Learning for Control from Multiple Demonstrations

Adam Coates ACOATES@CS.STANFORD.EDU
Pieter Abbeel PABBEEL@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Stanford University CS Department, 353 Serra Mall, Stanford, CA 94305 USA

Abstract

We consider the problem of learning to follow a desired trajectory when given a small number of demonstrations from a sub-optimal expert. We present an algorithm that (i) extracts the—initially unknown—desired trajectory from the sub-optimal expert's demonstrations and (ii) learns a local model suitable for control along the learned trajectory. We apply our algorithm to the problem of autonomous helicopter flight. In all cases, the autonomous helicopter's performance exceeds that of our expert helicopter pilot's demonstrations. Even stronger, our results significantly extend the state-of-the-art in autonomous helicopter aerobatics. In particular, our results include the first autonomous tic-tocs, loops and hurricane, vastly superior performance on previously performed aerobatic maneuvers (such as in-place flips and rolls), and a complete airshow, which requires autonomous transitions between these and various other maneuvers.

1. Introduction

Many tasks in robotics can be described as a trajectory that the robot should follow. Unfortunately, specifying the desired trajectory and building an appropriate model for the robot dynamics along that trajectory are often non-trivial tasks. For example, when asked to describe the trajectory that a helicopter should follow to perform an aerobatic flip, one would have to specify a trajectory that (i) corresponds to the aerobatic flip task, and (ii) is consistent with the helicopter's dynamics. The latter requires (iii) an accurate helicopter dynamics model for all of the flight regimes encountered in the vicinity of the trajectory. These coupled tasks are non-trivial for systems with complex dynamics, such as helicopters. Failing to adequately address these points leads to a significantly more difficult control problem.

In the apprenticeship learning setting, where an expert is available, rather than relying on a hand-engineered target trajectory, one can instead have the expert demonstrate the desired trajectory. The expert demonstration yields both a desired trajectory for the robot to follow, as well as data to build a dynamics model in the vicinity of this trajectory. Unfortunately, perfect demonstrations can be hard (if not impossible) to obtain. However, repeated expert demonstrations are often suboptimal in different ways, suggesting that a large number of suboptimal expert demonstrations could implicitly encode the ideal trajectory the suboptimal expert is trying to demonstrate.

In this paper we propose an algorithm that approximately extracts this implicitly encoded optimal demonstration from multiple suboptimal expert demonstrations, and then builds a model of the dynamics in the vicinity of this trajectory suitable for high-performance control. In doing so, the algorithm learns a target trajectory and a model that allows the robot 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 even 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.

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 present an EM algorithm—which uses a (extended) Kalman smoother and an efficient dynamic programming algorithm to perform the E-step—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

Appearing in *Proceedings of the 25^{th} International Conference on Machine Learning*, Helsinki, Finland, 2008.

good local models in the vicinity of the trajectory—such trajectory-specific local models tend to greatly improve control performance.

Our algorithm allows one to easily incorporate prior knowledge to further improve the quality of the learned trajectory. For example, for a helicopter performing in-place flips, it is known that the helicopter can be roughly centered around the same position over the entire sequence of flips. Our algorithm incorporates this prior knowledge, and successfully factors out the position drift in the expert demonstrations.

We apply our algorithm to learn trajectories and dynamics models for aerobatic flight with a remote controlled helicopter. Our experimental results show that (i) our algorithm successfully extracts a good trajectory from the multiple sub-optimal demonstrations, and (ii) the resulting flight performance significantly extends the state of the art in aerobatic helicopter flight (Abbeel et al., 2007; Gavrilets et al., 2002). Most importantly, our resulting controllers are the first to perform as well, and often even better, than our expert pilot.

We posted movies of our autonomous helicopter flights at:

`http://heli.stanford.edu`

The remainder of this paper is organized as follows: Section 2 presents our generative model for (multiple) suboptimal demonstrations; Section 3 describes our trajectory learning algorithm in detail; Section 4 describes our local model learning algorithm; Section 5 describes our helicopter platform and experimental results; Section 6 discusses related work.

2. Generative Model

2.1. Basic Generative Model

We are given 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 T, denoted similarly:

$$z_t = \begin{bmatrix} s_t^\star \\ u_t^\star \end{bmatrix}, \text{ for } t = 0..T-1.$$

We use the following notation: $\mathbf{y} = \{y_j^k \mid j = 0..N^k - 1, k = 0..M-1\}$, $\mathbf{z} = \{z_t \mid t = 0..T-1\}$, 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

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

The dynamics model does not need to be particularly accurate—in our experiments, we use a single generic model learned from a large corpus of data that is not specific to the trajectory we want to perform. In our experiments (Section 5) we provide some concrete examples showing how accurately the generic model captures the true dynamics for our helicopter.[1]

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

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

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 inaccuracy 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.[2]

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

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

$$\tau_0^k \equiv 0. \qquad (4)$$

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., $T = 2(\frac{1}{M}\sum_{k=1}^M N^k)$, 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 that it could be associated with.

2.2. Extensions to the Generative Model

Thus far we have assumed that the expert demonstrations are misaligned copies of the ideal trajectory

[1] 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.

[2] 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 $k = 0..M-1$.

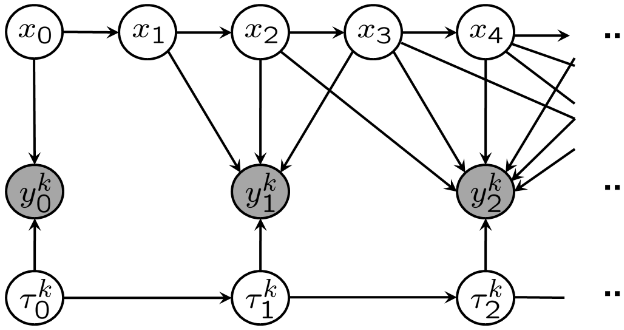


Figure 1. Graphical model representing our trajectory assumptions. (Shaded nodes are observed.)

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 (Listgarten, 2006; Listgarten et al., 2005). We now augment the basic model to account for other sources of error which are important for modeling and control.

2.2.1. Learning Local Model Parameters

For many systems, 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, f, augmented with a bias term[3], $\beta_t^\star$:

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

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

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 errors consistently observed in the demonstrations.

2.2.2. 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 our observation model.

To capture such slow drift in the demonstrated trajectories, we augment the latent trajectory's state with a "drift" vector δ_t^k for each time t and each demonstrated trajectory k. We model the drift as a zero-mean random walk with (relatively) small variance. The state observations are now noisy measurements of $z_t + \delta_t^k$ rather than merely z_t.

2.2.3. 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. Hence, we introduce additional observations $\rho_t = \rho(z_t)$ corresponding to our prior knowledge about the ideal trajectory at time t. The function $\rho(z_t)$ computes some features of the hidden state z_t and our expert supplies the value ρ_t that this feature should take. For example, for the case of a helicopter performing an in-place flip, we use an observation that corresponds to our expert pilot's knowledge that the helicopter should stay at a fixed position while it is flipping. We assume that these observations may be corrupted by Gaussian noise, 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.

Incorporating prior knowledge of this kind can greatly enhance the learned ideal trajectory. We give more detailed examples in Section 5.

2.2.4. Model Summary

In summary, we have the following generative model:

$$\begin{aligned} z_{t+1} &= f(z_t) + \beta_t^\star + \omega_t^{(z)}, & (5)\\ \beta_{t+1}^\star &= \beta_t^\star + \omega_t^{(\beta)}, & (6)\\ \delta_{t+1}^k &= \delta_t^k + \omega_t^{(\delta)}, & (7)\\ \rho_t &= \rho(z_t) + \omega_t^{(\rho)}, & (8)\\ y_j^k &= z_{\tau_j^k} + \delta_j^k + \omega_j^{(y)}, & (9)\\ \tau_j^k &\sim \mathbb{P}(\tau_{j+1}^k | \tau_j^k) & (10) \end{aligned}$$

Here $\omega_t^{(z)}, \omega_t^{(\beta)}, \omega_t^{(\delta)}, \omega_t^{(\rho)}, \omega_j^{(y)}$ are zero mean Gaussian random variables with respective covariance matrices $\Sigma^{(z)}, \Sigma^{(\beta)}, \Sigma^{(\delta)}, \Sigma^{(\rho)}, \Sigma^{(y)}$. The transition probabilities for τ_j^k are defined by Eqs. (3, 4) with parameters d_1^k, d_2^k, d_3^k (collectively denoted $\mathbf{d}$).

3. Trajectory Learning Algorithm

Our learning algorithm automatically finds the time-alignment indexes $\boldsymbol{\tau}$, 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 knowl-

[3]Our generative model can incorporate richer local models. We discuss our choice of merely using biases in our generative trajectory model in more detail in Section 4.

edge about the ideal trajectory $\boldsymbol{\rho}$, while marginalizing out over the unobserved, intended trajectory $\mathbf{z}$. Concretely, our algorithm (approximately) solves

$$\max_{\boldsymbol{\tau},\Sigma^{(\cdot)},\mathbf{d}} \log \mathbb{P}(\mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau} \; ; \; \Sigma^{(\cdot)}, \mathbf{d}). \tag{11}$$

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 $\boldsymbol{\rho}$ for the learned parameters $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.[4]

The joint optimization in Eq. (11) 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 Eq. (11) greatly simplifies thanks to context specific independencies (Boutilier et al., 1996). When $\boldsymbol{\tau}$ is fixed, we obtain a 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 (Dempster et al., 1977)—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 $\Sigma^{(\cdot)}$.

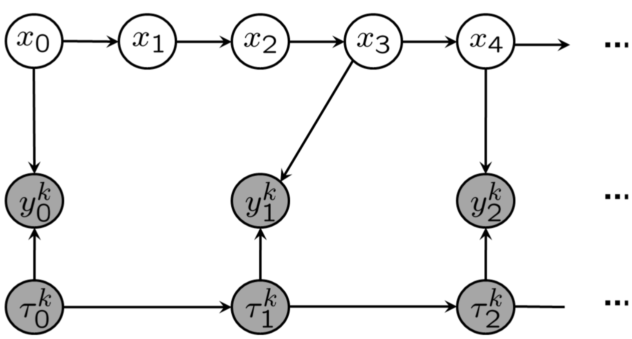


Figure 2. Example of graphical model when $\boldsymbol{\tau}$ is known. (Shaded nodes are observed.)

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 $\boldsymbol{\tau}$ is known in the speech recognition literature as dynamic time warping (Sakoe & Chiba, 1978) and in the biological sequence alignment literature as the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970). The fixed $\mathbf{z}$ we use, is the one that maximizes the likelihood of the observations for the current setting of parameters $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.[5]

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

We provide the complete details of our algorithm in the full paper (Coates et al., 2008).

4. Local Model Learning

For complex dynamical systems, the state z_t used in the dynamics model often does not correspond to the "complete state" of the system, since the latter could involve large numbers of previous states or unobserved variables that make modeling difficult.[6] However, when we only seek to model the system dynamics along a specific trajectory, knowledge of both z_t and how far we are along that trajectory is often sufficient to accurately predict the next state z_{t+1}.

Once the alignments between the demonstrations are computed by our trajectory learning algorithm, we can use the time aligned demonstration data to learn a sequence of trajectory-specific models. The time indices of the aligned demonstrations now accurately associate the demonstration data points with locations along the learned trajectory, allowing us to build models for the state at time t using the appropriate corresponding data from the demonstration trajectories.[7]

[4]Note maximizing over the hidden trajectory and the covariance parameters simultaneously introduces undesirable local maxima: the likelihood score would be highest (namely infinity) for a hidden trajectory with a sequence of states exactly corresponding to the (crude) dynamics model $f(\cdot)$ and state-transition covariance matrices equal to all-zeros as long as the observation covariances are nonzero. Hence we marginalize out the hidden trajectory to find $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.

[5]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.

[6]This is particularly true for helicopters. Whereas the state of the helicopter is very crudely captured by the 12D rigid-body state representation we use for our controllers, the "true" physical state of the system includes, among others, the airflow around the helicopter, the rotor head speed, and the actuator dynamics.

[7]We could learn the richer local model within the trajectory alignment algorithm, updating the dynamics model during the M-step. We chose not to do so since these models are more computationally expensive to estimate. The richer models have minimal influence on the alignment because the biases capture the average model error—the richer models capture the derivatives around it. Given the limited influence on the alignment, we chose to save computational time and only estimate the richer models after

Figure 3. Our XCell Tempest autonomous helicopter.

To construct an accurate nonlinear model to predict z_{t+1} from z_t, using the aligned data, one could use locally weighted linear regression (Atkeson et al., 1997), where a linear model is learned based on a weighted dataset. Data points from our aligned demonstrations that are nearer to the current time index along the trajectory, t, and nearer the current state, z_t, would be weighted more highly than data far away. While this allows us to build a more accurate model from our time-aligned data, the weighted regression must be done online, since the weights depend on the current state, z_t. For performance reasons[8] this may often be impractical. Thus, we weight data only based on the time index, and learn a parametric model in the remaining variables (which, in our experiments, has the same form as the global "crude" model, $f(\cdot)$). Concretely, when estimating the model for the dynamics at time t, we weight a data point at time t' by:[9]

$$W(t') = \exp\left(-\frac{(t-t')^2}{\sigma^2}\right),$$

where σ is a bandwidth parameter. Typical values for σ are between one and two seconds in our experiments. Since the weights for the data points now only depend on the time index, we can precompute all models $f_t(\cdot)$ along the entire trajectory. The ability to precompute the models is a feature crucial to our control algorithm, which relies heavily on fast simulation.

5. Experimental Results

5.1. Experimental Setup

To test our algorithm, we had our expert helicopter pilot fly our XCell Tempest helicopter (Figure 3), which can perform professional, competition-level maneuvers.[10]

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.

The (crude) helicopter dynamics $f(\cdot)$ is constructed using the method of Abbeel et al. (2006a).[11] The helicopter dynamics model predicts linear and angular accelerations as a function of current state and inputs. The next state is then obtained by integrating forward in time using the standard rigid-body equations.

In the trajectory learning algorithm, we have bias terms $\beta_t^\star$ for each of the predicted accelerations. We use the state-drift variables, δ_t^k, for position only.

For the flips, rolls, and tic-tocs we incorporated our prior knowledge that the helicopter should stay in place. We added a measurement of the form:

$$0 = p(z_t) + \omega^{(\rho_0)}, \quad \omega^{(\rho_0)} \sim \mathcal{N}(0, \Sigma^{(\rho_0)})$$

where $p(\cdot)$ is a function that returns the position coordinates of z_t, and $\Sigma^{(\rho_0)}$ is a diagonal covariance matrix. This measurement—which is a direct observation of the pilot's intended trajectory—is similar to advice given to a novice human pilot to describe the desired maneuver: A good flip, roll, or tic-toc trajectory stays close to the same position.

We also used additional advice in the airshow to indicate that the vertical loops, stall-turns and split-S should all lie in a single vertical plane; that the hurricanes should lie in a horizontal plane and that a good knife-edge stays in a vertical plane. These measurements take the form:

$$c = N^\top p(z_t) + \omega^{(\rho_1)}, \quad \omega^{(\rho_1)} \sim \mathcal{N}(0, \Sigma^{(\rho_1)})$$

where, again, $p(z_t)$ returns the position coordinates of z_t. N is a vector normal to the plane of the maneuver, c is a constant, and $\Sigma^{(\rho_1)}$ is a diagonal covariance matrix.

alignment.

[8]During real-time control execution, our model is queried roughly 52000 times per second. Even with KD-tree or cover-tree data structures a full locally weighted model would be much too slow.

[9]In practice, the data points along a short segment of the trajectory lie in a low-dimensional subspace of the state space. This sometimes leads to an ill-conditioned parameter estimation problem. To mitigate this problem, we regularize our models toward the "crude" model $f(\cdot)$.

[10]We instrumented the helicopter 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.

[11]The model of Abbeel et al. (2006a) naturally generalizes to any orientation of the helicopter regardless of the flight regime from which data is collected. Hence, even without collecting data from aerobatic flight, we can reasonably attempt to use such a model for aerobatic flying, though we expect it to be relatively inaccurate.

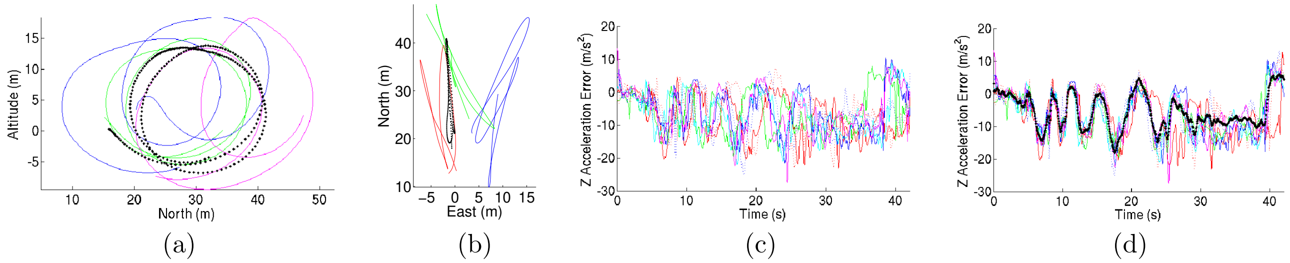


Figure 4. Colored lines: demonstrations. Black dotted line: trajectory inferred by our algorithm. (See text for details.)

5.2. Trajectory Learning Results

Figure 4(a) shows the horizontal and vertical position of the helicopter during the two loops flown during the airshow. 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 extent. Figure 4(b) shows a top-down view of the same demonstrations and inferred trajectory. The prior successfully encouraged the inferred trajectory to lie in a vertical plane, while obeying the system dynamics.

Figure 4(c) shows one of the bias terms, namely the model prediction errors for the Z-axis acceleration of the helicopter computed from the demonstrations, before time-alignment. Figure 4(d) shows the result after alignment (in color) as well as the inferred acceleration error (black dotted). We see that the unaligned bias measurements allude to errors approximately in the -1G to -2G range for the first 40 seconds 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.

Results for other maneuvers and state variables are similar. At the URL provided in the introduction we posted movies which simultaneously replay the different demonstrations, before alignment and after alignment. The movies visualize the alignment results in many state dimensions simultaneously.

5.3. Flight Results

After constructing the idealized trajectory and models using our algorithm, we attempted to fly the trajectory on the actual helicopter.

Our helicopter uses a receding-horizon differential dynamic programming (DDP) controller (Jacobson & Mayne, 1970). DDP approximately solves general continuous state-space optimal control problems by taking advantage of the fact that optimal control problems with linear dynamics and a quadratic reward function (known as linear quadratic regulator (LQR) problems) can be solved efficiently. It is well-known that the solution to the (time-varying, finite horizon) LQR problem is a sequence of linear feedback controllers. In short, DDP iteratively approximates the general control problem with LQR problems until convergence, resulting in a sequence of linear feedback controllers that are approximately optimal. In the receding-horizon algorithm, we not only run DDP initially to design the sequence of controllers, but also re-run DDP during control execution at every time step and recompute the optimal controller over a fixed-length time interval (the horizon), assuming the precomputed controller and cost-to-go are correct after this horizon.

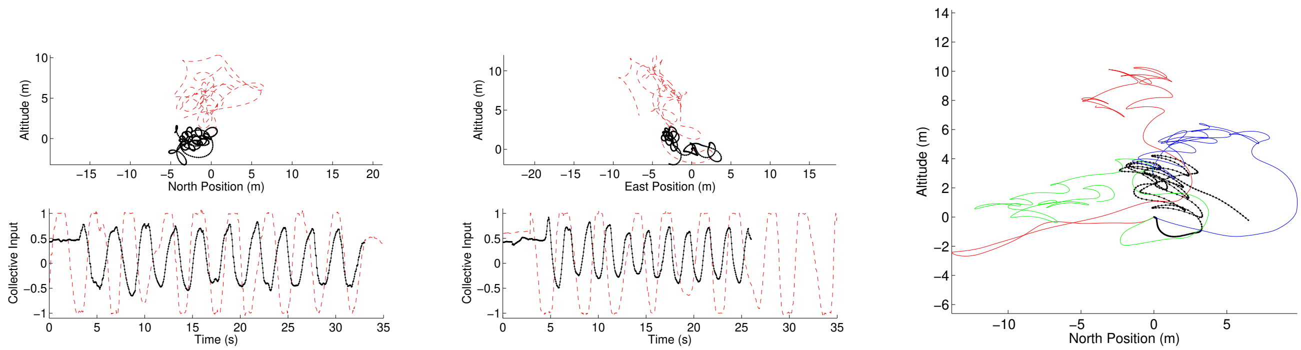
As described in Section 4, our algorithm outputs a sequence of learned local parametric models, each of the form described by Abbeel et al. (2006a). Our implementation linearizes these models on the fly with a 2 second horizon (at 20Hz). Our reward function penalizes error from the target trajectory, $s_t^\star$, as well as deviation from the desired controls, $u_t^\star$, and the desired control velocities, $u_{t+1}^\star - u_t^\star$.

First we compare our results with the previous state-of-the-art in aerobatic helicopter flight, namely the in-place rolls and flips of Abbeel et al. (2007). That work used hand-specified target trajectories and a single nonlinear model for the entire trajectory.

Figure 5(a) shows the Y-Z position[12] and the collective (thrust) control inputs for the in-place rolls for both their controller and ours. Our controller achieves (i) better position performance (standard deviation of approximately 2.3 meters in the Y-Z plane, compared to about 4.6 meters 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 that of Abbeel et al. (2007), both in position accuracy and in control energy expended.

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



(a) (b) (c)

Figure 5. Flight results. (a),(b) Solid black: our results. Dashed red: Abbeel et al. (2007). (c) Dotted black: autonomous tic-toc. Solid colored: expert demonstrations. (See text for details.)

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 with the DDP algorithm from (Abbeel et al., 2007) failed repeatedly. By contrast, our 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, 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.

The trajectory-specific local model learning typically captures 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. Similar to Abbeel et al. (2007), we incorporate this flight data into our model learning, 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 meters, and this procedure improved performance to 1.75 meters RMS position error.

Videos of all our flights are available at:

`http://heli.stanford.edu`

6. Related Work

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

Atkeson and Schaal (1997) 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. (2007) considered learning trajectories and constraints from demonstrations for robotic tasks. There, they do not consider the system's dynamics or provide a clear mechanism for the inclusion of prior knowledge. Our formulation presents a principled, joint optimization which takes into account the multiple demonstrations, as well as the (complex) system dynamics and prior knowledge. While Calinon et al. (2007) also use some form of dynamic time warping, they do not try to optimize a joint objective capturing both the system dynamics and time-warping.

Among others, An et al. (1988) and, more recently, Abbeel et al. (2006b) have exploited the idea of trajectory-indexed model learning for control. However, contrary to our setting, their algorithms do not time align nor coherently integrate data from multiple trajectories.

While the work by Listgarten et al. (Listgarten, 2006; Listgarten et al., 2005) does not consider robotic control and model learning, they also consider the problem of multiple continuous time series alignment with a hidden time series.

Our work also has strong similarities with recent work on inverse reinforcement learning, which extracts a reward function (rather than a trajectory) from the expert demonstrations. See, e.g., Ng and Russell (2000); Abbeel and Ng (2004); Ratliff et al. (2006); Neu and Szepesvari (2007); Ramachandran and Amir (2007); Syed and Schapire (2008).

Most prior work on autonomous helicopter flight only considers the flight-regime close to hover. There are three notable exceptions. The aerobatic work of Gavrilets et al. (2002) comprises three maneuvers: split-S, snap-roll, and stall-turn, which we also include during the first 10 seconds of our airshow for comparison. They record pilot demonstrations, and then hand-engineer a sequence of desired angular rates and velocities, as well as transition points. Ng et al. (2004) have their autonomous helicopter perform sustained inverted hover. We compared the performance of our system with the work of Abbeel et al. (2007), by far the most advanced autonomous aerobatics results to date, in Section 5.

7. Conclusion

We presented an algorithm that takes advantage of multiple suboptimal trajectory demonstrations to (i) extract (an estimate of) the ideal demonstration, (ii) learn a local model along this trajectory. Our algorithm is generally applicable for learning trajectories and dynamics models along trajectories from multiple demonstrations. We showed the effectiveness of our algorithm for control by applying it to the challenging problem of autonomous helicopter aerobatics. The ideal target trajectory and the local models output by our trajectory learning algorithm enable our controllers to significantly outperform the prior state of the art.

Acknowledgments

We thank Garett Oku for piloting and building our helicopter. 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

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. *NIPS 19*.

Abbeel, P., Ganapathi, V., & Ng, A. Y. (2006a). Learning vehicular dynamics with application to modeling helicopters. *NIPS 18*.

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Proc. ICML*.

Abbeel, P., Quigley, M., & Ng, A. Y. (2006b). Using inaccurate models in reinforcement learning. *Proc. ICML*.

An, C. H., Atkeson, C. G., & Hollerbach, J. M. (1988). *Model-based control of a robot manipulator*. MIT Press.

Atkeson, C., & Schaal, S. (1997). Robot learning from demonstration. *Proc. ICML*.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning for control. *Artificial Intelligence Review, 11*.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. *Proc. UAI*.

Calinon, S., Guenter, F., & Billard, A. (2007). On learning, representing and generalizing a task in a humanoid robot. *IEEE Trans. on Systems, Man and Cybernetics, Part B*.

Coates, A., Abbeel, P., & Ng, A. Y. (2008). Learning for control from multiple demonstrations (Full version). http://heli.stanford.edu/icml2008.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society*.

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002). Control logic for automated aerobatic flight of miniature helicopter. *AIAA Guidance, Navigation and Control Conference*.

Jacobson, D. H., & Mayne, D. Q. (1970). *Differential dynamic programming*. Elsevier.

Listgarten, J. (2006). *Analysis of sibling time series data: alignment and difference detection*. Doctoral dissertation, University of Toronto.

Listgarten, J., Neal, R. M., Roweis, S. T., & Emili, A. (2005). Multiple alignment of continuous time series. *NIPS 17*.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*

Neu, G., & Szepesvari, C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. *Proc. UAI*.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. *ISER*.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. *Proc. ICML*.

Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. *Proc. IJCAI*.

Ratliff, N., Bagnell, J., & Zinkevich, M. (2006). Maximum margin planning. *Proc. ICML*.

Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*.

Syed, U., & Schapire, R. E. (2008). A game-theoretic approach to apprenticeship learning. *NIPS 20*.

A. Trajectory Learning Algorithm

As described in Section 3, our algorithm (approximately) solves

$$\max_{\boldsymbol{\tau}, \Sigma^{(\cdot)}, \mathbf{d}} \log \mathbb{P}(\mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau} \ ; \ \Sigma^{(\cdot)}, \mathbf{d}). \tag{12}$$

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 $\boldsymbol{\rho}$ for the learned parameters $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.

To optimize Eq. (12), we alternatingly optimize over $\Sigma^{(\cdot)}$, $\mathbf{d}$ and $\boldsymbol{\tau}$. Section 3 provides the high-level description, below we provide the detailed description of our algorithm.

1. Initialize the parameters to hand-chosen defaults. A typical choice: $\Sigma^{(\cdot)} = I$, $d_i^k = \frac{1}{3}$, $\tau_j^k = \lceil j \frac{T-1}{N^k-1} \rceil$.
2. E-step for latent trajectory: For the current setting of $\boldsymbol{\tau}, \Sigma^{(\cdot)}$ run a (extended) Kalman smoother to find the distributions for the latent states, $\mathcal{N}(\mu_{t|T-1}, \Sigma_{t|T-1})$.
3. M-step for latent trajectory: Update the covariances $\Sigma^{(\cdot)}$ using the standard EM update.
4. E-step for the time indexing (using hard assignments): run dynamic time warping to find $\boldsymbol{\tau}$ that maximizes the joint probability $\mathbb{P}(\bar{\mathbf{z}}, \mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau})$, where $\bar{\mathbf{z}}$ is fixed to $\mu_{t|T-1}$, namely the mode of the distribution obtained from the Kalman smoother.
5. M-step for the time indexing: estimate $\mathbf{d}$ from $\boldsymbol{\tau}$.
6. Repeat steps 2-5 until convergence.

A.1. Steps 2 and 3 details—EM for non-linear dynamical systems

Steps 2 and 3 in our algorithm correspond to the standard E and M steps of the EM algorithm applied to a non-linear dynamical system with Gaussian noise. For completeness we provide the details below.

In particular, we have:

$$\begin{aligned} z_{t+1} &= f(z_t) + \omega_t, \quad \omega_t \sim \mathcal{N}(0, Q), \\ y_{t+1} &= h(z_t) + \nu_t, \quad \nu_t \sim \mathcal{N}(0, R). \end{aligned}$$

In the E-step, for $t = 0..T-1$, the Kalman smoother computes the parameters $\mu_{t|t}$ and $\Sigma_{t|t}$ for the distribution $\mathcal{N}(\mu_{t|t}, \Sigma_{t|t})$, which is the distribution of z_t conditioned on all observations up to and including time t. Along the way, the smoother also computes $\mu_{t+1|t}$ and $\Sigma_{t+1|t}$. These are the parameters for the distribution of z_{t+1} given only the measurements up to time t. Finally, during the backward pass, the parameters $\mu_{t|T-1}$ and $\Sigma_{t|T-1}$ are computed, which give the distribution for z_t given all measurements.

After running the Kalman smoother (for the E-step), we can use the computed quantities to update Q and R in the M-step. In particular, we can compute[13]:

$$\begin{aligned}
\delta\mu_t &= \mu_{t+1|T-1} - f(\mu_{t|T-1}), \\
A_t &= \mathcal{D}f(\mu_{t|T-1}), \\
L_t &= \Sigma_{t|t} A_t^\top \Sigma_{t+1|t}^{-1}, \\
P_t &= \Sigma_{t+1|T-1} - \Sigma_{t+1|T-1} L_t^\top A_t^\top - A_t L_t \Sigma_{t+1|T-1}, \\
Q &= \frac{1}{T} \sum_{t=0}^{T-1} \delta\mu_t \delta\mu_t^\top + A_t \Sigma_{t|T-1} A_t^\top + P_t, \\
\delta y_t &= y_t - h(\mu_{t|T-1}), \\
C_t &= \mathcal{D}h(\mu_{t|T-1}), \\
R &= \frac{1}{T} \sum_{t=0}^{T-1} \delta y_t \delta y_t^\top + C_t \Sigma_{t|T-1} C_t^\top.
\end{aligned}$$

A.2. Steps 4 and 5 details—Dynamic time warping

In Step 4 our goal is to compute $\bar{\boldsymbol{\tau}}$ as:

$$\begin{aligned}
\bar{\boldsymbol{\tau}} &= \arg\max_{\boldsymbol{\tau}} \log \mathbb{P}(\bar{\mathbf{z}}, \mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau}; \Sigma^{(\cdot)}, \mathbf{d}) \\
&= \arg\max_{\boldsymbol{\tau}} \log \mathbb{P}(\mathbf{y}|\bar{\mathbf{z}}, \boldsymbol{\tau}) \mathbb{P}(\boldsymbol{\rho}|\bar{\mathbf{z}}) \mathbb{P}(\bar{\mathbf{z}}) \mathbb{P}(\boldsymbol{\tau}) \quad (13) \\
&= \arg\max_{\boldsymbol{\tau}} \log \mathbb{P}(\mathbf{y}|\bar{\mathbf{z}}, \boldsymbol{\tau}) \mathbb{P}(\boldsymbol{\tau})
\end{aligned}$$

where $\bar{\mathbf{z}}$ is the mode of the distribution computed by the Kalman smoother (namely, $\bar{z}_t = \mu_{t|T-1}$) and Eq. (13) made use of independence assumptions implied by our model (see Figure 2). Again, using independence properties, the log likelihood above can be expanded to:

$$\bar{\boldsymbol{\tau}} = \arg\max_{\boldsymbol{\tau}} \sum_{k=0}^{M-1} \sum_{j=0}^{N^k-1} \left[\ell(y_j^k | \bar{z}_{\tau_j^k}, \tau_j^k) + \ell(\tau_j^k | \tau_{j-1}^k) \right] \tag{14}$$

Note that the inner summations in the above expression are independent—the likelihoods for each of the M observation sequences can be maximized separately. Hence, in the following, we will omit the k superscript, as the algorithm can be applied separately for each sequence of observations and indices.

At this point, we can solve the maximization over $\boldsymbol{\tau}$ using a dynamic programming algorithm known in the speech recognition literature as dynamic time warping (Sakoe & Chiba, 1978) and in the biological sequence alignment literature as the Needleman-Wunsch

[13] The notation $\mathcal{D}f(z)$ is the Jacobian of f evaluated at z.

algorithm (Needleman & Wunsch, 1970). For completeness, we provide the details for our setting below.

We define the quantity $\mathcal{Q}(s,t)$ to be the maximum obtainable value of the first $s+1$ terms of the inner summation if we choose $\tau_s = t$.

For $s = 0$ we have:

$$\mathcal{Q}(0,t) = \ell(y_0|\bar{z}_{\tau_0}, \tau_0 = t) + \ell(\tau_0 = t) \tag{15}$$

And for $s > 0$:

$$\begin{aligned}\mathcal{Q}(s,t) = \ell(y_s|\bar{z}_{\tau_s}, \tau_s = t) \\ + \max_{\tau_1,\ldots,\tau_{s-1}} [\ell(\tau_s = t|\tau_{s-1}) \\ + \sum_{j=0}^{s-1} \left[\ell(y_j|\bar{z}_{\tau_j}, \tau_j) + \ell(\tau_j|\tau_{j-1})\right]]\end{aligned}$$

The latter equation can be written recursively as:

$$\begin{aligned}\mathcal{Q}(s,t) = \ell(y_s|\bar{z}_{\tau_s}, \tau_s = t) + \\ \max_{t'} [\ell(\tau_s = t|\tau_{s-1} = t') + \mathcal{Q}(s-1,t')]\end{aligned} \tag{16}$$

The equations (15) and (16) can be used to compute $\max_t \mathcal{Q}(N^k - 1, t)$ for each observation sequence (and the maximizing solution, $\boldsymbol{\tau}$), which is exactly the maximizing value of the inner summation in Eq. (14). The maximization in Eq. (16) can be restricted to the relevant values of t'. In our application, we only allow $t' \in \{t-3, t-2, t-1\}$. As is common practice, we typically restrict the time-index assignments to a fixed-width band around the default, equally-spaced alignment. In our case, we only compute $\mathcal{Q}(s,t)$ if $2s - C \leq t \leq 2s + C$, for fixed C.

In Step 5 we compute the parameters $\mathbf{d}$ using standard maximum likelihood estimates for multinomial distributions.