MIT Technical Report No. 582, June 2004

The Essential Dynamics Algorithm: Fast Policy Search In Continuous Worlds

Martin C. Martin
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
USA

Abstract

This paper presents a novel algorithm for learning in a class of stochastic Markov decision processes (MDPs) with continuous state and action spaces that trades speed for accuracy. The algorithm can be seen as a generalization of linear quadratic control to non-linear, non-regulation problems. A transform is presented of the stochastic MDP into a deterministic one which captures the essence of the original dynamics, in a sense made precise. In this transformed MDP, the calculation of values is greatly simplified. The online algorithm estimates the model of the transformed MDP and simultaneously does policy search against it. Bounds on the error of this approximation are proven, and experimental results are presented in both a bicycle riding domain and the control of a robot arm on a dynamic base, a 14 dimensional state space. The algorithm learns near optimal policies in orders of magnitude fewer interactions with the stochastic MDP, using less domain knowledge. Code is available on the project's web site.

1 INTRODUCTION

There is currently much interest in the problem of learning in stochastic Markov decision processes (MDPs) with continuous state and action spaces (Forbes & Andre, 2000; Strens & Moore, 2002). For such domains the value and Q-functions may be quite complicated and difficult to approximate, especially when the state or action spaces are of high dimension. In fact, Baxter and Bartlett (2000) demonstrated that policies derived from approximate value functions can fail arbitrarily badly. However, there may be relatively simple policies which perform well. This has lead to recent interest in *policy search* algorithms, in which the reinforcement signal is used to modify the policy directly (Ng & Jordan, 2000; Roy & Thrun, 2002; Strens & Moore, 2002).

For many problems, a positive reward is only achieved at the end of a task if the agent reaches a "goal" state. For complex problems, the probability that an initial, random policy would reach such a state could be vanishingly small. A widely used methodology to address this problem is *shaping* (Colombetti & Dorigo, 1994; Mataric, 1994; Ng et al., 1999; Randløv, 2000). Shaping is the introduction of small rewards to reward partial progress toward the goal. A shaping function eases the problem of backing up rewards, since actions are rewarded or punished sooner.

When a policy changes, estimating the resulting change in values can be difficult, requiring the new policy to interact with the MDP for many episodes. In this paper we introduce a method of transforming a stochastic MDP into a deterministic one. Under certain conditions on the original MDP, and given a shaping reward of the proper form, the deterministic MDP can be used to estimate the value of any policy with respect to the original MDP. This leads to an online algorithm for policy search: simultaneously estimate the parameters of a model for the transformed, deterministic MDP, and use this model to estimate both the value of a policy and the gradient of that value with respect to the policy parameters. Then, using these estimates, perform gradient descent search on the policy parameters. Since the transformation captures what is important about the original MDP for planning, we call our method the "essential dynamics" algorithm.

1.1 RELATIONSHIP TO ADAPTIVE CONTROL

A control law is a function from state to action, and is thus a policy. The study of adjusting the parameters of a control law in response to interaction with the system being controlled is known as *adaptive control* (Åström & Wittenmark, 1995). A *loss function* is the cost, summed over future timesteps, that is to be minimized, and is thus analogous to the (negative of the) value function. In control theory, model learning is called *system identification,* while *excitory inputs* are the most common form of exploration.

The essential dynamics algorithm can be seen as a generalization of the *linear quadratic regulator* (Dorato et al., 1995). Let x be a vector of the current state minus the desired state. In a linear quadratic regulator, the next state is modelled as a linear function of both state x and action u, plus some noise:

$$x(t+1) = Ax(t) + Bu(t) + Ce(t)$$

where A, B and C are matricies and e(t) is a vector of uncorrelated zero mean Gaussian random variables (white noise). If the loss function is

$$V = E[\sum_{t=0}^{T} x(t)'Qx(t) + u(t)'Ru(t)]$$

then the optimal control law is of the form $u(t) = -Kx(t)$. If A, B and C have been estimated then it is straightforward to compute the future $x(t)$ and V as a function of K and $x(0)$. V is quadratic in K, and the minimizing K is easily found.

It is worth remarking that sixty years after the foundation of the field, linear models and linear control laws are still the norm, and are used extensively in practical systems. They are used primarily in *regulator problems,* where the goal is to stay near to a given desired state. In generalizing to the entire state space, we thus take as our underlying assumption that the model is locally linear, that is, that its second derivative w.r.t. state is small. As a result, the policy may be non-linear as well. We define our value function to be a limited horizon sum of the shaping reward, which we restrict to be quadratic.

With arbitrary functions for our model and policy, it is no longer possible to algebraically minimize the value function. Instead, we perform gradient descent. That is, given some initial parameters for the policy, we can compute not only the expected state and reward over the next T timesteps, but the derivative of the those with respect to each parameter in the policy. We then move the parameters in the direction that increases reward. While we are no longer guaranteed a single local optima, the similarity to the control theory case suggests that, at least for policies that are "similar" to a linear combination of state, the value should be roughly quadratic in the policy parameters.

The next section gives an overview of the technique, developing the intuition behind it. In section 3 we describe the mathematical foundations of the algorithm, including bounds on the difference between values in the original and transformed MDPs. Section 4 describes an application of this technique to learning to ride a bicycle. The last section discusses these results, comparing them to previous work. On the bicycle riding task, given the simulator, the only domain knowledge needed is a shaping reward that decreases as lean angle increases, and as angle to goal increases. Compared to previous work on this problem, a near optimal policy is found in *dramatically* less simulated time, and with less domain knowledge. Section 5 briefly describes the control of a five degree of freedom robot arm atop a Segway base, where the state space has 14 continuous dimensions, and the action space 5 dimensions. The only domain knowledge needed is inverse kinematics (which, in this case, simply involves high school trigonometry). The resulting policy is able to move the wrist to any desired location and to level the wrist, throughout a wide range of its workspace.

2 OVERVIEW OF THE ESSENTIAL DYNAMICS ALGORITHM

In the essential dynamics algorithm we learn a model of how state evolves with time, and then use this model to compute the value of the current policy. In addition, if the policy and model are from a parameterized family, we can compute the gradient of the value with respect to the parameters.

In putting this plan into practice, one difficulty is that state transitions are stochastic, so that the *expected* cumulative reward must be computed. One way to compute this is to generate many trajectories and average over them, but this can be very time consuming. Instead we might be tempted to estimate only the mean of the state at each future time, and use the rewards associated with that. However, we can do better. If the reward is quadratic, the expected reward is particularly simple. Given knowledge of the state at time t, we can then talk about the distribution of possible states at some later time. Because the state is a vector of real numbers, the expected state is well defined. Given a distribution of states, let $\bar{s}$ denote the expected state. Then

$$\begin{aligned} E[r(S)] &= \int (a(s-\bar{s})^2 + b(s-\bar{s}) + c)P(s)ds \\ &= a\text{var}(s) + b(\bar{s}-\bar{s}) + c \\ &= a\text{var}(s) + c, \end{aligned} \tag{1}$$

where a, b & c depend on r and $\bar{s}$.

Thus, to calculate the expected reward, we don't need to know the full state distribution, but simply its mean and variance. Therefore, our model should describe how the mean and variance evolve over time. If the state transitions are "smooth," they can be approximated by a Taylor series. Let π be the current policy, and let $\mu_\pi(s)$ denote the expected state that results from taking action $\pi(s)$ in state s. If $\bar{s}_t$ denotes the mean state at time t, and σ_t^2 the variance, and if state transitions were deterministic, then to first order we would have

$$\begin{aligned} \bar{s}_{t+1} &\approx \mu_\pi(\bar{s}_t) \\ \sigma_{t+1}^2 &\approx \left(\frac{d\mu_\pi}{ds}(\bar{s}_t)\right)^2 \sigma_t^2. \end{aligned}$$

For stochastic state transitions, let $\nu_\pi(s)$ be the variance of the state that results from taking action $\pi(s)$ in state s. It turns out that the variance at the next time step is simply $\nu_\pi(s)$ plus the transformed variance from above, leading to

$$\begin{aligned} \bar{s}_{t+1} &\approx \mu_\pi(\bar{s}_t) \\ \sigma_{t+1}^2 &\approx \nu_\pi(\bar{s}_t) + \left(\frac{d\mu_\pi}{ds}(\bar{s}_t)\right)^2 \sigma_t^2. \end{aligned} \tag{2}$$

Thus, we learn estimates $\tilde{\mu}$ and $\tilde{\nu}$ of μ and ν respectively, use (2) to estimate the mean and variance of future states, and (1) to calculate the expected reward. The resulting algorithm, which we call the *expected dynamics* algorithm, is presented in Figure 1.

```
Suppose the policy depends on a vector of parameters $\theta$. When interacting with the MDP, at every
time $t$ after having taken action $a_{t-1}$ in state $s_{t-1}$ and arriving in state $s_t$:

1. $\tilde{\mu}(s_{t-1}, a_{t-1}) \leftarrow s_t$
2. $\tilde{\nu}(s_{t-1}, a_{t-1}) \leftarrow (s_t - \tilde{\mu}(s_{t-1}, a_{t-1}))^2$
3. $\tilde{s}_0 = s_t$
4. $\tilde{\sigma}_0^2 = 0$
5. $\tilde{V} = 0$
6. For every $\tau$ in $1 \dots n$:
   (a) $\tilde{s}_\tau = \tilde{\mu}(\tilde{s}_{\tau-1}, \pi(\tilde{s}_{\tau-1}))$
   (b) $\tilde{\sigma}_\tau^2 = \tilde{\nu}(\tilde{s}_{\tau-1}, \pi(\tilde{s}_{\tau-1})) + \left(\frac{d\tilde{\mu}_\pi(s)}{ds}(\tilde{s}_{\tau-1})\right)^2 \tilde{\sigma}_{\tau-1}^2$
   (c) $\tilde{r}_\tau = \frac{1}{2}\frac{d^2 r(s)}{ds^2}(\tilde{s}_\tau)\tilde{\sigma}_\tau^2 + r(\tilde{s}_\tau)$
   (d) $\tilde{V} = \tilde{V} + \gamma^{\tau-1}\tilde{r}_\tau$
7. Update the policy in the direction that increases $\tilde{V}$: $\theta = \theta + \alpha\frac{\partial \tilde{V}}{\partial \theta}$
```

Figure 1: The essential dynamics algorithm for a one dimensional state space. The notation $f(x) \leftarrow a$ means "adjust the parameters that determine f to make $f(x)$ closer to a," e.g. by gradient descent. $\tilde{\mu}_\pi(s) = \tilde{\mu}(s, \pi(s))$, and $\frac{d\tilde{\mu}_\pi(s)}{ds}(\tilde{s}_{\tau-1})$ is the derivative of $\tilde{\mu}_\pi(s)$ with respect to s, evaluated at $\tilde{s}_{\tau-1}$.

3 ALGORITHM DERIVATION

A Markov Decision Process (MDP) is a tuple $\langle S, D, A, P_{s,a}, r, \gamma\rangle$ where: S is a set of *states;* $D : S \to \mathbb{R}$ is the *initial-state distribution;* A is a set of *actions;* $P_{s,a} : S \to \mathbb{R}$ are the *transition probabilities;* $r : S \times A \to \mathbb{R}$ is the *reward;* and γ is the *discount factor.* This paper is concerned with continuous state and action spaces, in particular we assume $S = \mathbb{R}^{n_s}$ and $A = \mathbb{R}^{n_a}$. We use subscripts to denote time and superscripts to denote components of vectors and matrices. Thus, s_t^i denotes the ith component of the vector s at time t.

A (deterministic) *policy* is a mapping from a state to the action to be taken in that state, $\pi : S \to A$. Given a policy and a distribution P_t of states at time t, such as the initial state distribution or the observed state, the distribution of states at future times is defined by the recursive relation $P_{\tau+1}(s) = \int_S P_{s',\pi(s')}(s)P_\tau(s')ds'$ for $\tau > t$. Given such a distribution, we can define the expectation and the covariance matrix of a random vector x with respect to it, which we denote $\mathrm{E}_t[x]$ and $\mathrm{cov}_t(x)$ respectively. Thus, $\mathrm{E}_t[x] = \int xP_t(x)dx$ and $\mathrm{cov}_t^{i,j}(x) = \mathrm{E}_t[(x^i - \mathrm{E}_t[x^i])(x^j - E_t[x^j])]$. When P_t is zero except for a single state s_t, we introduce $\mathrm{E}[x|s_t]$ as a synonym for $\mathrm{E}_t[x]$ which makes the distribution explicit.

Given an MDP, we define the *limited horizon value function* for a given policy as $V_\pi(s_t) = \sum_{\tau=t}^{t+n} \gamma^{\tau-t}\mathrm{E}_\tau[r(s_\tau, \pi(s_\tau))]$ where the probability density at time t is zero except for state s_t. Also given a policy, we define two functions, the vector valued mean $\mu_\pi(s)$ and covariance matrix $\nu_\pi(s)$ of the next state. Thus, $\mu_\pi(s_t) = \mathrm{E}[s_{t+1}|s_t]$ and $\nu_\pi(s_t) = \mathrm{E}[(s_{t+1} - \mu_\pi(s_t))(s_{t+1} - \mu_\pi(s_t))^T|s_t]$. In *policy search,* we have a fixed set of policies Π and we try to find one that results in a value function with high values.

We transform the stochastic MDP M to a deterministic one $M' = \langle S', s_0', A', f', r', \gamma'\rangle$ as follows. A state in the new MDP is an ordered pair consisting of a state from S and a covariance matrix, denoted (s, Σ). The new initial state $s_0' = (\mathrm{E}_D[s], \mathrm{cov}_D[s])$. The new action space is the set of all possible policies for M, that is $A' = \{\pi | \pi : A \to S\}$. The state transition probabilities are replaced with a (deterministic) *state transition function* $f'(s'_t, a'_t)$, which gives the unique successor state that results from taking action $a'_t = \pi$ in state $s'_t = (s_t, \Sigma_t)$. We set $f'(s'_t, a'_t) = f'(s_t, \Sigma_t, \pi) = (\mu_\pi(s_t), \nu_\pi(s_t) + (\nabla\mu_\pi)\Sigma_t(\nabla\mu_\pi)^T)$. The reward $r'(s, \Sigma, \pi) = r(s) + \frac{1}{2}\mathrm{tr}(\left[\frac{\partial^2 r}{\partial i \partial j}(s)\right]\Sigma)$ where $\left[\frac{\partial^2 r}{\partial i \partial j}(s)\right]$ denotes the matrix of second derivatives of r with respect to each state variable. Finally, $\gamma' = \gamma$.

The strength of the method comes from the theorems below, which state that the above transform approximately captures the dynamics of the original probabilistic MDP to the extent that the original dynamics are "smooth." More specifically, the conditions are that: we have a good estimate of expected state and variance at a given time; the second partial derivatives of μ_π are all small; the first partials of ν_π are all small; the third partials of r are small; the gradient of μ_π is bounded; and the first four moments of the state distribution are bounded at every time.

Under these conditions the transformed MDP will result in good estimates at later times, and hence the reward and value functions will also be good estimates. Note that no particular distribution of states is assumed. The most unusual conditions are that the reward r be roughly quadratic, and that the value function is limited horizon, that is, includes only a limited number of future rewards. This motivates the use of shaping rewards. The first theorem bounds the error in approximating state, the second in covariance, the third in reward and the fourth in value. The heart of the first three proofs is simply a Taylor series expansion of the specified function, using the Lagrange form of the remainder. The fourth proof uses the familiar sum of an exponential series.

Theorem 1 *Fix a time t, a policy π, and a distribution of states P_t. Choose M_μ and M such that $\forall s, \sqrt{\sum_{i,j,k=1}^{n_s}\left(\frac{\partial^2}{\partial j \partial k}\mu^i_\pi(s)\right)^2} < M_\mu$, $\|\nabla\mu_\pi(\tilde{s}_t)\| < M$ and $\|\mathrm{cov}_t(s_t, s_t)\|_F < M$, where $\|\|_F$ denotes the Frobenius norm. Let $\tilde{s}_t$ be given, and define $\tilde{s}_{t+1} = \mu_\pi(\tilde{s}_t)$, $\epsilon^s_t = \mathrm{E}_t[s_t] - \tilde{s}_t$ and $\epsilon^s_{t+1} = \mathrm{E}_t[s_{t+1}] - \tilde{s}_{t+1}$. Then $\|\epsilon^s_{t+1}\| < (\|\epsilon^s_t\| + M_\mu)(\frac{3}{2}M + \frac{1}{2}\|\epsilon^s_t\|^2)$.*

Theorem 2 *Suppose M_ν and M are chosen so that $\forall s, \sqrt{\sum_{i,j,k=1}^{n_s}\left(\frac{\partial \nu^{i,j}}{\partial k}(s)\right)^2} < M_\nu$, $\|\mathrm{E}_t[|s_t - \mathrm{E}_t[s_t]|^k]\|_F < M$ for $k = 1, 2, 3, 4$, $\|\tilde{s}_{t+1}\| = \|\mu_\pi(\tilde{s}_t)\| < M$ and all the conditions of Theorem 1. Let $\tilde{\Sigma}_t$ be given, and define $\tilde{\Sigma}^{i,j}_{t+1} = \nu^{i,j}(\tilde{s}_t) + (\nabla\mu^i(\tilde{s}_t))^T\tilde{\Sigma}_t(\nabla\mu^j(\tilde{s}_t))$. Let $\epsilon^\Sigma_t = \mathrm{cov}_t(s_t, s_t) - \tilde{\Sigma}_t$, similarly for ϵ^Σ_{t+1}. Then*

$$\|\epsilon^\Sigma_{t+1}\|_F \leq (\|\epsilon^\Sigma_t\|_F + \|\epsilon^s_t\| + M_\mu + M_\nu)M^2(10 + O(\|\epsilon^s_t\|)).$$

Theorem 3 *Suppose $\forall s, \sqrt{\sum_{i,j,k=1}^{n_s}\left(\frac{\partial^3 r}{\partial i \partial j \partial k}(s)\right)^2} < M_r$, $\sqrt{\sum_{i,j=1}^{n_s}\left(\frac{\partial^2 r}{\partial i \partial j}(\tilde{s}_t)\right)^2} < M$ and $\|\nabla r(\tilde{s}_t)\| < M$ and the conditions of the previous two theorems. Let $\epsilon^r_t = \mathrm{E}_t[r(s_t)] - r'(\tilde{s}_t)$. Then $\mathrm{E}_t[r(s_t)] = r'(\tilde{s}_t) + \epsilon^r_t = r(\tilde{s}_t) + \frac{1}{2}\mathrm{tr}(\left[\frac{\partial^2 r}{\partial i \partial j}\right]\tilde{\Sigma}_t) + \epsilon^r_t$ where*

$$|\epsilon^r_t| < (\|\epsilon^\Sigma_t\|_F + \|\epsilon^s_t\| + M_r)\left(\frac{5}{3}M + O(\|\epsilon^s_t\|)\right).$$

Theorem 4 *Fix a time t and a policy π, and a distribution of states P_t. Let $\tilde{s}_t$ and $\tilde{\Sigma}_t$ be given, and define $\tilde{s}_\tau$ and $\tilde{\Sigma}_\tau$ for $\tau = t+1 \ldots t+n$ recursively as in Theorems 1 and 2 above. Let $M_{\epsilon r}$ be an upper bound for $|\epsilon^r_\tau|$ for all $\tau \in [t, t+n]$. Then under the conditions of the above three theorems, $\mathrm{E}[V(s_t)] = V'(\tilde{s}_t) + \epsilon^V_t$ where $|\epsilon^V_t| < \frac{1-\gamma^{n+1}}{1-\gamma}M_{\epsilon r}$.*

The proofs have been omitted for brevity, but are available in Martin (2003).

4 EXPERIMENT: LEARNING TO RIDE A BICYCLE

The code used for all experiments in this paper is available from www.metahuman.org/martin/Research.html.

The essential dynamics algorithm was applied to Randløv and Alstrøm's bicycle riding task (Randløv, 2000), with the objective of riding a bicycle to a goal 1 km away. The five state variables were simply the lean angle, the handlebar angle, their time derivatives, and the angle to the goal. The two action dimensions were the torque to apply to the handlebars and the horizontal displacement of the rider's center of mass from the bicycle's center line. The stochasticity of state transitions came from a uniform random number added to the rider's displacement. If the lean angle exceeded $\pi/15$, the bicycle fell over and the run terminated.

If the variance of the state is not too large at every time step, then the variance term in the transformed reward can simply be considered another form of error, and only $\tilde{\mu}$ need be estimated. This was done here. The task is not naturally divided into discrete time steps, but rather state and action are continuous in time. Rather than arbitrarily discretize time, a continuous time formulation was used where, instead of estimating the next state, their derivatives were estimated. The model was of the form

$$\frac{\partial s^i}{\partial t} = \tilde{\mu}_{w^i}(s, a) = w^i \cdot \phi(s, a)$$

where $\phi(s, a)$ was a vector of features and w^i was a vector of weights. The features were simply the state and action variables themselves. The derivative of each state variable was estimated using gradient descent on w^i with the error measure $err_i = |\frac{\partial s^i}{\partial t} - w^i \cdot \phi(s,a)|$ and a learning rate of 1.0. This absolute value error measure was found to work better than the more traditional squared error. The squared error is minimized by the mean of the observed values, whereas the absolute value is minimized by the median (Press et al., 1992). The median is a more robust estimate of central tendency, i.e. less susceptible to outliers, and therefore may be a better choice in many practical situations.

Model estimation was done online, simultaneous with policy search. In the continuous formulation, the value function is $V_\pi(s) = \int_t^{t+n} \gamma^{\tau-t} \mathrm{E}_\tau[r(s_\tau, \pi(s_\tau))]d\tau$. The future state was estimated using Euler integration (Press et al., 1992). While the bicycle simulator also used Euler integration, these choices were unrelated. In fact, the timesteps were different, with $\Delta t = 0.01$ sec for the bicycle simulator and 0.051sec for integrating the estimated reward. The horizon for the value function was 1.53sec, that is, 30 integration steps.

The shaping reward was the square of the angle to goal plus 10 times the square of the lean angle. The policy was a weighted sum of features, with a small Gaussian added for exploration, $\pi(s) = \theta \cdot \phi(s) + N(0, 0.05)$. The features were simply the state variables themselves. When the model is poor or the policy parameters are far from a local optimum, $\partial V/\partial\theta$ can be quite large, resulting in a large gradient descent step which may overshoot its region of applicability. This can be addressed by reducing the learning rate, but then learning becomes interminably slow. Thus, the gradient descent rule was modified to $\frac{\partial\theta}{\partial t} = -\alpha \frac{\partial V/\partial\theta}{(\beta + \|\partial V/\partial\theta\|)}$. Near an optimum, when $\|\partial V/\partial\theta\| \ll \beta$, this reduces to the usual rule with a learning rate of α/β. In this experiment, $\alpha = 0.01$ and $\beta = 1.0$.

A graph of episode time vs. learning time is shown in Figure 3. After falling over between 40 and 60 times, the controller was able to ride to the goal or the time limit without falling over. After a single such episode, it consistently rode directly to the goal in a near minimum amount of time. The resulting policy was essentially an optimal policy.

5 EXPERIMENT: A ROBOT ARM ON A DYNAMIC BASE

Cardea (Brooks, 2003) is a humanoid robot consisting, during this experiment, of a five degree of freedom arm on a Segway RMP. The Segway is essentially an inverted pendulum, and it's controller that can't be modified. The state space consisted of the five revolute joint angles, the tilt angle and fwd/back position

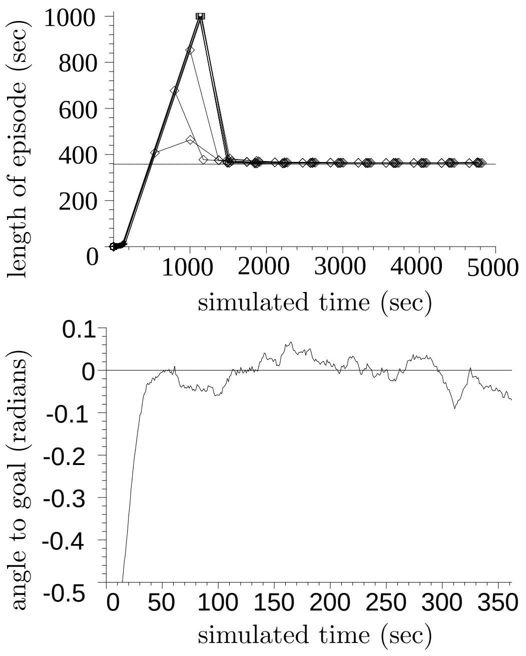

Figure 2: The upper graph shows length of episode vs. training time for 10 runs. The dashed line indicates the optimal policy. Stable riding was achieved within 200 simulated seconds. The lower graph shows angle to goal vs. time for a single episode starting after 3000 simulated seconds of training.

of the base, and their time derivatives, for a total of 14 dimensions. An action was vector of five joint torques. Cardea was simulated using the Open Dynamics Engine (Smith, 2003), and the results are currently being applied to the actual robot. For more details, see Martin (2004).

The goal was simply to achieve a given end effector position. The desired angles (and associated shaping reward) could change between episodes, which means this isn't a single task, but rather a family of tasks. As in many robotics tasks, it is easier to plan in state space than in action space. Therefore, an *inverse model* (Atkeson et al., 1997) was learned, which mapped the desired next state to action, along with an *inverse policy*, mapping current state and goal state to the desired next state. It was straightforward to modify the essential dynamics engine to perform this.

The model was a linear combination of features, where the features were products of various subsets of: the sin and cosine of all angles, base position, all velocities, desired acceleration and a constant (1). The policy was a linear combination of features, where the features were simply the state variables and desired joint angles. Gaussian noise was added to the desired torques to ensure exploration. The algorithm parame-

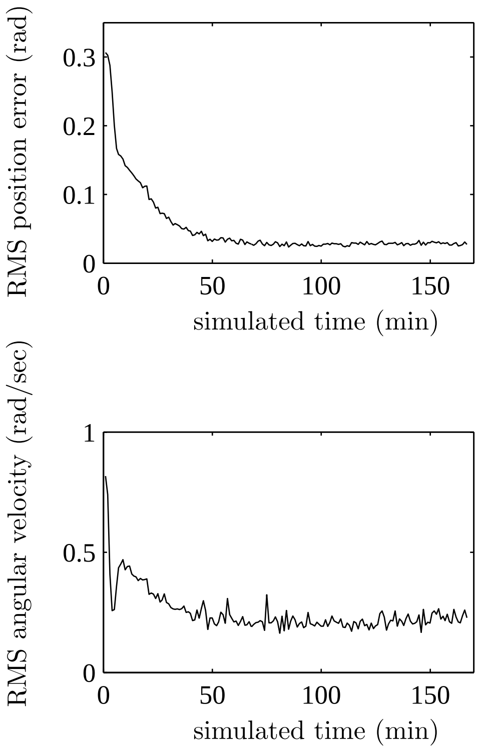

Figure 3: Position error and angular velocity vs. simulated time, at the end of every episode. Averaged over 20 runs, with actions randomly perturbed for exploration. The algorithm ran slightly faster than real time.

ters were only tuned to within an order of magnitude. The shaping reward was simply the sum of the squared error in each joint angle.

A 990 MHz mobile Pentium III processed 3000 episodes per hour, slightly faster than real time. A graph of the error in joint angles and joint velocities at the end of every episode is shown in Figure 3. After three or four minutes, the policy was good enough to allow the arm to stay in the air without hitting the base or itself. By half an hour it reached most goals from most initial positions, although it slowly oscillates around the target. After four hours the workspace had widened considerably, while the oscillation was greatly reduced.

6 DISCUSSION

For learning and planning in complex worlds with continuous, high dimensional state and action spaces, the goal is not so much to converge on a perfect solution, but to find a good solution within a reasonable time. Such problems often use a shaping reward to accelerate learning. For a large class of such problems, this paper proposes approximating the problem's dynamics in such a way that the mean and covariance of the future state can be estimated from the observed current state. We have shown that, under certain conditions, the rewards in the approximation are close to those in the original, with an error that grows boundedly as time increases. Thus, if the value function has a limited horizon, the resulting values will approximate the values of the original system. Learning in this transformed problem is considerably easier than in the original, and both model estimation and policy search can be achieved online.

The algorithm can be seen as a generalization of the linear quadratic regulator, an adaptive control theory technique. We consider the success of linear models in many practical regulator problems over half a century to indicate that the dynamics of many problems are locally linear. For these problems, we extend work on regulators (which are only interested in the state near the desired) to general control algorithms (able to work in the entire state space) by using an arbitrary function approximator for both model and policy, but assuming that the second derivative of the model with respect to state is small. An extra benefit comes if the policy is roughly a weighted sum of the state variables. In that case the value is roughly quadratic in the policy parameters, so the search problem is roughly convex and we would expect all local minima to be clustered around the global minima.

To gain insight into how the algorithm compares to existing techniques, and how it works on an existing problem, it was applied to the bicycle riding task of Randløv and Alstrøm, which has also been tackled by the PEGASUS algorithm of Ng and Jordan (2000). They describe the PEGASUS algorithm in terms of MDPs, but the algorithm does not exploit the structure of MDPs and can be described more simply without them.

Given a family of policies parameterized by a vector θ, the PEGASUS algorithm attempts to find the θ that maximizes the total expected reward. It requires a simulator which it treats as a black box function f that, given policy parameters θ and a sequence of (pseudo-) random numbers ξ, returns the total reward. Thus, PEGASUS attempts to estimate $E_\xi[f(\theta, \xi)]$, where f is deterministic, although ξ is random. This is a traditional operations research problem, and PEGASUS applies the variance reduction technique known as *common random numbers* (see section 5 of Gaver (1969), section 11.7 of (Fishman, 1973)). Specifically, a number of ξs are generated, and that one set of ξs is used for all θs. A thorough theoretical analysis of this technique can be found in Gal et al. (1984).

The bicycle riding task is a good example of a problem where the value function can be quite complicated, yet a simple policy is near optimal. Randløv, using a traditional value function approximation approach, needed to augment the state with the second derivative of the lean angle ($\ddot{\Omega}$) and provide shaping rewards (Randløv, 2000). The resulting algorithm took 1700 episodes to ride stably, and 4200 episodes to get to the goal for the first time. The resulting policies tended to ride in circles and precess toward the goal, riding roughly 7 km to get to a goal 1km away.

Both the essential dynamics algorithm and the PEGASUS algorithm are policy search algorithms where the policy is a weighted sum of features, and both find policies within 1% of optimal. In fact, a random search of policies can find such near optimal ones quickly. We tested this experimentally[1], and found that 0.55% of random policies consistently reached the goal when $\ddot{\Omega}$ was included in the state, and 0.30% when it wasn't. What's more, over half of these policies had a path length within 1% of optimal. Policies that rode stably, but not to the goal, were obtained 0.89% and 0.24 % of the time respectively. Thus, a random search of policies needs only a few hundred episodes to find a near optimal one.

Turning to learning time, the essential dynamics algorithm took 40 to 60 episodes to ride stably, that is, to the goal or until the time limit without falling over. After a single such episode, the policy consistently rode directly to the goal in a near minimum amount of time. In contrast, PEGASUS used at least 450 episodes to evaluate each policy[2]. One reasonable initial policy is to always apply zero torque to the handlebars and zero displacement of body position. This falls over in an average of 1.74 seconds, so PEGASUS would need 780 simulated seconds to evaluate such a policy. The essential dynamics algorithm learns to ride stably in approximately 200 simulated seconds, and in the second 780 simulated seconds will have found a near optimal policy.

A closer look at the algorithms explains the disparity of learning times. In value and Q-learning, as well as REINFORCE style policy search, a single episode gives a sample of the value for every state visited. That is, the reward from a given timestep is used to update the value of every previous state. The simulator in PEGASUS, on the other hand, returns only the discounted reward from the initial state, which is much less information per episode.

The essential dynamics algorithm achieved its performance using very little domain knowledge. $\ddot{\Omega}$ was not needed in the state, and the features were trivial. The experiments in section 4 added the square of the lean angle to the shaping reward, but did not use any information about dynamics (i.e. velocities or accelerations), nor about the handlebars. In fact, the shaping reward simply corresponded to the common sense advice "stay upright and head toward the goal." A final advantage of the algorithm is that it can be used for online learning, or can learn from trajectories provided by other policies, that is, it can be an off-policy method.

These advantages allowed it to learn to control a robot arm on a Segway base, a problem with a 14 dimensional state space and 5 dimensional action space. This is remarkable, because existing applications of policy search are typically 5 or 6 dimensions, with a single example of an 8 dimensional search. The algorithm was easily modified to learn an inverse model and policy, and parameters were only tuned to within an order of magnitude. The shaping reward was again a simple function of position, yet the algorithm learned to include damping in the policy.

However, these advantages do not come without drawbacks. It needed many times more computing power per simulated second than PEGASUS, although it was still faster than real time on a 1GHz mobile Pentium III, and therefore could presumably be used for learning on a real bicycle. It only does policy search in an approximation to the original MDP, so an optimal policy for this approximate MDP won't, in general, be optimal for the original MDP. The theorems in section 3 give bounds on this error, and for bicycle riding this error is small.

As well, it relies on the state being a continuous function of the previous state. When the algorithm was applied to a simulation of a bipedal walking robot, it failed to accurately predict future state when a joint limit was reached, where an angular velocity instantaneously dropped to zero. This prevented the policy search from finding a policy that could take even a single step, even with the help of a user provided hierarchical decomposition.

[1] Our experiment contained two conditions, namely with or without $\ddot{\Omega}$ in the state, resulting in 5 or 6 state variables. The features were the state variables themselves, state and action variables were scaled to roughly the range [-1, +1], weights were chosen uniformly from [-2, +2], and each policy was run 30 times. In 100,000 policies per condition, 549 (0.55%) reached the goal all 30 times when was included, and 300 (0.30%) when it wasn't. For such policies, the median riding distance was 1009m and 1008m respectively. The code used is available on our web site.

[2] Ng and Jordan (2000) evaluated a given policy by simulating it 30 times. The derivative with respect to each of the 15 weights was evaluated using finite differences, requiring another 30 simulations per weight, for a total of $30 \times 15 = 450$ simulations. In general, the starting weights at a given stage are often evaluated during the previous stage, so only the derivatives need to be calculated.

7 CONCLUSION

For many problems (e.g. NP-hard problems), the optimal solution can take far too long to find. Thus, it is little comfort to know that a given algorithm would eventually converge to the optimal solution. In such domains, it may be more useful to quickly find an approximate solution, even if there is a lower bound on the solution's error. The essential dynamics algorithm does just that, approximating a given MDP by a deterministic one, thus greatly speeding up learning. When applied to the bicycle riding domain, it finds a near optimal solution in orders of magnitude less time than previous approaches. It has also allowed us to control a simulation of a robot arm on a Segway base, a problem of far higher dimensionality than previous applications.

And it accomplishes this in a few hours with very little domain knowledge. Crossing control theory techniques with reinforcement learning seems to hold much promise.

ACKNOWLEDGEMENTS

The author would like to thank Kevin Murphy, Leslie Kaelbling, Russ Tedrake, Ali Rahimi, and Eduardo Torres-Jara for enlightening comments and discussions of this work. This work was funded by DARPA under contract number DABT 63-00-C-10102.

References

Åström, K. J., & Wittenmark, B. (1995). *Adaptive control*. Addison-Wesley. 2nd edition edition.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning for control. *Artificial Intelligence Review*, *11*, 75–113.

Baxter, J., & Bartlett, P. (2000). Reinforcement learning in pomdp's via direct gradient ascent. *Proc. 17th Intl. Conf. on Machine Learning*.

Brooks, R. (2003). Cardea web site. http://www.ai.mit.edu/projects/cardea.

Colombetti, M., & Dorigo, M. (1994). Training agents to perform sequential behavior. *Adaptive Behavior*, *2*, 247–275.

Dorato, P., Abdallah, C. T., & Cerone, V. (1995). *Linear quadratic control: An introduction*. Prentice-Hall.

Fishman, G. S. (1973). *Concepts and methods in discrete event digital simulation*. John Wiley & Sons.

Forbes, J., & Andre, D. (2000). Real-time reinforcement learning in continuous domains. *AAAI Spring Symposium on Real-Time Autonomous Systems*.

Gal, S., Rubinstein, R. Y., & Ziv, A. (1984). On the optimality and efficiency of common random numbers. *Math. Comput. Simulation*, *26*, 502–512.

Gaver, D. P. (1969). Statistical methods for improving simulation efficiency. *Proc. of the third conf. on Applications of simulation* (pp. 38–46).

Martin, M. C. (2003). *The essential dynamics algorithm: Essential results* (Technical Report AIM-2003-014). Massachusetts Institute of Technology, Artificial Intelligence Lab.

Martin, M. C. (2004). Controlling cardea: Fast policy search in a high dimensional space. *Proc. 21st Intl. Conf. on Machine Learning (ICML '04) in submission.*

Mataric, M. J. (1994). Reward functions for accelerated learning. *Proc. 11th Intl. Conf. on Machine Learning*.

Ng, A., et al. (1999). Policy invariance under reward transformations: Theory and applications to reward shaping. *Proc. 16th Intl. Conf. on Machine Learning* (pp. 406–415).

Ng, A., & Jordan, M. (2000). Pegasus: A policy search method for large mdps and pomdps. *Uncertainty in Artificial Intelligence (UAI), Proc. of the Sixteenth Conf.* (pp. 406–415).

Press, W. H., Teukolsky, S. A., Vetterling, W., & Flannery, B. (1992). *Numerical receipes: The art of scientific computing*. Cambridge University Press. 2 edition.

Randløv, J. (2000). Shaping in reinforcement learning by changing the physics of the problem. *Proc. 17th Intl. Conf. on Machine Learning* (pp. 767–774).

Roy, N., & Thrun, S. (2002). Motion planning through policy search. *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*. Lausanne, Switzerland.

Smith, R. (2003). Open dynamics engine. http://q12.org/ode/.

Strens, M. J. A., & Moore, A. (2002). Policy search using paired comparisons. *Journal of Machine Learning Reserach*, *3*, 921–950.