expert human pilot. In particular, we have shown how to (i) build a rough global model from demonstration data, (ii) approximately infer the expert's ideal desired trajectory, (iii) learn accurate, trajectory-specific local models suitable for high-performance control, and (iv) build control systems using the outputs of our trajectory learning algorithm. Our experiments demonstrated that this design pipeline enables our controllers to fly extreme aerobatic maneuvers. Our results have shown that our system not only significantly outperforms the previous state of the art, but even outperforms our own expert pilot on a wide variety of difficult maneuvers.

Acknowledgments

We thank Garett Oku for piloting and building our helicopters. Adam Coates is supported by a Stanford Graduate Fellowship. This work was also supported in part by the DARPA Learning Locomotion program under contract number FA8650-05-C-7261.

References

1. Abbeel, P., Coates, A., Hunter, T., Ng, A.Y. Autonomous autorotation of an RC helicopter. *ISER 11* (2008).
2. Abbeel, P., Coates, A., Quigley, M., Ng, A.Y. An application of reinforcement learning to aerobatic helicopter flight. *NIPS 19* (2007), 1–8.
3. Abbeel, P., Ganapathi, V., Ng, A. Learning vehicular dynamics, with application to modeling helicopters. *NIPS 18* (2006), 1–8.
4. Abbeel, P., Ng, A.Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of ICML* (2004).
5. Abbeel, P., Quigley, M., Ng, A.Y. Using inaccurate models in reinforcement learning. In *Proceedings of ICML* (2006), ACM, NY, 1–8.
6. An, C.H., Atkeson, C.G., Hollerbach, J.M. *Model-Based Control of a Robot Manipulator*. MIT Press, 1988.
7. Anderson, B., Moore, J. *Optimal Control: Linear Quadratic Methods.* Prentice-Hall, 1989.
8. Atkeson, C., Schaal, S. Robot learning from demonstration. In *Proceedings of ICML* (1997).
9. Bagnell, J., Schneider, J. Autonomous helicopter control using reinforcement learning policy search methods. In *IEEE International Conference on Robotics and Automation* (2001).
10. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D. Context-specific independence in Bayesian networks. In *Proceedings of UAI* (1996).
11. Calinon, S., Guenter, F., Billard, A. On learning, representing and generalizing a task in a humanoid robot. In *IEEE Transactions on Systems, Man and Cybernetics, Part B*, volume 37, 2007.
12. Coates, A., Abbeel, P., Ng, A.Y. Learning for control from multiple demonstrations. In *Proceedings of ICML* (2008), 144–151.
13. Dempster, A.P., Laird, N.M., Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc.* (1977).
14. Gavrilets, V., Martinos, I., Mettler, B., Feron, E. Control logic for automated aerobatic flight of miniature helicopter. In *AIAA Guidance,*

Navigation and Control Conference (2002).
15. Jacobson, D.H., Mayne, D.Q. *Differential Dynamic Programming*. Elsevier, 1970.
16. La Civita, M. *Integrated Modeling and Robust Control for Full-Envelope Flight of Robotic Helicopters*. PhD thesis, Carnegie Mellon University, 2003.
17. La Civita, M., Papageorgiou, G., Messner, W.C., Kanade, T. Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. *J. Guid. Control. Dynam.*, *29*, 2 (Mar.–Apr. 2006), 485–494.
18. Leishman, J. *Principles of Helicopter Aerodynamics*. Cambridge University Press, 2000.
19. Listgarten, J. *Analysis of Sibling Time Series Data: Alignment and Difference Detection*. PhD thesis, University of Toronto, 2006.
20. Listgarten, J., Neal, R.M., Roweis, S.T., Emili, A. Multiple alignment of continuous time series. In *NIPS 17* (2005).
21. Needleman, S., Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 1970.
22. Neu, G., Szepesvari, C. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of UAI* (2007).
23. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E. Autonomous inverted helicopter flight via reinforcement learning. In *ISER* (2004).
24. Ng, A.Y., Kim, H.J., Jordan, M., Sastry, S. Autnonomous helicopter flight via reinforcement learning. In *NIPS 16* (2004).
25. Ng, A.Y., Russell, S. Algorithms for inverse reinforcement learning. In *Proceedings of ICML* (2000).
26. Ramachandran, D., Amir, E. Bayesian inverse reinforcement learning. In *Proceedings of IJCAI* (2007).
27. Ratliff, N., Bagnell, J., Zinkevich, M. Maximum margin planning. In *Proceedings of ICML* (2006).
28. Roberts, J.M., Corke, P.I., Buskey, G. Low-cost flight control system for a small autonomous helicopter. In *IEEE International Conference on Robotics and Automation* (2003).
29. Sakoe, H., Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1978).
30. Saripalli, S., Montgomery, J., Sukhatme, G. Visually-guided landing of an unmanned aerial vehicle, 2003.
31. Seddon, J. *Basic Helicopter Aerodynamics*. AIAA Education Series, American Institute of Aeronautics and Astronautics, 1990.
32. Syed, U., Schapire, R.E. A game-theoretic approach to apprenticeship learning. In *NIPS 20* (2008).

Adam Coates (acoates@cs.stanford.edu), Computer Science Department, Stanford University, Stanford, CA.

Pieter Abbeel (pabbeel@eecs.berkeley.edu), Computer Science Division, University of California, Berkeley, CA.

Andrew Y. Ng (ang@cs.stanford.edu), Computer Science Depament, Stanford University, Stanford, CA.

© 2009 ACM 0001-0782/09/0600 $10.00

Take Advantage of
ACM's Lifetime Membership Plan!

- ACM Professional Members can enjoy the convenience of making a single payment for their entire tenure as an ACM Member, and also be protected from future price increases by taking advantage of ACM's Lifetime Membership option.
- ACM Lifetime Membership dues may be tax deductible under certain circumstances, so becoming a Lifetime Member can have additional advantages if you act before the end of 2009. (Please consult with your tax advisor.)
- Lifetime Members receive a certificate of recognition suitable for framing, and enjoy all of the benefits of ACM Professional Membership.

Learn more and apply at:
http://www.acm.org/life

acm Association for Computing Machinery
Advancing Computing as a Science & Profession