Controlling Cardea:
Fast Policy Search in a High Dimensional Space

Martin C. Martin
mcm@media.mit.edu
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

The *essential dynamics* algorithm is a novel policy search algorithm for learning in a class of stochastic Markov decision processes (MDPs) with continuous state and action spaces. We apply it to the control of a 5 degree of freedom robot arm atop a Segway base. Movement of the arm causes the base to translate and tilt, which in turn affects the movement of the arm. The state space has 14 dimensions, and the action space 5 dimensions, twice the dimensionality of typical policy search applications. Despite the highly non-linear dynamics, the algorithm is able to control the robot through a wide range. What's more, this is accomplished using very little domain knowledge and no knowledge of dynamics.

1 INTRODUCTION

In stochastic Markov decision processes (MDPs) with continuous state and action spaces, the value and Q-functions are often very complicated and difficult to approximate. In fact, Baxter and Bartlett (2000) demonstrated that policies derived from approximate value functions can fail arbitrarily badly. This has lead to recent interest in *policy search algorithms*, that search the space of policies directly (Roy & Thrun, 2002; Bagnell & Schneider, 2001; Strens & Moore, 2002).

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. To circumvent this, the *essential dynamics algorithm* (Martin, 2003) transforms the stochastic MDP into a deterministic one. The resulting MDP only approximates the original, but for a wide class of problems, values in the deterministic MDP are good approximations to those in the original. This approximation can then be used to 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.

It can also be seen as a generalization of the adaptive control theory technique *linear quadratic regulation*. In a *regulation task* the goal is stay near to a desired point in the state space. A linear quadratic regulator models the next state as a linear function of current state and action, and uses a quadratic shaping reward. The essential dynamics algorithm generalizes the goal to acting in the entire state space. It does this by assuming that the next state is locally linear, that is, that its second derivative is small. It also assumes the shaping reward is only roughly quadratic. The optimal control parameters can no longer be found analytically, so gradient descent is used. We note in passing that other combinations of adaptive control and policy search may prove even more fruitful.

The algorithm has been applied to applied to Randløv and Alstrøm's bicycle riding task (Randløv, 2000). In previous approaches the state was augmented with an acceleration, "training wheels" were added, or 15 hand crafted features were used. In contrast, the only domain knowledge needed by the essential dynamics algorithm was a shaping reward penalizing for lean angle and facing away from the goal. This corresponds to the common sense advice "stay upright and head toward the goal." Even with this meagre domain knowledge, it discovered a near optimal policy in orders of magnitude less simulated time (Martin, 2004).

Cardea (Brooks, 2003) is the latest robot of MIT's humanoid robotics group. It uses a SEGWAY RMP (robotic mobility platform) for locomotion, and a 5 degree of freedom (DOF) robotic arm for manipulation. Learning was performed in a detailed physical simulation of Cardea, and is currently being transferred to

the physical robot.

The control of a robot arm has been studied for several decades. The classical approach (Craig, 1989) involves two steps. First a PD controller to provide the desired angular acceleration of each joint, then these are used to compute the torques. This second step is complicated by the fact that a torque at one joint can cause accelerations at all of the other joints. The classical solution requires that the acceleration of the shoulder is known, and works through kinematics and dynamics of each joint to determine the torques.

The classical method requires knowledge of the mass and moment of inertia of each link, parameters that are difficult to measure. But more importantly for Cardea, the acceleration of the shoulder depends on the motion of the base, which in turn depends on the torques applied to the arm. Thus, the classical technique doesn't apply.

Many function approximators have been applied to the problem, such as neural networks and genetic algorithms (Moriarty & Miikkulainen, 1996). However, they often take a long time to learn, especially in the presence of significant noise. Reinforcement learning has been applied as well (Morimoto & Doya, 2000), but even a simple two joint, three link arm required a hierarchical form of RL.

The next section motivates and derives the essential dynamic algorithm. Section 3 discusses Cardea and its simulator, and describes the details of how the algorithm learned to control the simulation.

2 THE ESSENTIAL DYNAMICS ALGORITHM

In the essential dynamics algorithm we learn a model and a policy simultaneously. A *model* describes the dynamics of the world: it maps a state and action to the next state. A *policy* describes an agent's behavior: it maps the current state to the action to be taken in that state. A *reward* maps the current state and current action to a real number. The *value* of a given policy in a given state is the sum of all rewards received from starting in the state and following the policy.

In typical applications of reinforcement learning, a positive reward comes only after successfully completing the task. For many problems, the probability of doing this at random is vanishingly small. Even when a successful finish can be found with reasonable frequency, the credit assignment problem – backing up the rewards to the values of the previous states – can be incredibly time consuming, especially when the state space is large. A widely used methodology to ad-

dress this problem is *shaping* (Colombetti & Dorigo, 1994; Ng et al., 1999; Randløv, 2000). In shaping, rewards are introduced for intermediate progress toward the goal. A shaping function eases the problem of backing up rewards, since actions are rewarded or punished sooner.

Therefore, we use a *limited horizon* for our value, that is, we only sum the rewards over the next T timesteps. The value of the current state is easily computed. Applying the policy to current state gives us an action, and applying the model to the current state and action gives us the next state. Iterating T times gives us a *trajectory* of T states and actions, from which we can compute values and rewards. If the policy and model are from a parameterized family, we can compute the gradient of the value with respect to the policy 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

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

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

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

Suppose the policy depends on a vector of parameters ξ. 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 τ in $1 \ldots n$:

    (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 $\tilde{V}$: $\xi = \xi + \alpha\frac{\partial \tilde{V}}{\partial \xi}$

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 $\tilde{s}_{\tau-1}$.

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

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

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.

Martin (2003) provides a mathematically rigorous derivation of the algorithm, and proves bounds on the magnitude of the error in expected state, state variance, reward, and value. The algorithm is also used to learn to ride a bicycle in simulation, and is compared to existing techniques.
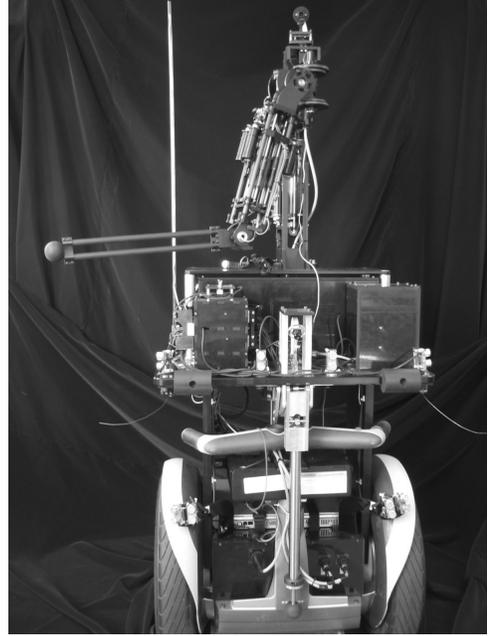

Figure 2: Cardea consists of a 5 degree of freedom arm atop a Segway RMP base.

3 CONTROLLING CARDEA

3.1 CARDEA

Cardea locomotes using a SEGWAY RMP (robotic mobility platform). Its cameras and sonar sensors were not used in this experiment. It also has a 5 degree of freedom (DOF) robotic arm. Eventually Cardea's current vision system will be replaced by a robotic head with active perception, and will be outfitted with three arms.

The goal of the Cardea project is to navigate unstructured interior spaces using simple active vision and sensors while being able to use its single arm to reach out and interact with objects in compliant and safe ways. One behavior-level scenario fitting these goals is Cardea wheeling along a corridor trying to detect a door. Once the door is detected, Cardea will turn and approach it, open the door, and enter the room. The Segway RMP is attractive because it can dynamically balance on a small wheelbase that is able to support a torso, arms and "head" at a human scale height. Without dynamic balancing, a two wheeled base would have to be much wider than the typical width of a persons "footprint".

The current prototype arm under development has 5 DOF. There is a 2 DOF differentially driven shoulder which provides pitch and yaw, a 1 DOF elbow, and a 2 DOF differentially driven wrist. The arm has been de-

signed to roughly match the speed, power, weight, and proportions of a human arm. It has analog torque and angle feedback at each joint. Each DOF uses a *series elastic actuator* (Pratt & Williamson, 1995), which is an intentional spring between the gears and the load, that allows the applied force (or torque) to be measured, and therefore controlled directly.

The results reported here are for a detailed physical simulation of Cardea, and are currently being transferred to the real robot. The Cardea simulation used the Open Dynamics Engine (ODE) (Smith, 2003). ODE is an open source engine for simulated rigid body, and provides geometrical primitives such as boxes and cylinders, as well as constraints between them. The simulated Cardea is comprised of six bodies: two wheels, the Segway base, the upper arm, the fore arm, and the hand. The shoulder and wrist are both modelled as a universal joints, and the elbow as a hinge. The only other object in Cardea's world was the ground plane.

3.2   CONTROL

The classical solution (Craig, 1989) requires calibration, including knowledge of parameters such as moment of inertia, that are difficult to measure. It also requires that the acceleration of the shoulder is known at each timestep, yet the acceleration of the shoulder depends on the motion of the base, which in turn depends on the very torques we wish to calculate.

Tasks were specified as the desired angle of each joint. The task was fixed during the episode, but changed between episodes. Tasks were generated by choosing an end point for the hand in spherical coordinates centered at the shoulder. Because the immediate goal of Cardea's larger research program is to open doors, the desired orientation of the hand was parallel to the ground. Inverse kinematics was then performed to determine the desired angles. The configuration of the robot made this a straightforward application of high school trigonometry.

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.

At the start of the simulation, the arm was initialized to be fully extended in front of the robot, with zero velocity. An episode would end when either (a) $\|\vec{\theta} - \vec{\theta}_d\|^2 < 0.001$ and $\|\dot{\vec{\theta}}\|^2 < 0.001$, (b) 2 seconds had elapsed, (c) the arm collided with either the body or

itself, or (d) parts of the arm moved so fast that the simulation became unstable. In the last two cases, the arm was reset to fully extended. In the first two it was untouched, starting the next episode with whatever angles and velocities it had at the end of the previous episode. In all cases, a new task was generated.

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 the next state was replaced by the time derivatives of state.

Model estimation was done online, simultaneous with policy search. The policy mapped the 14 state variables and the five desired angles to five desired arm accelerations. The policy and (inverse) model were both weighted sums of features. The policy's features were simply the input variables themselves, plus a constant (1). The inverse model mapped the 14 state variables and five desired angular accelerations to the five joint torques and two accelerations of the base. The model features were various products of the angular velocities, the desired accelerations, and the sin and cosine of the joint angles. Using the sin and cosine of the joint angles rather than the angles themselves is a standard technique from the neural network literature (Masters, 1993), which ensures that the model is periodic in the angles, without discontinuity.

The model parameters were estimated using gradient descent on the absolute value of the error, rather 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. The learning rate was 0.001.

The shaping reward was simply $-\|\vec{\theta} - \vec{\theta}_d\|^2$, where $\vec{\theta}_d$ are the desired joint angles. It needed no knowledge of dynamics, and did not directly reward damping. Gaussian noise was added to the torques. In the continuous time 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 physical simulation also used Euler integration, these choices were unrelated. In fact, the timesteps were different, with $\Delta t = 0.01$ sec for the physical simulator and 0.05 sec for integrating the estimated reward. The horizon for the value function was 1.5 sec, that is, 30 integration steps.
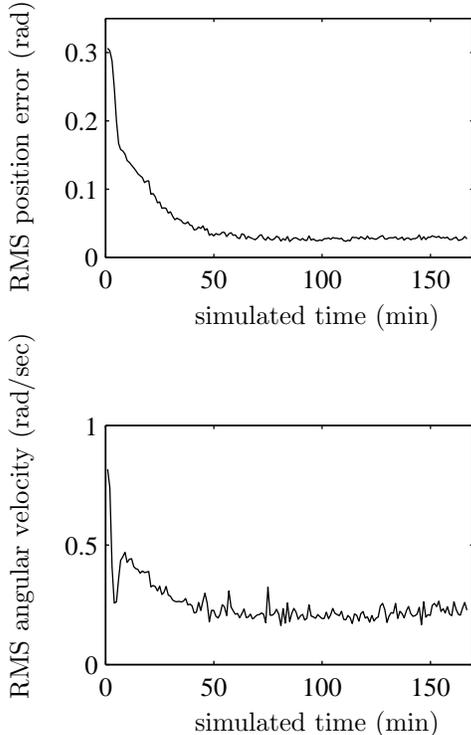
Figure 3: Position error and angular velocity vs. simulated time, at the end of every episode. Averaged over 20 runs, with actions randomly perturbed for exploration. The algorithm ran slightly faster than real time.

Figure 4: Position error and angular velocity vs. simulated time, over a single episode. Without exploration.

Estimating the future state and integrating the reward were the most CPU intensive aspects of algorithm, but were only needed for updating the policy, not the model. Thus, while the model was updated every timestep, the policy was only updated every 100 timesteps. When the model is poor or the policy parameters are far from a local optimum, $\partial V / \partial \xi$ 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. Therefore, the gradient descent rule was modified to $\Delta \xi = -\alpha \frac{\partial V / \partial \xi}{(\beta + \| \partial V / \partial \xi \|)}$. Near an optimum, when $\| \partial V / \partial \xi \| \ll \beta$, this reduces to the usual rule with a learning rate of α / β. In this experiment, $\alpha = 0.1$ and $\beta = 100$.

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. The initial policy applied zero torque in every state, causing the arm to fall under gravity. After three or four minutes, the policy was good enough to allow the arm to stay in
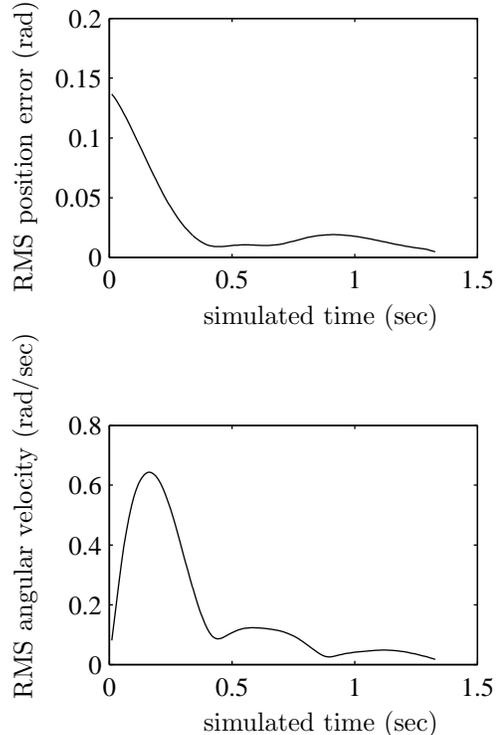
the air without hitting the base or itself. By half an hour it reached most goals from most initial positions, although it slowly oscillates around the target. After four hours the workspace had widened considerably, while the oscillation was greatly reduced. An example trajectory of the final policy is shown in Figure 4.

4 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. If the shaping reward is quadratic, the reward and value of future states depend only on the mean and variance of the state distribution. Thus, the essential dynamics algorithm estimates these for future time, uses them to estimate the value of a given state, then uses this to adjust the policy. 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 real world regulator problems over half a century to indicate that the dynamics of many systems are locally linear. For these systems, 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 assuming that the second derivative of the model 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.

Martin (2004) compares the algorithm to traditional value function approximation as well as PEGASUS (Ng & Jordan, 2000) on the bicycle riding task of Randløv and Alstrøm (Randløv, 2000). Value function approximation took 4200 episodes to get to the goal for the first time, and only ever found policies that rode in circles, precessing toward the goal. In contrast, when cast as a policy search problem, not only do PEGASUS and the essential dynamics algorithm find essentially optimal policies, but somewhat surprisingly, so does random search. In fact, PEGASUS takes far more time than random search, while the essential dynamics algorithm takes significantly less. This is not surprising, as PEGASUS doesn't exploit the structure of the MDP, and thus requires many episodes to compute the gradient of the policy.

Given that the essential dynamics algorithm trades accuracy and general applicability for speed, we would expect it to perform well at higher dimensions than existing techniques. Table 1 lists the policy search problems with the largest state space dimensionality found in an extensive literature search. The helicopter control is cast as a regulation problem, i.e. the goal is to hover over a fixed point on the ground, or follow a slowly moving point. Their model, cost function, and even choice of state were heavily informed by helicopter modeling and control theory. The resulting policy was able to hover with much less error than under a highly trained pilot's control.

The multiple-pursuer evader is a toy problem in which two pursuers must learn to cooperate to catch a highly maneuverable evader in a two dimensional plane. The evader followed a fixed, hand written policy. The policy for each pursuer was a weighted sum of 6 features: relative angle, range, the time derivatives of those, the relative range of the pursuers, and a constant. Unfortunately, the authors don't say how well their method performed, nor how long it took. How other methods fair on this problem is unknown.

In learning to control Cardea, the essential dynamics algorithm used very little domain knowledge. The only

Table 1: Dimentionality of Successful Policy Search Applications

Problem Name	Dimensions State, Action	Reference
Pole Balancing	4, discrete	(Davies et al., 1998)
Robot Motion Planning	5, 2	(Roy & Thrun, 2002)
Riding a Bicycle	5 or 6, 2	(Ng & Jordan, 2000)
Helicopter Control	8, 2	(Bagnell & Schneider, 2001)
Multiple-Pursuer Evader	9, 2	(Strens & Moore, 2002)
Robot Arm on Segway	14, 5	this paper

features were products of the various state variables, with angles represented as their sin and cosine. The reward was a function of position only, not velocity, yet it nonetheless learned to damp the motions.

Learning took only a few hours on a laptop computer with parameters that had been tuned to merely an order of magnitude. It learned not just to stay near a desired point in state space, but to competently reach all states of practical interest in a very high dimensional state space. Encouraged by it's success in the Cardea simulation, we are currently applying the essential dynamics algorithm to the physical robot.

Along with these advantages, the algorithm has some drawbacks. The policies only approximate the optimal ones, although Martin (2003) proves bounds on the error. The error seems small in both the bicycle riding domain and for Cardea.

More significantly, it does not account for discontinuities in state transitions. When the algorithm was applied to a simulation of a bipedal walking robot, joint limits would cause angular velocities to instantly drop to zero. The algorithm failed to accurately predict future state in this case, which prevented it from finding a policy that could take even a single step. This was even with the help of a hierarchical decomposition constructed by hand.

5 Conclusion

For many real world problems, finding the optimal solution can take prohibitively long. For these problems it is little comfort to know at a given algorithm will converge to the optimal solution eventually. In such domains it may be more useful to quickly find an approximate solution, even if there is an upper bound on the solution's performance.

One such domain is the control of a robot arm atop a Segway base. Torques on the arm cause the base to react which in turn causes motion of the arm, making traditional methods inapplicable. The essential dynamics algorithm performed well, learning a policy that worked reliably for the entire space in only a few hours, with very little domain knowledge. This is all the more remarkable given that the state space had 14 dimensions and the action space 5.

The combination of control theory and reinforcement learning techniques seems very powerful. Other combinations may prove even more fruitful.

Acknowledgements

References

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

Bagnell, J. A., & Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. *Proc. Intl. Conf. on Robotics and Automation*. Korea.

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.

Craig, J. J. (1989). *Introduction to robotics: Mechanics and control*. 2nd edition.

Davies, S., Ng, A., & Moore, A. (1998). Applying online search techniques to continuous-state reinforcement learning. *Proc. of the Fifteenth Natioinal Conf. on Artificial Intelligence (AAAI-98)* (pp. y53–760).

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

Martin, M. C. (2004). *The essential dynamics algorithm: Fast policy search in continuous worlds* (Technical Report 582). Massachusetts Institute of Technology, Media Laboratory, Vision and Modelling.

Masters, T. (1993). *Practical neural network recipes*. Academic Press.

Moriarty, D. E., & Miikkulainen, R. (1996). Evolving obstacle avoidance behavior in a robot arm. *From Animals to Animats: The Fourth Intl. Conf. on Simulation of Adaptive Behavior (SAB96)*.

Morimoto, J., & Doya, K. (2000). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Proc. 17th Intl. Conf. on Machine Learning (ICML'00)*.

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).

Pratt, G. A., & Williamson, M. M. (1995). Series elastic actuators. *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS-95)* (pp. 399–406). Pittsburgh, PA.

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.