Multiple Model Robust Dynamic Programming

Eric C. Whitman and Christopher G. Atkeson

Abstract— **Modeling error is a common problem for model-based control techniques. We present multiple model dynamic programming (MMDP) as a method to generate controllers that are robust to modeling error. Our method generates controllers that are approximately optimal for a collection of models, thereby forcing the controller to be less model-dependent. We compare MMDP to stochastic dynamic programming, minimax dynamic programming, and a baseline implementation of dynamic programming on the test problem of pendulum swing-up. We simulate modeling error by varying model parameters.**

I. INTRODUCTION

Model based optimal control is a powerful tool that suffers from a major difficulty: it requires a model of the system. Developing such a model can be a difficult task. Even if you can find a good model structure, parameter measurement and sensor calibration can be inaccurate. Even if initially accurate, the model may change over time due to wear, temperature, weather, contact condition, or any number of unforseen influences. Additionally, you may wish to develop a single controller for a large number of instances of a manufactured product which have variations between them due to manufacturing inconsistencies or unequal change over time. In all of these cases, the controller must cope with a model that is not identical to the true system.

Optimal control methods can often be very sensitive to even small modeling errors. Optimizers are very good at finding and exploiting any possible advantage, even if that advantage is a modeling error. Optimal paths often lie along constraints, so even small errors can make them infeasible [1]. Methods that plan a long way into the future may rely on the first portion of their plan working precisely in order for the latter portion to perform well.

Most methods that aim to reduce the sensitivity of optimal control algorithms to modeling error fall into one of two general categories: (i) minimizing an expected cost over possible outcomes or (ii) minimax formulations that minimize the maximum cost (worst case) of possible outcomes.

For expected cost approaches, a distribution of possible outcomes is required. It can be either a discrete list of possibilities (with associated probabilities) or a continuous distribution. The distribution can take many forms, ranging from additive output noise to sets of models. Some examples include additive or multiplicative state or control process noise on top of the nominal dynamics [2] [3], random variables in the process model [4], arbitrary state-dependent stochastic transition functions [5], and Markov decision processes (MDP) with known or even unknown transition probabilities [6].

Minimax methods are concerned with minimizing the worst case cost, so the probabilities of the potential outcomes are irrelevant and unnecessary. Again, there exists a large variety of ways to describe the set of possible outcomes. An H_∞ controller for linear systems can be found by solving a Riccati equation [7]. These controllers are robust to bounded input and process noise. Some examples for nonlinear systems include additive disturbances [8], discrete sets of models [9], and continuous sets of models [10]. Minimax formulations are generally able to provide stronger theoretical guarantees of robustness because they deal with worst case scenarios.

Most robust control algorithms are modifications of existing algorithms for deterministic problems. Differential Dynamic Programming (DDP) [8] [2] [3] and Model Predictive Control (MPC) [11] are popular techniques based on optimizing trajectories. Dynamic Programming (DP) is a class of algorithms that rely on the observation that any portion of an optimal trajectory is itself optimal. This leads to a situation where efficient computation can be achieved by reusing the solution to overlapping subproblems [12] [13] [14]. Most versions of DP produce control policies that are valid for large regions of the state space.

A popular approach to robust control law design is to optimize a policy by evaluating its performance in simulation on a distribution of possible models [9], [15], [16], [17], [18], [19], [20], [21], [22]. In this paper, we present Multiple Model Dynamic Programming (MMDP), which attempts to make a baseline implementation of DP more robust to modeling error by taking the expectation over a discrete set of multiple models. Importantly, we are not taking the expectation over a different model being randomly selected at each timestep; instead, we take the expectation over a single model being randomly selected once and used for all time.

The remainder of this paper is organized as follows: In

E. C. Whitman and C.G. Atkeson are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA. ewhitman@cmu.edu, cga@cmu.edu

This material is based upon work supported in part by the US National Science Foundation (ECCS-0824077 and IIS-0964581), the DARPA M3 program, and a National Science Foundation Graduate Fellowship to Eric Whitman (grant number 0750271).

Section II, we describe a test problem and a means of evaluating robustness to modeling error. In Section III, we describe the baseline DP algorithm. In Sections IV and V, we describe and present results for stochastic and minimax versions of DP. In Section VI, we describe and present results for MMDP. In Section VII, we discuss our algorithms and results, and we make concluding remarks in Section VIII.

II. TEST PROBLEM

A. Pendulum Swing-Up

We will demonstrate our algorithm on the problem of inverted pendulum swing-up. The controller must apply torques to a rigid pendulum in order to raise it to the inverted position and maintain it there. This is a good test problem because it involves both a dynamic travel component (getting to the inverted position) and a regulation component (remaining at the inverted position once there). The inverted pendulum is also one of the simplest nonlinear systems and a system about which we have good physical intuition, which makes it easier to interpret our results. The policy can also be severely limited by action-space constraints with it still remaining possible to achieve the goal from anywhere in the state space.

The pendulum is a second order system with one degree of freedom, so it has a 2-dimensional state space, $\mathbf{x} = \{\theta, \dot{\theta}\}$, where θ is the pendulum angle ($\theta = 0$ defined as upright), and the dot indicates a derivative with respect to time. It has a one-dimensional action space, $\mathbf{u} = \{\tau\}$, where τ is the control torque. The pendulum dynamics are given by

$$\ddot{\theta} = \frac{mLg\sin(\theta) + \tau}{mL^2}, \tag{1}$$

where m is the mass, L is the length, and $g = 9.81\,\mathrm{m/s^2}$ is the acceleration due to gravity. For the nominal system (we will perturb it later) the mass is 1 kg and the length is 1 m. We also constrain the control torque to $|\tau| \leq 1.5\,\mathrm{Nm}$, which requires the system to swing back and forth multiple times before reaching the goal.

The swing-up task is formally defined as minimizing the total cost, C, given by the integral,

$$C = \int_0^\infty L(\mathbf{x}, \mathbf{u})\, dt, \tag{2}$$

of the one step cost function,

$$L(\mathbf{x}, \mathbf{u}) = \theta^2 + 0.5\dot{\theta}^2 + \tau^2. \tag{3}$$

B. Tests of Robustness

The goal of this paper is to produce steady state controllers, $\mathbf{u} = \pi(\mathbf{x})$, that are as robust as possible to modeling error. By this, we mean that we wish the controllers to be effective on a set of modified pendulum systems. We measure the effectiveness of a particular policy, $\pi(\mathbf{x})$, on a particular model by starting the system at $\mathbf{x} = \{\pi, 0\}$ (down), simulating it forward, and measuring (2). If the system does not reach the goal ($\mathbf{x} = \{0, 0\}$) and remain there within 60 seconds of simulation, we assume it never will and assign an infinite cost.

In order to test the robustness of our policies, we will modify the nominal model in four ways:

1) Varying the mass, m.
2) Varying the length, L.
3) Adding varying amounts of viscosity, ν. Equation (1) becomes
$$\ddot{\theta} = \frac{mLg\sin(\theta) + \tau - \nu\dot{\theta}}{mL^2}. \tag{4}$$
4) Misaligning the goal with gravitational up, representing an off-center mass or sensor miscalibration. An offset, θ_0 is added to (1), changing it to
$$\ddot{\theta} = \frac{mLg\sin(\theta - \theta_0) + \tau}{mL^2}. \tag{5}$$

For the final test, there is no state-action pair that results in both no acceleration and no cost, L, so the cost given by (2) will always be infinite. To accomodate this, we consider the task complete and stop integrating (2) after the system has remained near the goal for 15 seconds.

III. DYNAMIC PROGRAMMING ALGORITHM

A major disadvantage of trajectory optimization based control techniques is that they are typically only effective in a narrow tube around the planned trajectory. Even with a locally stabilizing controller, large disturbances such as modeling error can push the system far from the nominal trajectory, resulting in poor performance. They are also generally unable to support qualitative changes in strategy such as an extra swing of the pendulum if more energy needs to be added.

Dynamic Programming (DP) [12] [13] [14] avoids these disadvantages by generating optimal controllers for a large region of state space. We use a version of modified policy iteration that generates time-invariant policies for systems with continuous state and action spaces. We represent the continuous state space by dividing it into a grid. Since angles are topologically circular, we can represent the entire θ direction with a finite grid. However, we must bound the $\dot{\theta}$ dimension to the somewhat arbitrarily selected range of $\pm 10\,\mathrm{rad/s}$. A grid resolution of at least about {50, 100} is necessary to achieve swing-up. All experiments in this paper were performed with a resolution of {400, 900} to give a good tradeoff between computation time (5 minutes on a PC using a 6-core processor with a clock speed of 3.33 GHz) and policy quality. This gives a resolution of {0.016 radians, 0.022 radians/second} and a grid of 360,000 discretized states.

Each grid point stores the current value estimate, $V(\mathbf{x})$, and the current best control vector, $\mathbf{u} = \pi(\mathbf{x})$ ($\mathbf{u} = \tau$ for the pendulum). During an iteration, for each grid point, we use the Bellman equation [12],

$$V(\mathbf{x}) = \min_{\mathbf{u}\in\{\mathbf{u}_0,\mathbf{u}_r\}} L(\mathbf{x},\mathbf{u}) + V(f(\mathbf{x},\mathbf{u})), \tag{6}$$

to update the value function where $\mathbf{x}_{k+1} = f(\mathbf{x}_k,\mathbf{u}_k)$ is the discrete time system dynamics, $\mathbf{u}_0$ is the current best action, and $\mathbf{u}_r$ is a random action [23] drawn from the legal range: [-1.5 Nm, 1.5 Nm]. The policy is then updated to the corresponding action.

If the final state, $f(\mathbf{x},\mathbf{u})$, is outside of the grid ($|\dot{\theta}| > 10$ rad/s), we assign an infinite cost, $V(f(\mathbf{x},\mathbf{u})) = \infty$. Otherwise, we use multilinear interpolation [24] on the previous iteration's estimate of $V(\mathbf{x})$ to estimate the terminal value. To ensure that we get a good estimate of the value function, if the final state is in a cell with the originating grid point as a corner, we simulate forward additional time steps (with the same $\mathbf{u}$, adding the appropriate one step cost, $L(\mathbf{x}_k,\mathbf{u})$, until we enter a cell not bordering the originating grid point. Additionally, if we enter a cell where some corners have infinite value and some corners have finite value, we continue to simulate until we reach a cell where either all corners have infinite value or all corners have finite value.

It takes about 500 iterations to generate a controller that can achieve swing-up. We run all of our experiments for 5000 iterations to allow the policy and value function to converge very closely to the optimum.

Note that this formulation requires the discrete time form of the pendulum dynamics, which we obtain by integrating (1):

$$\begin{bmatrix} \theta_{k+1} \\ \dot{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} \theta_k + \dot{\theta}_k T + 0.5\ddot{\theta}_k T^2 \\ \dot{\theta}_k + \ddot{\theta}_k T \end{bmatrix} \tag{7}$$

where $T = 0.001$ seconds is the time step, and $\ddot{\theta}$ is determined by (1).

The computational and memory costs of DP are linear in the number of grid points, and therefore exponential in the number of state space dimensions. This limits DP to problems with low dimensional state spaces (about 5 dimensions on a modern desktop computer), though in some cases it is possible to treat higher dimensional problems by breaking them into multiple lower dimensional problems and coordinating the policies [25].

IV. MODELING ERROR AS NOISE

Much work has been done on generating optimal controllers for stochastic systems by optimizing the expected value of the cost function. Modeling error is not actually stochastic, but these techniques can be used to increase robustness to modeling error by generally increasing robustness to unexpected dynamics.

For DP, we do not have an analytical expression for the value function, so we take the expectation of the stochastic dynamics by sampling from it, turning (6) into

$$V(\mathbf{x}) = \min_{\mathbf{u}\in\{\mathbf{u}_0,\mathbf{u}_r\}} \sum_{i=1}^{N} (L(\mathbf{x},g_i(\mathbf{u})) + V(f_i(\mathbf{x},g_i(\mathbf{u}))))p_i \tag{8}$$

$$\sum_{i=1}^{N} p_i = 1 \tag{9}$$

where g_i is an instantiation of the control noise, f_i is an instantiation of the process noise, and p_i is the probability of the ith sample. To avoid optimizing over the instantiations of the random noise, we must fix the g_i's, f_i's, and p_i's for all iterations, which essentially forces us to approximate the noise distribution as a sum of delta functions. In general, this can require a large number of samples (and correspondingly large computational cost) to adequately model the distribution.

For the case of additive noise, however, the Central Limit Theorem tells us that regardless of what distribution we use, after a large number of time steps, the total noise will have an approximately Gaussian distribution. This frees us to use a small number of samples and rely upon summation over many time steps to produce a fuller distribution. For our swing-up example, we used no process noise ($f_i = f$), and additive control noise ($p_1 = p_2 = 0.5$, $g_1(\tau) = \tau - \Delta$ and $g_2(\tau) = \tau + \Delta$ where Δ is a parameter that controls the size of the noise). Note that additive control noise is identical to additive acceleration noise because $\partial\ddot{\theta}/\partial\tau = 1/mL^2$, which is constant.

Fig. 1 shows robustness results for stochastic DP compared to the baseline DP implementation.

V. MINIMAX FORMULATION

The unexpected dynamics introduced by modeling error are poorly represented by random noise because they are not independent at every time step. The errors introduced by modeling error often consistently push the system in the same direction, a situation which can get ignored by noise-based formulations as being extremely low probability. Another possibility for handling noise is to use a minimax formulation. Rather than minimizing the expected value, minimax algorithms attempt to minimize the maximum value or worst case scenario. In [8], a minimax version of DDP is developed and demonstrated on a walking robot.

We implement minimax DP much like stochastic DP, but instead of taking the expected value, we take the maximum, so the Bellman equation becomes

$$V(\mathbf{x}) = \min_{\mathbf{u}\in\{\mathbf{u}_0,\mathbf{u}_r\}} \max_i L(\mathbf{x},g_i(\mathbf{u})) + V(f_i(\mathbf{x},g_i(\mathbf{u}))). \tag{10}$$

We no longer require the Central Limit Theorem to approximate the full distribution. Instead, we rely upon the assumption that the worst case disturbance will be an extreme disturbance: either the maximum push in one direction or

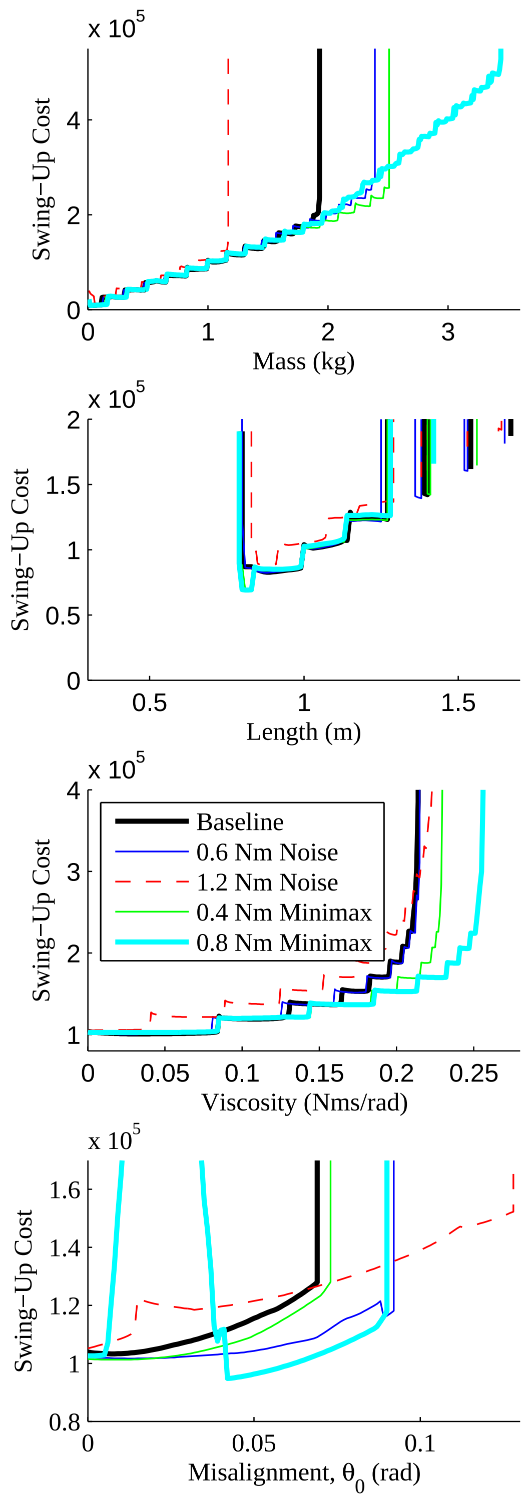

Fig. 1. Robustness to model variations of mass, length, viscosity, and gravity misalignment. Results are shown for the baseline DP (Section III), DP with small and large amounts of noise (Section IV), and DP with small and large minimax perturbations (Section V).

the maximum push in the other direction. This assumption is an approximation for nonlinear systems, but in the case of short time steps, additive disturbances, and affine in controls, it is very nearly accurate. This again allows us to keep the computation managable by using only two samples. For the pendulum example, we again use no process noise ($f_i = f$), and additive control noise ($g_1(\tau) = \tau - \Delta$ and $g_2(\tau) = \tau + \Delta$).

Fig. 1 also shows results for the minimax version of DP. Both methods generally increase robustness to modeling error with increased magnitude of the disturbance until they reach a point of diminishing returns. This point can be quite abrupt, and it can be different for different types of error. For each method, we show results for two different disturbance sizes, one slightly before the point of diminishing returns on any of the tests, and another where performance has decreased on some tests but not on others. As expected, the minimax formulation generally outperforms the noise formulation, but it does well on different tests. Qualitatively, these trends can be summarized by noting that the minimax formulation tends to do better when pushing harder is necessary to overcome the unexpected modeling error (increased mass and viscosity), but the noise formulation deals better with small errors at inopportune times (misaligning the goal with gravitational up). Discontinuities in the cost result from changes in strategy: using more or fewer back and forth swings to reach the upright position.

VI. MULTIPLE MODEL DYNAMIC PROGRAMMING

If the true system is deterministic, but different from your model, neither random nor worst-case disturbances are a good description of the true error. We are interested in the situation where there exists one true, deterministic model, but we do not know what it is. We approach this by generating a single policy, $\mathbf{u} = \pi(\mathbf{x})$, that is optimized to perform well on a set of N candidate models. We denote the dynamics of the ith model as $\mathbf{x}_{k+1} = f_i(\mathbf{x}_k, \mathbf{u}_k)$. This method only directly optimizes performance for the specific models it is given, but we have a reasonable expectation of good performance for a range of model space around each of the candidate models. If we know something about the type of modeling error we expect, we can choose candidate models that cover the range of expected models. Formally, we wish to find the policy, π, that minimizes the total cost criterion,

$$C = \sum_{i=1}^{N} \sum_{x_0} \sum_{t=0}^{\infty} L(\mathbf{x}_t, \pi(\mathbf{x}_t)). \qquad (11)$$

The innermost sum is trajectory cost and is the discrete form of (2). The middle sum is over trajectories started at each grid point. The outermost sum is over all N models.

For a single model, we have the convenient property that a single policy is optimal for all start states. Unfortunately, this property does not hold for the multiple model case. To show this, consider a policy that is optimal for a single start state. If we have a single start state, a state-indexed policy can control each of the models independently because they will travel through different states. We can therefore construct an optimal policy for multiple models with a single start state by copying the optimal policy along the optimal trajectory from each of the individual models' optimal policies. The optimal action at a given state will therefore depend on which model passes through it (and which corresponding optimal policy we must copy from). Since which model passes through a given point depends upon the start state, the optimal policy must also depend upon the start state.

We modify our DP algorithm to approximately minimize (11) by maintaining individual value functions, $V_i(\mathbf{x})$, for each of the N dynamic models (but only a single policy). To update a grid point, we pick the action by adding the value from each of the models:

$$\pi(\mathbf{x}) = \arg \min_{\mathbf{u} \in \{\mathbf{u}_0, \mathbf{u}_r\}} L(\mathbf{x}, \mathbf{u}) + \sum_{i=1}^{N} p_i V_i(f_i(\mathbf{x}, \mathbf{u})), \tag{12}$$

where p_i is again the probability of each model. We then update each value function individually according to

$$V_i(\mathbf{x}) = L(\mathbf{x}, \pi(\mathbf{x})) + V_i(f_i(\mathbf{x}, \pi(\mathbf{x}))). \tag{13}$$

As in the baseline implementation, we do many iterations of updating every point in the grid this way. This gives us a policy, $\pi(\mathbf{x})$, that is approximately optimized to perform well on all of the models, and a value function, $V_i(\mathbf{x})$, for each of the models given this policy.

To analyze our update rule with respect to (11), we note that the value function, V, is exactly equal to the innermost sum, so we can rewrite (11) as

$$C = \sum_{i=1}^{N} \sum_{x_0} p_i V_i(x_0). \tag{14}$$

We define $d_i(\mathbf{x})$ as the number of states for which if you start a trajectory there it will pass through state $\mathbf{x}$ for model i (and a given policy). If we make a single change to our policy at state $\mathbf{x}$, it will change the value at that state as well as at any state whose trajectory passes through it. All other values will remain the same. We can write the resulting change in total cost as

$$\Delta C = \sum_{i=1}^{N} p_i \left[\Delta V_i(\mathbf{x}) + d_i(\mathbf{x}) \Delta V_i(\mathbf{x})\right] \tag{15}$$

where ΔV_i is the new value minus the old value for model i. To ensure that this is decreasing, we must assume that

$$d_i(\mathbf{x}) = d_j(\mathbf{x}) \forall i, j, \mathbf{x} \tag{16}$$

which means that all models have an equal chance (given a random start state) to get into any given state. If this were not true, it would make sense to focus on minimizing the value

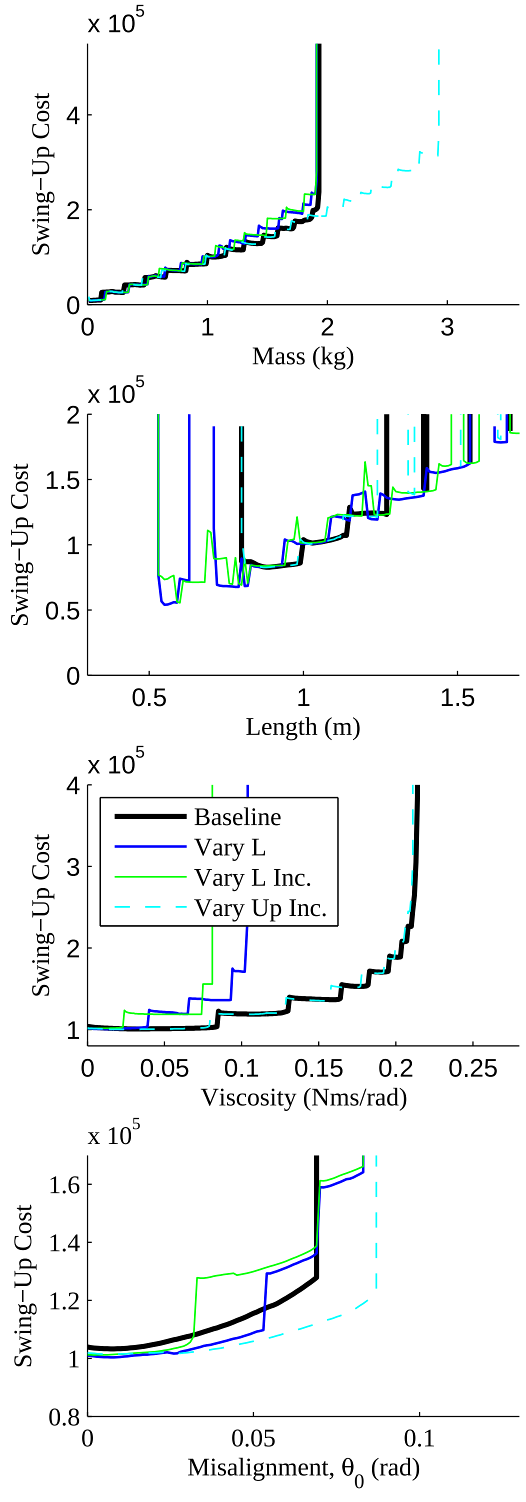

Fig. 2. Robustness to model variations of mass, length, viscosity, and gravity misalignment. Results are shown for the baseline DP, MMDP with models of various length added all at once and added incrementally, and MMDP with models of various θ_0 added incrementally.

for the models that are more likely to end up in that state, which our algorithm does not do. This assumption will not generally be true, but for many problems where the candidate models are similar, it will be approximately true.

Given (16), (15) simplifies to

$$\Delta C = (1 + d(\mathbf{x})) \sum_{i=1}^{N} p_i \Delta V_i(\mathbf{x}). \tag{17}$$

Given our update rule, we know that $\sum_{i=1}^{N} p_i \Delta V_i(\mathbf{x}) \leq 0$, so we have $\Delta C \leq 0$.

In practice, we generally have a nominal (or best guess) dynamic system and wish to increase the range of model space surrounding it for which the policy performs well. Either because the convergence rate is very slow or because of nonidealities such as (16) not holding, MMDP sometimes finds solutions that work well for the extreme cases of model, but not for the nominal case. A less drastic problem is gaps in the region of model space for which the policy succeeds.

To cope with this, we add models incrementally. This technique is similar to shaping, which was first invented in the field of psychology for training animals [26], but has since been used for machine learning and optimization [27]. Shaping is used to train animals to do complex tasks by first training them to do something simple, then training them to do successively closer approximations of the desired behavior. We start off by running DP on just the nominal model. We then add dynamic models that are very similar to the nominal case, and run a new optimization starting with the previously solved for policy and the previously solved for value function. We can then continue to add additional models progressively further from the nominal dynamics and rerunning the optimization. When new models are added, the value function can be initialized by copying the value function of the most similar model already in the training set.

Fig. 2 shows results for the MMDP. We took 21 pendulum models ($p_i = 1/N$) with lengths, L, ranging from 0.5 m to 1.5 m in 0.05 m increments and ran MMDP on them. The result was a drastically improved robustness to variation in the length. Swing-up was achievable for nearly the entire 0.5 m to 1.5 m range, but with a gap centered around 0.65 m.

We can compare this to the result when we add the same 21 models incrementally. We start with the nominal model ($L = 1.0$ m). After doing 5000 iterations of DP, we add the two adjacent models ($L = 0.95$ m and $L = 1.05$ m), initializing their value functions from the nominal models and then do another 5000 iterations of DP. We then add another 9 pairs of models, initializing each from the previous pair, and doing 5000 iterations of DP between each addition. By shaping the policy in this way and gradually building its robustness, we are able to cover the same range of modeling error, but without the gap centered at $L = 0.65$ m.

We also use the same method to target improving robustness to the gravitational angle by incrementally adding pairs of models with positive and negative θ_0's further and further from 0. The result is an improved robustness to varying this angle while maintaining a low cost. On the other hand, optimizing on this set of models does not improve performance with respect to changing the pendulum length. This illustrates a downside of MMDP: it often does not provide improved robustness for model variations other than the type specifically targeted.

VII. DISCUSSION AND FUTURE WORK

Because (16) will generally not hold, in practice MMDP is a heuristic method. The more qualitatively similar the models, the closer (16) will be to holding, and the closer MMDP will get to truly minimizing (11).

MMDP works well when you know something about the type of modeling error you have. It works best when you are able to provide it with models that span the range of possible systems. Attempts to generally increase robustness by providing models with multiple types of changes performed less well. This was at least partially due to the fact that in the higher dimensional model space, our sampling was much more sparse.

Another limitation of the MMDP approach is that it is computationally expensive to handle types of error that increase the dimensionality of the state space (e.g. time delays, filters, structural flexibility) even if the policy does not depend on these extra dimensions. MMDP must include these extra states in the grid and therefore pay the exponential computational and memory cost of the extra states. Methods that do not explicitly model the modeling error (such as the noise and minimax formulations) need not pay this cost. Methods that simulate entire trajectories (instead of single time steps as in DP) can avoid it as well because they need only fully explore the space that the controller depends on.

For a known, deterministic system, the optimal controller for any cost function where failure has an infinite cost will have the same basin of attraction, which is identical to the maximum feasible region. However, this says nothing about what happens when the system is different from what was expected. Robustness to modeling error can be very sensitive to the choice of cost function. Experiments support our intuition that in MMDP, the robustness to the type of model variation is relatively insensitive to choice of cost function because robustness is being directly optimized for. However, for types of modeling error other than what we were directly optimizing for, we see just as much sensitivity to cost function choice as we see in the stochastic and minimax formulations.

In [25], we presented a controller for walking applied to a simulated 3D biped based on coordinating multiple

DP policies. One of the main obstacles preventing us from applying this controller to our hydraulic, force-controlled humanoid robot [28] (and a major motivation for this work) is modeling error. We are currently attempting to increase the robustness of our controller to modeling error. In a preliminary attempt at this, we applied MMDP to the controller responsible for sagittal plane balance using three models with -3 cm, 0 cm, and 3 cm offsets to the touchdown location. We then measured how large a forward push we could apply to the walking simulation without it falling down. The result depended on the timing of the push, but averaging over all possible push timings, we achieved a 10% increase in the size of the survivable push using MMDP when compared to the baseline DP implementation. We hope to achieve a larger increase by using more than 3 models and experimenting with different types of model variation.

VIII. CONCLUSION

We present multiple model dynamic programming (MMDP), which uses multiple models to generate controllers which are robust to modeling error. We demonstrate the effectiveness of MMDP on the test problem of pendulum swing-up, and compare it to stochastic DP and minimax DP as well as a baseline implementation of DP. MMDP is able to improve the robustness to types of modeling error that the other two methods are ineffective against. However, it requires some knowledge of the type of modeling error to be expected and is less effective at generally increasing robustness to all types of modeling error simultaneously.

REFERENCES

[1] Yaman Arkun and George Stephanopoulos, "Studies in the synthesis of control structures for chemical processes: Part IV. design of steady-state optimizing control structures for chemical process units", *AIChE Journal*, vol. 26, pp. 975–991, 1980.

[2] Evangelos Theodorou, Yuval Tassa, and Emo Todorov, "Stochastic differential dynamic programming", in *Proceedings of American Control Conference*, 2010.

[3] David Andrew Kendrick, *Stochastic control for economic models*, McGraw-Hill, second edition, 2002.

[4] Marc C. Steinbach, "Robust process control by dynamic stochastic programming", *Proceedings in Applied Mathematics and Mechanics (PAMM)*, vol. 4, no. 1, pp. 11–14, 2004.

[5] Wee Chin Wong and Jay H. Lee, "Postdecision-state-based approximate dynamic programming for robust predictive control of constrained stochastic processes", *Industrial & Engineering Chemistry Research*, vol. 50, no. 3, pp. 1389–1399, 2011.

[6] Arnab Nilim, Laurent El Ghaoui, and Vu Duong, "Robust dynamic routing of aircraft under uncertainty", in *Proceedings of Digital Avionics Systems Conference*, 2002.

[7] Geir E. Dullerud and Fernando Paganini, *A course in robust control theory: A convex approach*, Texts in Applied Mathematics. Springer, New York, NY, 2000.

[8] Jun Morimoto, Garth Zeiglin, and Christopher G. Atkeson, "Minimax differential dynamic programming: Application to a biped walking robot", in *Proceedings of Intl. Conference on Intelligent Robots and Systems*, 2003.

[9] V. Azhmyakov, V.G. Boltyanski, and A.S. Poznyak, "On the dynamic programming approach to multi-model robust optimal control problems", *American Control Conference*, , no. 1, pp. 4468–4473, 2008.

[10] Moritz Diehl and Jakob Bjornberg, "Robust dynamic programming for min-max model predictive control of constrained uncertain systems", *Automatic Control, IEEE Transactions on*, vol. 49, no. 12, pp. 2253–2257, 2004.

[11] Alberto Bemporad and Manfred Morari, "Robust model predictive control: A survey", in *Robustness in identification and control*, vol. 245 of *Lecture Notes in Control and Information Sciences*, pp. 207–226. Springer Berlin / Heidelberg, 1999.

[12] Richard Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[13] Dimitri P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, 1995.

[14] Jennie Si, Andrew G. Barto, Warren B. Powell, and Don Wunsch, Eds., *Handbook of Learning and Approximate Dynamic Programming*, Wiley-IEEE Press, 2004.

[15] A. Varga, "Optimal output feedback control: a multi-model approach", in *IEEE International Symposium on Computer-Aided Control System Design*, 1996, pp. 327–332.

[16] H. T. Su and T. Samad, "Neuro-control design: Optimization aspects", in *Neural Systems For Control*, O. Omidvar and D. L. Elliott, Eds., pp. 259–288. Academic Press, 1997.

[17] Y. Piguet, U. Holmberg, and R. Longchamp, "A minimax approach for multi-objective controller design using multiple models", *International Journal of Control*, vol. 72, no. 7-8, pp. 716–726, 1999.

[18] D. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods", in *IEEE International Conference on Robotics and Automation*, 2001.

[19] R. H. Nyström, J. M. Böling, J. M. Ramstedt, H. T. Toivonen, and K. E. Häggblom, "Application of robust and multimodel control methods to an ill-conditioned distillation column", *Journal of Process Control*, vol. 12, pp. 39–53, 2002.

[20] F. J. Bejarano, A. Poznyak, and L. Fridman, "Min-max output integral sliding mode control for multiplant linear uncertain systems", in *American Control Conference*, 2007, pp. 5875–5880.

[21] M. McNaughton, "CASTRO: robust nonlinear trajectory optimization using multiple models", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 177–182.

[22] J. Shinar, V. Glizer, and V. Turetsky, "Solution of a linear pursuit-evasion game with variable structure and uncertain dynamics", in *Advances in Dynamic Game Theory - Numerical Methods, Algorithms, and Applications to Ecology and Economics*, S. Jorgensen, M. Quincampoix, and T. Vincent, Eds., vol. 9, pp. 195–222. Birkhauser, 2007.

[23] Christopher G. Atkeson, "Randomly sampling actions in dynamic programming", in *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

[24] Scott Davies, "Multidimensional triangulation and interpolation for reinforcement learning", in *NIPS*, 1996, pp. 1005–1011.

[25] Eric C. Whitman and Christopher G. Atkeson, "Control of instantaneously coupled systems applied to humanoid walking", in *Proceedings of the IEEE International Conference on Humanoid Robots*, 2010.

[26] Gail B Peterson, "A day of great illumination: B. F. Skinner's discovery of shaping", *Journal of the Experimental Analysis of Behavior*, vol. 82, pp. 317–328, 2004.

[27] George Konidaris and Andrew Barto, "Autonomous shaping: knowledge transfer in reinforcement learning", in *Proceedings of the 23rd International Conference on Machine learning*, 2006, ICML '06, pp. 489–496.

[28] Darrin C. Bentivegna, Christopher G. Atkeson, and Jung-Yup Kim, "Compliant control of a compliant humanoid joint", in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2007.