2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

A Probabilistic Approach to Mixed Open-loop and Closed-loop Control, with Application to Extreme Autonomous Driving

J. Zico Kolter, Christian Plagemann, David T. Jackson, Andrew Y. Ng, Sebastian Thrun

Abstract— **We consider the task of accurately controlling a complex system, such as autonomously sliding a car sideways into a parking spot. Although certain regions of this domain are extremely hard to model (i.e., the dynamics of the car while skidding), we observe that in practice such systems are often remarkably deterministic over short periods of time, even in difficult-to-model regions. Motivated by this intuition, we develop a probabilistic method for combining closed-loop control in the well-modeled regions and open-loop control in the difficult-to-model regions. In particular, we show that by combining 1) an inaccurate model of the system and 2) a demonstration of the desired behavior, our approach can accurately and robustly control highly challenging systems, without the need to explicitly model the dynamics in the most complex regions and without the need to hand-tune the switching control law. We apply our approach to the task of autonomous sideways sliding into a parking spot, and show that we can repeatedly and accurately control the system, placing the car within about 2 feet of the desired location; to the best of our knowledge, this represents the state of the art in terms of accurately controlling a vehicle in such a maneuver.**

I. INTRODUCTION

In this paper, we consider control strategies that employ a mixture of *closed-loop* control (actively controlling the system based on state feedback) and *open-loop* control (executing a fixed sequence of control inputs without any feedback). To motivate such strategies, we focus on the task of sliding a car sideways into a parking spot — that is, accelerating a car to 25 mph, then slamming on the brakes while turning the steering wheel, skidding the car sideways into a narrow desired location; this is a highly challenging control task, and therefore an excellent testbed for learning and control algorithms. One feature of this and many other control tasks is that the system operates in multiple distinct regimes that can be either easy or hard to model: for instance, a "normal driving" regime, where the dynamics of the car are well understood and easily modeled, and a "sideways sliding" regime, where the dynamics are extremely difficult to model due to time-delay or non-Markovian effects.

Although it may appear that controlling the system in these hard-to-model regimes is nearly impossible, if we have previously observed a trajectory demonstrating the desired behavior from the current or nearby state, then we could execute this same behavior in hopes of ending up in a similar final state. It may seem surprising that such a method has any chance of success, due to the stochasticity of the world. However, we repeatedly observe in practice that the world is often remarkably deterministic over short periods of time, given a fixed sequence of control inputs, even if it is very hard to model how the system would respond to different inputs; we have observed this behavior in systems such as the sideways-sliding car, quadruped robots, and helicopters. Intuitively, this observation motivates an alternating control strategy, where we would want to actively control the system using classical methods in well-modeled regimes, but execute open-loop trajectories in poorly modeled regions, provided we have seen a previous demonstration of the desired behavior. Indeed, as we will discuss shortly, such strategies have previously been used to control a variety of systems, though these have mostly relied on hand-tuned switching rules to determine when to execute the different types of control.

In this paper, we make two contributions. First, we develop a probabilistic approach to mixed open-loop and closed-loop control that achieves performance far superior to each of these elements in isolation. In particular, we propose a method for applying optimal control algorithms to a probabilistic combination of 1) a "simple" model of the system that can be highly inaccurate in some regions and 2) a dynamics model that describes the expected system evolution when executing a previously-observed trajectory. By using variance estimates of these different models, our method probabilistically interpolates between open-loop and closed-loop control, thus producing mixed controllers without any hand-tuning. Our second contribution is that we apply this algorithm to the aforementioned task of autonomously sliding a car sideways into a parking spot, and show that we can repeatedly and accurately control the car, without any need to model the dynamics in the sliding regime. To the best of our knowledge, this is the first autonomous demonstration of such behavior on a car, representing the state of the art in accurate control for such types of maneuvers.

II. BACKGROUND AND RELATED WORK

There is a great deal of work within both the machine learning and control communities that is relevant to the work we have presented here. As mentioned, previously, the mixed use of open-loop and closed-loop control is a common strategy in robotics and control (e.g., [3], [10], [23]), though typically these involve hand-tuning the regions of closed-loop/open-loop control. The work of Hansen et al., [9] also considers a mixture of open-loop and closed-loop control, though here the impetus for such control comes from a cost to sensing actions, not from difficulty of modeling the system, and thus the algorithms are very different.

All authors are with the Computer Science Department, Stanford University, Stanford CA, 94305 {kolter, plagem, dtj, ang, thrun}@cs.stanford.edu

978-1-4244-5040-4/10/$26.00 ©2010 IEEE

Our work also relates to work on multiple models for controls tasks [14]. However, with few exceptions the term "multiple models" is typically synonymous with "local models," building or learning a collection of models that are each accurate in a different portion of the state space – for example, *locally linear models*, [18], [5], [24] or non-parametric models such as Gaussian Processes [13]. While our algorithm can be generally interpreted as an application of a particular type of local models, the basic intuition behind the way we combine models is different: as we will discuss, we aren't seeking to necessarily learn a better model, nor is each model always tailored to a specific region along the trajectory; rather the models we will use capture different elements — an inaccurate model of the whole system, and a model of the trajectory that provides no information at all about state or control derivatives. Doya et al., [7] also propose a model-based reinforcement learning algorithm based on multiple models, but again these models are local in nature.

From the control literature, our work perhaps most closely resembles the work on multiple model adaptive control [15], [19]. Like our work, this line of research takes motivation from the fact that different models can capture the system dynamics with different degrees of uncertainty, but again like the previous work in machine learning, this research has typically focused on learning or adapting multiple local models of the system to achieve better performance.

Our work also relates somewhat more loosely to a variety of other research in reinforcement learning. For instance, the notion of "trajectories" or "trajectory libraries" has been explored for reinforcement learning and robotics [20], [21], but this work has primarily focused on the trajectories as a means for speeding up *planning* in a difficult task. Abbeel et al., [1] consider the problem of reinforcement learning using inaccurate models, but their approach still relies on the inaccurate model having accurate derivatives, and indeed the *pure LQR* approach we employ is very similar to their algorithm provided we linearize around a trajectory from the real system.

The notion of learning *motor primitives* has received a great deal of attention recently (e.g. [16]), though such research is generally orthogonal to what we present here. These motor primitives typically involve feedback policies that are not model-based, and we imagine that similar techniques to what we propose here could also be applied to learn how to switch between model-based and motor-primitive-based control. Indeed to some extent our work touches on the issue of model-based versus model-free reinforcement learning [4] where our open-loop controllers are simply a very extreme example of model-free control, but we could imagine applying similar algorithms to more general model-free policies.

Finally, there has been a great deal of past work on autonomous car control, for example [11]. Some of this work has even dealt with slight forays into control at the limits of handling [12], but all such published work that we are aware of demonstrates significantly less extreme situations than what we describe here, or includes only simulation results. We are also aware of another group at Stanford working independently on control at the limits of handling [8], but this work revolves more around physics-based techniques for stabilizing the system in an unstable "sliding" equilibrium, rather than executing maneuvers with high accuracy.

III. Preliminaries

We consider a discrete-time, continuous state, and continuous action environment, where the state at some time t is denoted as $s_t \in \mathbb{R}^n$ and the control as $u_t \in \mathbb{R}^m$. The dynamics of the system are governed by some unknown (and noisy) dynamics

$$s_{t+1} = f(s_t, u_t) + \epsilon_t,$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ gives the mean state and ϵ_t is some zero-mean noise term.

The control task we consider is that of following a desired *trajectory* on the system, $\{s^\star_{0:H}, u^\star_{0:H-1}\}$.[1] More formally, we want to choose controls that will minimize the expected cost

$$J = \mathbf{E}\Bigg[\sum_{t=0}^{H-1}\Big(\delta s_t^T Q \delta s_t + \delta u_t^T R \delta u_t\Big) + \delta s_H^T Q \delta s_H \;\Bigg|\; s_{t+1} = f(s_t, u_t) + \epsilon_t\Bigg] \quad (1)$$

where $\delta s_t \equiv s_t - s^\star_t$ and $\delta u_t \equiv u_t - u^\star_t$ denote the state and control error, and where $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m \times m}$ are symmetric positive semidefinite matrices that determine how the cost function trades off errors in different state and control variables. Even when the desired trajectory is "realizable" on the true system — for example, if it comes from an expert demonstration on the real system — minimizing this cost function can be highly non-trivial: due to the stochasticity of the world, the system will naturally deviate from the intended path, and we will need to execute controls to steer the system back to its intended path.

A. Linear Quadratic Regulator control

While there are many possible methods for minimizing the cost function (1), one method that is particularly well-suited to this task is the Linear Quadratic Regulator (LQR) algorithm (sometimes also referred to as the Linear Quadratic Tracker when the task is specifically to follow some target trajectory) — there have been many recent successes with LQR-based control methods in the both the machine learning and control communities [6], [22]. Space constraints preclude a full review of the algorithm, but for a more detailed description of such methods see e.g. [2]. Briefly, the LQR algorithm assumes that the (error) dynamics of the system evolve according to a *linear* model

$$\delta s_{t+1} = A_t \delta s_t + B_t \delta u_t + w_t \quad (2)$$

[1] Note that in this formulation the trajectory includes both states *and* controls and could be produced, for instance, by a human expert demonstrating the control task. In cases where the controls (or the desired intermediate states) are *not* specified, we could use a variety of planning algorithms to determine controls and desired states, though this is an orthogonal issue.

where $A_t \in \mathbb{R}^{n\times n}$ and $B_t \in \mathbb{R}^{n\times m}$ are system matrices, and w_t is a zero-mean noise term. Under this assumption, it is well-known that the optimal policy for minimizing the quadratic cost (1) is a *linear* policy — i.e., the optimal control is given by $\delta u_t = K_t \delta s_t$ for some matrix $K_t \in \mathbb{R}^{m\times n}$ that can be computed using dynamic programming.

To apply LQR to our control task, we of course need to specify the actual A_t and B_t matrices. Ideally, these matrices would be the state and action derivatives (of the true model) evaluated along the trajectory. However, since we do not have access to the true model, we typically determine these matrices using an *approximate* model. More formally, we suppose we have some approximate model of the form

$$s_{t+1} \approx \hat{f}(s_t, u_t)$$

where, as before, $\hat{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ predicts the next state given the current state and control, and we set A_t and B_t to be the Jacobians of this model, evaluated along the target trajectory

$$A_t = D_s \hat{f}(s,u)\Big|_{s=s_t^\star, u=u_t^\star} \quad B_t = D_u \hat{f}(s,u)\Big|_{s=s_t^\star, u=u_t^\star}.$$

Despite the fact that we now are approximating the original control problem in two ways — since we use an approximate model $\hat{f}$ and additionally make a linear approximations to this function — the algorithm can work very well in practice, *provided the approximate model captures the derivatives of the true system well* (where we are intentionally somewhat vague in what we mean by "well"). When the approximation model is not accurate in this sense (for example, if it misspecifies the sign of certain elements in the derivatives at some point along the trajectory), then the LQR algorithm can perform very poorly.

IV. A PROBABILISTIC FRAMEWORK FOR MIXED CLOSED-LOOP AND OPEN-LOOP CONTROL

In this section, we present the main algorithmic contribution of the paper, a probabilistic method for mixed closed-loop and open-loop control. We first present a general method, called Multi-model LQR, for combining multiple probabilistic dynamics models with LQR — the model combination approach uses estimates of model variances and methods for combining Gaussian predictions, but some modifications are necessary to integrate fully with LQR. Then, we present a simple model for capturing the dynamics of executing open-loop trajectories, and show that when we combine this with an inaccurate model using Multi-model LQR, the resulting algorithm naturally trades off between actively controlling the system in well-modeled regimes, and executing trajectories in unmodelled portions where we have previously observed the desired behavior.

A. LQR with multiple probabilistic models

Suppose we are given two approximate models of a dynamical system M_1 and M_2 — the generalization to more models is straightforward and we consider two just for simplicity of presentation. We also suppose that these approximate models are probabilistic, in that for a given state and control they return a Gaussian distribution over next states, with mean $f_i(s_t, u_t)$ and covariance $\Sigma_i(s_t, u_t)$, i.e.,

$$p(s_{t+1}|M_i) = \mathcal{N}(f_i(s_t, u_t), \Sigma_i(s_t, u_t)).$$

The covariance terms $\Sigma_i(s_t, u_t)$ are interpreted as maximum likelihood covariance estimates, i.e., for true next state s_{t+1},

$$\Sigma_i(s_t, u_t) = E\left[(s_{t+1} - f_i(s_t,u_t))(s_{t+1} - f_i(s_t,u_t))^T\right]$$

so the covariance terms capture two sources of error: 1) the true stochasticity of the world (assumed small), and 2) the inaccuracy of the model. Thus, even if the world is largely deterministic, model variance still provides a suitable means for combining multiple probabilistic models.

To combine multiple probabilistic models of this form, we interpret each prediction as an independent observation of the true next state, and can then compute the posterior distribution over the next state given both models by a standard manipulation of Gaussian probabilities,

$$p(s_{t+1}|M_1, M_1) = \mathcal{N}(\bar{f}(s_t, u_t), \bar{\Sigma}(s_t, u_t))$$

where

$$\bar{\Sigma} = \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)^{-1}, \text{ and } \bar{f} = \bar{\Sigma}\left(\Sigma_1^{-1} f_1 + \Sigma_2^{-1} f_2\right)$$

and where the dependence on s_t and u_t is omitted to simplify the notation in this last line. In other words, the posterior distribution over next states is a weighted average of the two predictions, where each model is weighted by its inverse covariance. This combination is analogous to the Kalman filter measurement update.

We could now directly apply LQR to this joint model by simply computing $\bar{f}(s_t^\star, u_t^\star)$ and its derivatives at each point along the trajectory — i.e., we could run LQR exactly as described in the previous section using the dynamics (2) where A_t and B_t are the Jacobians of $\bar{f}$, evaluated at $s_t^\star$ and $u_t^\star$, and where $w_t \sim \mathcal{N}(0, \bar{\Sigma}(s_t^\star, u_t^\star))$. However, combining these models in this manner will lead to poor performance. This is due to the fact that we would be computing $\Sigma_1(s_t^\star, u_t^\star)$ and $\Sigma_2(s_t^\star, u_t^\star)$ only at the desired location, which can give a very inaccurate estimate of each model's true covariance. For example, consider a learned model that is trained only on the desired trajectory; such a model would have very low variance at the actual desired point on the trajectory, but much higher variance when predicting nearby points. Thus, what we really want to compute is the *average* covariance of each model, in the region where we expect the system to be. We achieve this effect by maintaining a *distribution* D_t over the expected state (errors) at time t. Given such a distribution, we can approximate the average covariance

$$\Sigma_i(D_t) \equiv \mathbf{E}_{\delta s_t, \delta u_t \sim D_t}\left[\Sigma_i(s_t^\star + \delta s_t, u_t^\star + \delta u_t)\right]$$

via sampling or other methods.

The last remaining element we need is a method for computing the state error distributions D_t. Fortunately, given the linear model assumptions that we already employ for LQR, computing this distribution analytically is straightforward. In

Algorithm 1
Multi-model LQR($D_{0:H}$, M_1, M_2, $s^\star_{0:H}$, $u^\star_{0:H-1}$, Q, R)

Input:

$D_{0:H}$: initial state error distributions
M_1, M_2: probabilistic dynamics models
$s^\star_{0:H}, u^\star_{0:H-1}$: desired trajectory
Q, R: LQR cost matrices

Repeat until convergence:

1. For $t = 0, \ldots, H-1$, compute dynamics models
   - $\Sigma_i(D_t) \leftarrow \mathbf{E}_{\delta s_t, \delta u_t \sim D_t}\left[\Sigma_i(s^\star_t + \delta s_t, u^\star_t + \delta u_t)\right]$, $i = 1, 2$ (computed via sampling)
   - $\bar{\Sigma}_t \leftarrow \left(\Sigma_1^{-1}(D_t) + \Sigma_2^{-1}(D_t)\right)^{-1}$
   - $\bar{f} \leftarrow \bar{\Sigma}_t \left(\Sigma_1^{-1}(D_t) f_1 + \Sigma_2^{-1}(D_t) f_2\right)$
   - $A_t \leftarrow D_s \bar{f}(s, u)\big|_{s=s^\star_t, u=u^\star_t}$
   - $B_t \leftarrow D_s \bar{f}(s, u)\big|_{s=s^\star_t, u=u^\star_t}$
2. Run LQR to find optimal controllers K_t
   - $K_{0:H-1} \leftarrow \mathrm{LQR}(A_{0:H-1}, B_{0:H-1}, Q, R)$
3. For $t = 0, \ldots, H-1$, update state distributions
   - $\Gamma_{t+1} \leftarrow (A_t + B_t K_t)\Gamma_t (A_t + B_t K_t)^T + \bar{\Sigma}(D_t)$
   - $D_t \leftarrow \{\delta s_t \sim \mathcal{N}(0, \Gamma_t),\ \delta u_t \sim \mathcal{N}(0, K_t \Gamma_t K_t^T)\}$

particular, we first assume that our initial state distribution D_0 is a zero-mean Gaussian with mean zero and covariance Γ_0. Then, since our state dynamics evolve according to the linear model (2), at each time step the covariance is updated by

$$\Gamma_{t+1} = (A_t + B_t K_t)\Gamma_t (A_t + B_t K_t)^T + \bar{\Sigma}(D_t).$$

Of course, since we average the different models according to the $\Sigma_i(D_t)$, changing D_t will therefore also change A_t and B_t. Thus, we iteratively compute all these quantities until convergence. A formal description of the algorithm is given in Algorithm 1.

B. A dynamics model for open-loop trajectories

Here we propose a simple probabilistic model that predicts how the state will evolve when executing a previously observed sequence of control actions. While generally one would want to select from any number of possible trajectories to execute, for the simplified algorithm in this section we consider only the question of executing the control actions of the ideal trajectory $u^\star_{0:H-1}$.

To allow our model to be as general as possible, we assume a very simple probabilistic description of how the states evolve when executing these fixed sequences of actions. In particular, we assume that the error dynamics evolve according to

$$\delta s_{t+1} = \rho \delta s_t + w_t$$

where $\rho \in \mathbb{R}$ (typically ≈ 1) indicates the stability of taking such trajectory actions, and where w_t is a zero-mean Gaussian noise term with covariance $\Sigma(\delta s_t, \delta u_t)$, which depends only on the state and control error and which captures the covariance of the model as a function of how far away we are from the desired trajectory. This model captures the following intuition: if we are close to a previously observed trajectory, then we expect the next error state to be similar, with a variance that increases the more we deviate from the previously observed states and actions.

To see how this model naturally leads to trading off between actively controlling the system and largely following known trajectory controls, we consider a situation where we run the Multi-model LQR algorithm with an inaccurate model[2] M_1 and this described trajectory model, M_2. Consider a situation where our state distribution D_t is tightly peaked around the desired state, but where M_1 predicts the next state very poorly. In this case, the maximum likelihood covariance $\Sigma_1(D_t)$ will be large, because M_1 predicts poorly, but $\Sigma_2(D_t)$ will be much smaller, since we are still close to the desired trajectory. Therefore $\bar{f}$ will be extremely close to the trajectory model M_2, and so will lead to system matrices $A_t \approx \rho I$ and $B_t \approx 0$ (the derivatives of the trajectory model). Since this model predicts that no control can affect the state very much, LQR will choose controls $\delta u_t \approx 0$ — i.e., execute controls very close to $u^\star_t$, as this will minimize the expected cost function. In contrast, if we are far away from the desired trajectory, or if the model M_1 is very accurate, then we would expect to largely follow this model, as it would have lower variance than the naive trajectory model. In practice, the system will smoothly interpolate between these two extremes based on the variances of each model.

C. Estimating Variances

Until now, we have assumed that the covariance terms $\Sigma_i(s_t, u_t)$ have been given, though in practice these will need to be estimated from data. Recall that the variances we want to learn are ML estimates of the form

$$\Sigma_i(s_t, u_t) = E\left[(s_{t+1} - f_i(s_t, u_t))(s_{t+1} - f_i(s_t, u_t))^T\right]$$

so we could use any number of estimation methods to learn these covariances, and our algorithm is intentionally agnostic about how these covariances are obtained. Some modeling algorithms, such as Gaussian Processes, naturally provide such estimates, but for other approaches we need to use additional methods to estimate the covariances. Since we employ different methods for estimating the variance in the two sets of experiments below, we defer a description of the methods to these sections below.

V. Experiments

In this section we present experimental results, first on a simple simulated cart-pole domain, and then on our main application task for the algorithm, the task of autonomous sideways sliding into a parking spot.

[2]For the purposes of this discussion it does not matter how M_1 is obtained, and could be built, for example, from first principles, or learned from data, as we do for the car in Section V.

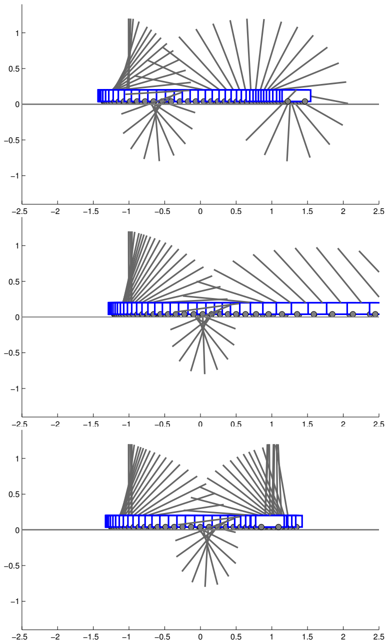
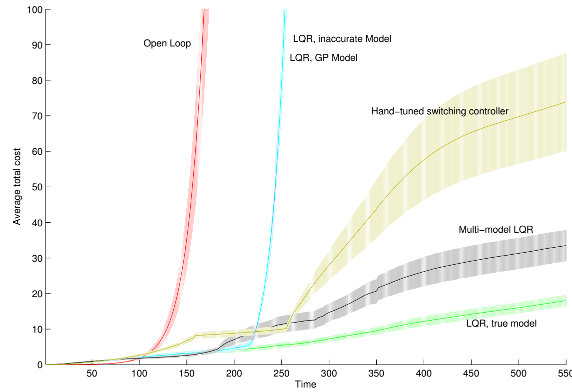


Fig. 1. (top) Trajectory followed by the cart-pole under open-loop control (middle) LQR control with an inaccurate model and (bottom) Multi-model LQR with both the open loop trajectory and inaccurate model.



Method	Avg. Total Cost
LQR (true model)	18.03 ± 1.61
LQR (inaccurate model)	96,191 ± 21,327
LQR (GP model)	96,712 ± 21,315
Open Loop	67,664 ± 13,585
Multi-model LQR	33.51 ± 4.38
Hand-tuned Switching Control	73.91 ± 13.74

Fig. 2. Average total cost, with 95% confidence intervals, for different cart-pole control methods.

A. Cart-pole Task

To easily convey the intuition of our algorithm, and also to provide a publicly available implementation, we evaluated our algorithm on a simple cart-pole task (where the pole is allowed to freely swing entirely around the cart). The cart-pole is a well-studied dynamical system, and in particular we consider the control task of balancing an upright cart-pole system, swinging the pole around a complete revolution while moving the cart, and then balancing upright once more. Although the intuition behind this example is simple, there are indeed quite a few implementation details, and due to space constraints we defer most of this discussion to the source code, available at: `http://ai.stanford.edu/~kolter/icra10`.

The basic idea of the cart-pole task for our setting is as follows. First, we provide our algorithm with an inaccurate model of the dynamical system; the model uses a linear function of state features and control to predict the next state, but the features are insufficient to fully predict the cart-pole dynamics. As a result the model is fairly accurate in the upright cart-pole regions, but much less accurate during the swing phase. In addition, we provide our algorithm with a single desired trajectory (the target states and a sequence of controls that will realize this trajectory under a zero-noise system). Because we are focused on evaluating the algorithm without consideration for the method used to estimate the variances, for this domain we estimate the variances exactly using the true simulation model and sampling.

Figure 1 shows the system performance using three different methods: 1) fully "open loop" control, which just replays the entire sequence of controls in the desired trajectory, 2) running LQR using only the inaccurate model, and 3) using the Multi-model LQR algorithm with both the inaccurate model and the trajectory. As can be seen, both pure open-loop and LQR with the inaccurate model fail to control the system, but Multi-model LQR is able to reliably accomplish the control task. Figure 2 similarly shows the average total costs achieved by each of the methods, averaged over 100 runs. Although open loop and inaccurate LQR fail at different points, they are never able to successfully swing and balance the pole, while Multi-model LQR performs comparably to LQR using the actual model of the system dynamics by naturally interpolating between the two methods in order to best control the system. In the figure we also show the error of a hand-tuned switching policy that switches between purely model-based and open-loop control; despite exhaustive search to find the optimal switch points, the method still performs worse here than Multi-model LQR.

In Figure 2, we also compare to a Gaussian Process (GP) model (see, e.g. [17] for detailed information about Gaussian Processes) that attempted to learn a better dynamics model by treating the inaccurate model as the "prior" and updating this model using state transitions from the desired trajectory. However, the resulting model performs no better than LQR with the inaccurate model. We emphasize that we are *not* suggesting that Gaussian processes cannot model this dynamical system — they can indeed do so quite easily given the proper training data. Rather, this shows that the desired trajectory alone is not sufficient to improve a GP model, whereas the Multi-model LQR algorithm can perform well using only these two inputs; this is an intuitive result, since observing only a single trajectory says very little about the state and control *derivatives* along the trajectory, which are ultimately necessary for good fully LQR-based control. Indeed, we have been unable to develop any other controller based only on the inaccurate model and the desired trajectory that performs as well as Multi-model LQR.

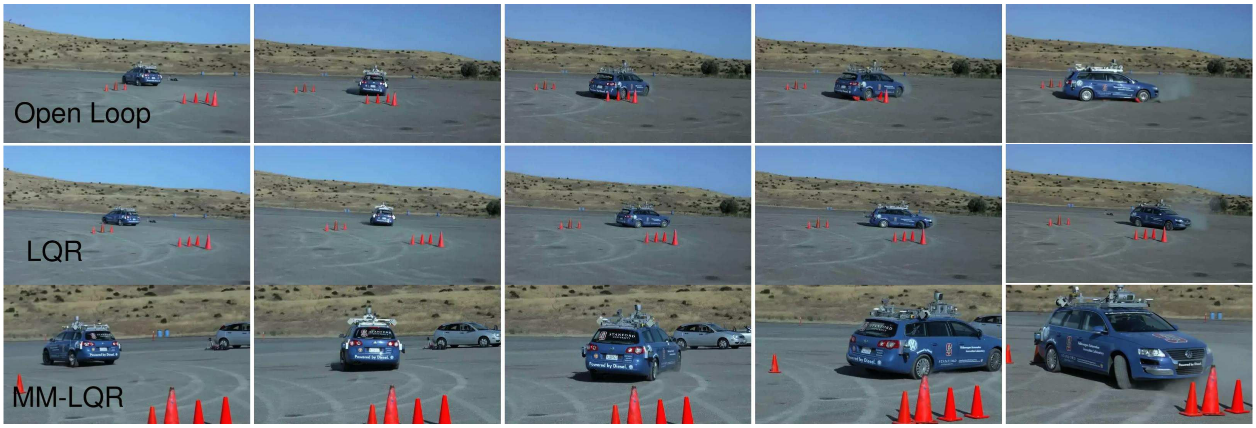


Fig. 3. Snapshots of the car attempting to slide into the parking spot using (top) open-loop control, (middle) pure LQR control, and (bottom) Multi-model LQR control. The desired trajectory in all cases is to slide in between the cones.

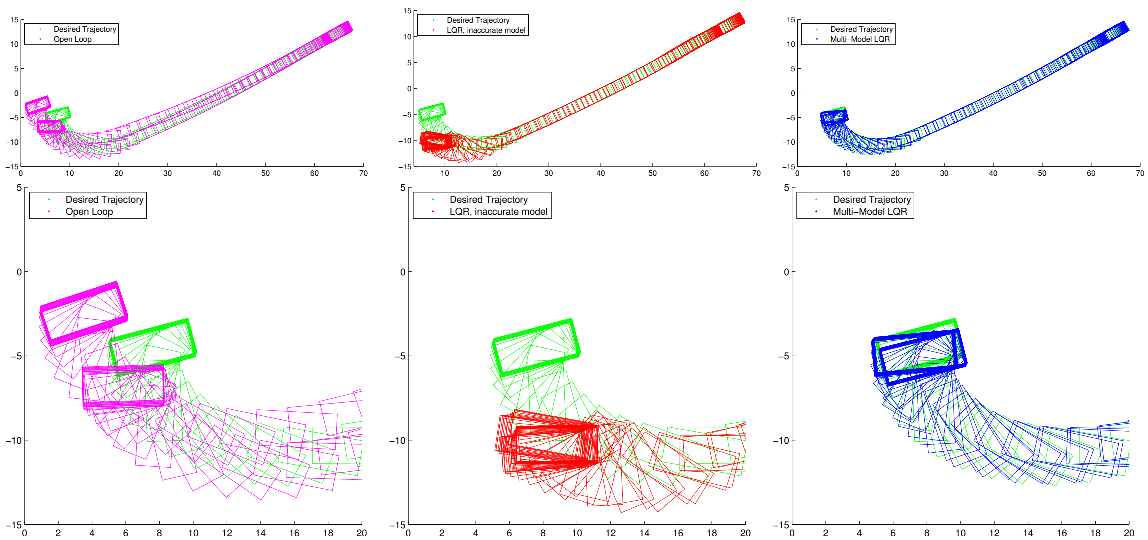


Fig. 4. Plots of the desired and actual trajectories followed by the car under (left) open-loop control, (middle) pure LQR control, and (bottom) Multi-model LQR control. Bottom plots show a zoomed-in view of the final location.

B. Extreme Autonomous Driving

Finally, in this section we present our chief applied result of the paper, an application of the algorithm to the task of extreme autonomous driving: accurately sliding a car sideways into a narrow parking space. Concurrent to this work, we have spent a great deal of time developing an LQR-based controller for "normal" driving on the car, based on a non-linear model of the car learned entirely from data; this controller is capable of robust, accurate forward driving at speeds up to 70 mph, and in reverse at speeds up to 30 mph. However, despite significant effort we were unable to successfully apply the fully LQR-based approach to the task of autonomous sliding, which was one of the main motivations for this current work.

Briefly, our experimental process for the car sliding task was as follows. We provided to the system two elements; first we learned a driving model for "normal" driving, learned with linear regression and feature selection, built using about 2 minutes of driving data. In addition, we provided the algorithm a single example of a human driver executing a sideways sliding maneuver; the human driver was making no attempt to place the car accurately, but rather simply putting the car into an extreme slide and seeing where it wound up. This demonstration was then treated as the target trajectory, and the goal of the various algorithms was to accurately follow the same trajectory, with cones placed on the ground to mark the supposed location of nearby cars.

For this domain, we learned domain parameters needed by the Multi-model LQR algorithm (in particular, the covariance terms for the inaccurate and open-loop models, plus the ρ term) from data. To reduce the complexity of learning the model variances, we estimated the covariance terms as follows: for the inaccurate model we estimated a time-varying (but state and control independent) estimate of the variance by computing the error of the model's predictions for each point along the trajectory, i.e.,

$$e_t = (s_{t+1} - f_1(s_t, u_t))(s_{t+1} - f_1(s_t, u_t))^T$$

then averaged these (rank-one) matrices over a small time window to compute the covariance for time t. For the open-loop trajectory model, we learned a state and control dependent (but not time dependent) estimate of the covariance of the form $\Sigma_2(s_t, u_t) = \left(w_1\|\delta u_t\|^2 + w_2\|\delta s_t\|^2 + w_3\right) I$, where we learned the parameters $w_1, w_2, w_3 > 0$ via least-squares; this model captures the intuition that the variance of the open-loop model increases for points that are farther from the desired trajectory. Finally we selected $\rho = 1$ due to the fact that the system rarely demonstrates extremely unstable behavior, even during the slide.

Figure 3 shows snapshots of the car attempting to execute the maneuver under the three methods of control: open-loop, pure LQR, and our Multi-model LQR approach integrating both the inaccurate model and the trajectory. Videos of the different slides are included in the video accompanying this paper. It is easy to understand why each of the methods perform as they did. Purely open-loop control actually does perform a reasonable slide, but since it takes some distance for the car to build up enough speed to slide, the trajectory diverges significantly from the desired trajectory during this time, and the slide slams into the cones. Pure LQR control, on the other hand, is able to accurately track the trajectory during the backward driving phase, but is hopeless when the car begins sliding: in this regime, the LQR model is extremely poor, to the point that the car executes completely different behavior while trying to "correct" small errors during the slide. In contrast, the Multi-model LQR algorithm is able to capture the best features of both approaches, resulting in an algorithm that can accurately slide the car into the desired location. In particular, while the car operates in the "normal" driving regime, Multi-model LQR is able to use its simple dynamics model to accurately control the car along the trajectory, even in the presence of slight stochasticity or a poor initial state. However, when the car transitions to the sliding regime, the algorithm realizes that the simple dynamics model is no longer accurate, and since it is still very close to the target trajectory, it largely executes the open-loop slides controls, thereby accurately following the desired slide. Figure 4 shows another visualization of the car for the different methods. As this figure emphasizes, the Multi-model LQR algorithm is both accurate and repeatable on this task: in the trajectories shown in the figure, the final car location is about two feet from its desired location.

VI. CONCLUSION

In this paper we presented a control algorithm that probabilistically combines multiple models in order to integrate knowledge both from inaccurate models of the system and from observed trajectories. The resulting controllers naturally trade off between closed-loop and open-loop control in an optimal manner, without the need to hand-tune a switching controller. We applied the algorithm to the challenging task of autonomously sliding a car sideways into a parking spot, and show that we can reliably achieve state-of-the-art performance in terms of accurately controlling such a vehicle in this extreme maneuver.

ACKNOWLEDGMENTS

We thank the Stanford Racing Team, Volkswagen ERL, and Boeing for support with the car. J. Zico Kolter is supported by an NSF Graduate Research Fellowship.

REFERENCES

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using innaccurate models in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2006.

[2] Brian D. O. Anderson and John B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, 1989.

[3] Christoper G. Atkeson and Stefan Schaal. Learning tasks from a single demonstration. In *Proceedings of the International Conference on Robotics and Automation*, 1997.

[4] Christopher G. Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of the International Conference on Robotics and Automation*, 1997.

[5] Stefan Schaal Christopher G. Atkeson, Andrew W. Moore. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113, 1997.

[6] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from muliple demonstrations. In *Proceedings of the International Conference on Machine Learning*, 2008.

[7] Kenji Doya, Kazuyuki Samejima, Ken ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural Computation*, pages 1347–1369, 2002.

[8] Chris Gerdes. Personal communication.

[9] Eric Hansen, Andrew Barto, and Shlomo Zilberstein. Reinforcement learning for mixed open-loop and closed-loop control. In *Neural Information Processing Systems*, 1996.

[10] Jessica K. Hodgins and Marc H. Raibert. Biped gymnastics. *International Journal of Robotics Research*, 9(2):115–128, 1990.

[11] Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo, and Sebastian Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *Proceedings of the 26th American Control Conference*, 2007.

[12] Yung-Hsiang Judy Hsu and J. Christian Gerdes. Stabilization of a steer-by-wire vehicle at the limits of handling using feedback linearization. In *Proceedings of the 2005 ASME International Mechanical Engineering Congress and Exposition*, 2005.

[13] Jonathan Ko, Daniel J. Klein, Dieter Fox, and Dirk Hhnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proceedings of the International Conference on Robotics and Automation*, 2007.

[14] Roderick Murray-Smith and Tor Arne Johansen. *Multiple Model Approaches to Modelling and Control*. Taylor and Francis, 1997.

[15] Kumpati S. Narendra and Jeyendran Balakrishnan. Adaptive control using multiple models. *IEEE Transactions on Automatic Control*, pages 171–187, 1997.

[16] Jan Peters and Stefan Schaal. Learning motor primatives with reinforcement learning. In *Proceedings of the 11th Joint Symposium on Neural Computation.*, 2004.

[17] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[18] Stefan Schaal. Nonparametric regression for learning. In *Proceedings of the Conference on Adaptive Behavior and Learning*, 1994.

[19] Kevin D. Schott and B. Wayne Bequette. Multiple model adaptive control. In *Multiple Model Approaches to Modelling and Control*. 1997.

[20] Martin Stolle and Christopher G. Atkeson. Policies based on trajectory libraries. In *Proceedings of the International Conference on Robotics and Automation*, 2006.

[21] Martin Stolle, Hanns Tappeiner, Joel Chestnutt, and Christopher G. Atkeson. Transfer of policies based on trajectory libraries. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2007.

[22] Yuval Tassa, Tom Erez, and William Smart. Receding horizon differential dynamic programming. In *Neural Information Processing Systems 20*, 2007.

[23] Claire Tomlin. Maneuver design using reachability algorithms, with applications to STARMAC flight. Presentation at RSS Workshop: Autonomous Flying vehicles - Fundamentals and Applications.

[24] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17:2602–2634, 2005.