Lyapunov-Constrained Action Sets for Reinforcement Learning

Theodore J. Perkins PERKINS@CS.UMASS.EDU
Andrew G. Barto BARTO@CS.UMASS.EDU
Department of Computer Science, University of Massachusetts Amherst, Amherst, MA, 01003

Abstract

Lyapunov analysis is a standard approach to studying the stability of dynamical systems and to designing controllers. We propose to design the actions of a reinforcement learning (RL) agent to be descending on a Lyapunov function. For minimum cost-to-target problems, this has the theoretical benefit of guaranteeing that the agent will reach a goal state on every trial, regardless of the RL algorithm it uses. In practice, Lyapunov-descent constraints can significantly shorten learning trials, improve initial and worst-case performance, and accelerate learning. Although this method of constraining actions may limit the extent to which an RL agent can minimize cost, it allows one to construct robust RL systems for problems in which Lyapunov domain knowledge is available. This includes many important individual problems as well as general classes of problems, such as the control of feedback linearizable systems (e.g., industrial robots) and continuous-state path-planning problems. We demonstrate the general approach on two simulated control problems: pendulum swing-up and robot arm control.

1. Introduction

In many interesting minimum cost-to-target problems, the target, or goal, region is not a natural attractor in the dynamics of the system. Consider, for example, the acrobot—a two link pendulum which is to be swung to a near upright position (e.g. Sutton, 1996; Boone, 1997a; DeJong, 2000). To swing up the acrobot, a controller must counteract the effects of gravity and friction (if any), which favor a stationary, downward-hanging position. The fact that the goal is not an attractor in the dynamics is part of what makes this problem challenging for RL, despite the moderate dimensionality of its state space (4 dimensions) and action space (1 dimension, usually discretized into 2 or 3 actions). Conversely, when the goal is a natural attractor, the problem of learning to control the system may be significantly easier. The success of TD-gammon (Tesauro, 1994) may partly be due to the fact that games naturally tend to end; under almost any policy, pieces progress around the backgammon board and get home. Attracting goal states tend to shorten learning trials. This eases the temporal credit assignment problem and allows more trials, and thus more meaningful learning, per unit time.

In this paper, we study the possibility of making the goal of a control problem an attracting region by judicious selection of the actions made available to an RL agent. In particular, we propose to use Lyapunov methods, which have been developed primarily for the analysis of systems of (controlled) differential equations and for control design (e.g., Vincent & Grantham, 1997). A Lyapunov function constitutes a form of domain knowledge about a control problem. Such a function can be used to design or constrain the action choices of an RL agent so that they naturally lead the agent to the goal.

Expressing domain knowledge by manipulating the actions available to an RL agent has been explored before by a number of researchers (e.g. Maclin & Shavlik, 1996; Clouse, 1997; Parr, 1998; Precup, 2000; Dietterich, 2000). By focusing an agent's attention on actions already known or expected to be good, or by decomposing a problem into smaller subproblems, faster and better learning can be achieved. Our work can be viewed as seeking similar benefits, but we concentrate on a particular kind of domain knowledge, namely, Lyapunov functions. In addition to providing a beneficial heuristic bias, we seek to provide theoretical guarantees on the behavior of the RL agent. Lyapunov methods can be used to prove that an agent will reach a goal state on every trial and to give time bounds on how long trials can take. Such guarantees are particularly important for systems in which training time is at a premium or in which reasonable performance must be maintained during learning. Because Lyapunov-based performance guarantees follow from properties of the environment and of the actions available to an agent, they hold for any RL algorithm.

Although Lyapunov-based methods can ensure that an RL agent always reaches a goal state and protect against catastrophic behavior, they do not solve the optimal control problem. Typically, Lyapunov methods do not directly address the cost of controlling the system at all. Thus, it is sensible to combine Lyapunov methods with control methods, such as RL, that do attempt to minimize the cost incurred by the controller.

The combination of Lyapunov methods and RL has been studied before by Singh et al. (1994). Although specific to path-planning tasks and a particular kind of Lyapunov function, that work and the present paper have similar goals: robustness and more efficient learning.

2. Problem Definition

In this paper, we restrict attention to the control of deterministic dynamical systems. Lyapunov methods and RL are applicable to both deterministic and stochastic systems, but Lyapunov theory is considerably more complex in the stochastic case. The general ideas we present can be extended to the control of stochastic systems; this constitutes one direction for future work.

We use the following deterministic Markov decision process (MDP) framework to describe the environment of the RL agent. The environment has a state set S, which can be an arbitrary set. The set of goal states is denoted $G \subset S$; the set of non-goal states is denoted $\overline{G}$. In each non-goal state s, k actions are available to the agent. Each action a_j incurs some non-negative cost $c_j(s)$ and results in a successor state $s'_j(s)$.

A policy is a mapping $\pi : \overline{G} \mapsto \{1, \ldots, k\}$ which selects an action based on the current state. For any start state s_0, a policy π induces a trajectory: $s_1 = s'_{\pi(s_0)}(s_0)$, $s_2 = s'_{\pi(s_1)}(s_1)$, etc. A trajectory is either finite, ending at a goal state, or infinite. The value of a policy for a start state, s_0, is the discounted sum of action costs in the trajectory: $V^\pi(s_0) = \sum_i \gamma^i c_{\pi(s_i)}(s_i)$, where $\gamma \in [0,1]$ is the discount rate. We assume there exists a start state distribution S_0 on $\overline{G}$. The value of a policy is defined as its expected value across start states: $V^\pi = E_{s_0 \sim S_0} V^\pi(s_0)$. The goal of an RL agent is to find a policy that minimizes V^π.

Note that an RL agent is not explicitly required to reach a goal state. If this happens, it is as a side effect of minimizing cost. Proper use of Lyapunov functions can ensure that the RL agent reaches G on every trial.

3. Lyapunov Functions

To introduce the main idea of this paper, consider a very simple example. Figure 1a depicts a five by five "gridworld". Each square corresponds to a state of an RL agent's environment. The "S" square represents the start state, and the "G" square represents the goal state. The agent can move to any up-, down-, left-, or right-adjacent square. There is some unknown cost for entering each square.

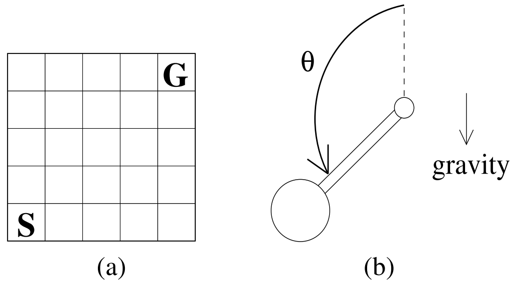

Figure 1. (a) Gridworld for Section 3. (b) Pendulum diagram for Section 4.

Suppose some unspecified RL agent is put into this environment. Knowing nothing about the agent—its learning algorithm, whether it learns on- or off-policy, its method of function approximation, its method of exploration—it is difficult to predict how the agent will behave. It is possible that it will quickly find an optimal path and follow it thereafter. On the other hand, it may not even reach the goal state once.

Now consider the case in which the RL agent is only allowed to move to up- or right-adjacent squares. Then the agent would, by necessity, reach the goal state on every trial in no more (and no less) than 10 time steps. Each action that the agent takes reduces its Manhattan distance to goal by one, leading it eventually to the goal state. The Manhattan distance of the agent to goal is a simple example of a Lyapunov function. This kind of guarantee on goal achievement, deriving from the actions available to RL agent and not the RL algorithm itself, is the main idea we pursue in this paper. But we address more interesting and difficult classes of problems than this simple gridworld.

Lyapunov functions are often thought of as generalized "energy" or "potential" functions. They take large values away from the goal, and decrease monotonically towards the goal. There are many different definitions of "Lyapunov function," depending on the context in which it is applied. We use the following:

Definition *Given a deterministic MDP, a* Lyapunov function *is a function $L : S \mapsto \Re$ such that* (i) $L(s) > 0$ *for all states $s \in \overline{G}$, and* (ii) *there exists $\delta > 0$ such that for all states $s \in \overline{G}$ there exists an action a_j such that either $s'_j(s) \in G$ or $L(s) - L(s'_j(s)) \geq \delta$.*

That is, a Lyapunov function is positive for any non-goal state, and in any non-goal state some action either leads immediately to the goal or to a state at least δ lower on the function. If the second condition holds for all actions, i.e., the "exists an action a_j" is replaced by "for all actions a_j", then we will say the MDP is descending on L. This condition can be achieved by proper selection, or restriction, of the actions available to an agent and has both theoretical and practical benefits. The main theoretical benefit is formalized by the following:

Theorem *If a deterministic MDP is descending on a Lyapunov function L, then any trajectory starting from $s_0 \in \overline{G}$ must enter G in no more than $\lceil L(s_0)/\delta \rceil$ time steps.*

Proof Suppose that for some $s_0 \in \overline{G}$ an agent could perform a sequence of $\lceil L(s_0)/\delta \rceil$ actions and never enter G. Since each action leads to a successor state where L is lower by at least δ, the non-goal state at the end of the action sequence would have L-value no more than $L(s_0) - \delta \lceil L(s_0)/\delta \rceil \leq 0$. But no non-goal states have $L \leq 0$, so we have a contradiction.

A Lyapunov function constitutes a strong form of domain knowledge, and certainly is not always available. In principle, Lyapunov functions exist for any problem in which the goal is reachable (Vincent & Grantham, 1997). Finding a Lyapunov function in analytic form for a given problem is, however, something of an art. Numerous sources discuss methods for finding Lyapunov functions (e.g., Vincent & Grantham, 1997; Krstić et al., 1995). For many important problems and classes of problems, standard Lyapunov functions have already been developed. For example, feedback linearizable systems are easily analyzed by Lyapunov methods. These systems include almost all modern industrial robots (Vincent & Grantham, 1997). At least two major types of Lyapunov functions can be applied to continuous-state path planning problems (Rimon & Koditschek, 1992; Connolly & Grupen, 1993). Other concrete and abstract control problems are discussed in, e.g., Kalman & Bertram (1960), Anderson & Grantham (1989), Krstić et al. (1995), and Vincent & Grantham (1997).

4. Pendulum Swing-up and Balance Task

Boone (1997a; 1997b) and DeJong (2000), among others, have discussed the role of mechanical energy in controlling pendulum-type systems. The standard tasks are either to swing the pendulum's end point above some height (swing-up) or to swing the pendulum up and balance it in a vertical position (swing-up and balance). In either task, the goal states have greater total mechanical energy (potential plus kinetic) than the start state, which is typically the hanging-down rest position. Thus, any controller that solves one of these tasks must somehow impart a certain amount of energy to the system. Our first demonstration problem is a swing-up and balance task for a single-link pendulum.

4.1 Pendulum Problem Definition

Figure 1b depicts the pendulum. Its state is given by its angular position, $\theta \in [-\pi, \pi]$, and its angular velocity, ω. The pendulum's continuous-time dynamics are given by

$$\dot{\theta} = \omega, \quad \dot{\omega} = \sin(\theta) + u,$$

where the sine term is due to the acceleration of gravity, and u is a control torque applied at the fulcrum of the pendulum. These equations assume a mass one, length one pendulum. The magnitude of the control torque is bounded by some $\mu > 0$. We assume μ is relatively small, so that the pendulum cannot be directly driven to the balanced position. Instead, the solution involves swinging the pendulum back and forth repeatedly, pumping energy into it until it can swing upright.

The mechanical energy of the pendulum is $\mathrm{ME}(\theta, \omega) = 1 + \cos(\theta) + \frac{1}{2}\omega^2$. When the pendulum is upright and stationary, ME $= 2.0$. For any other state with ME $= 2.0$, if u is taken to be zero, the pendulum will naturally swing upright and asymptotically approach the vertical, stationary state $(\theta = 0, \omega = 0)$. The pendulum swing-up and balance problem for the single-link pendulum can therefore be reduced to the problem of achieving any state with ME $= 2.0$. We take $G = \{(\theta, \omega) \mid \mathrm{ME}(\theta, \omega) = 2.0\}$. The cost incurred by any action is 1.0, making this a minimum-time control problem.

We define four actions for controlling the pendulum. Each action corresponds to applying a continuous-time control law to the pendulum for a time interval of unit duration, or less if the pendulum reaches G before the interval ends. In the experiments reported below, optimal policies are approximately 15 steps long and random policies take, on average, between 25 and 100 steps to reach G, depending on which action set is used.

The first two actions we define correspond to constant-torque controls laws: a_1 corresponds to the torque $-\mu$ and a_2 corresponds to the torque $+\mu$. The second two actions are based on an "energy-ascent" control law, EA. To a first approximation, EA applies maximum allowed torque in the direction of current pendulum motion. This has the effect of maximizing the time derivative of mechanical energy, which is just $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{ME} = \omega \cdot u$. However, the rule is made somewhat more complex by the need to avoid potential equilibrium points where gravity and $\pm\mu$ torque cancel out:

$$\mathrm{EA}(u, \varepsilon_\theta, \varepsilon_\omega) = \begin{cases} \mathrm{sign}(\theta) u & \text{if } \omega = 0 \\ \frac{1}{2}\mathrm{sign}(\omega) u & \text{if } (0 < \omega \leq \varepsilon_\omega \text{ and} \\ & \quad (\theta \in [\theta_3 - \varepsilon_\theta, \theta_3) \text{ or} \\ & \quad \theta \in [\theta_4 - \varepsilon_\theta, \theta_4) \,)) \\ & \text{or } (-\varepsilon_\omega < \omega < 0 \text{ and} \\ & \quad (\theta \in (\theta_1, \theta_1 + \varepsilon_\theta] \text{ or} \\ & \quad \theta \in (\theta_2, \theta_2 + \varepsilon_\theta] \,)) \\ \mathrm{sign}(\omega) u & \text{else} \end{cases}$$

where $\mathrm{sign}(x) = 1$ if $x \geq 0$ and -1 otherwise, $u > 0$, ε_θ and

ε_ω are small positive constants, and $\theta_1, \ldots, \theta_4$ are equilibrium points of $\pm\mu$ with gravity: $\theta_1 = \arcsin(\mu)$, $\theta_2 = \pi - \theta_1$, $\theta_3 = \theta_1 - \pi$, $\theta_4 = -\theta_1$. The complex condition in the middle case checks for the pendulum having low velocity and being near one of the equilibrium points. When that condition holds, the torque $\frac{1}{2}\mathrm{sign}(\omega)u$ is used instead of $\mathrm{sign}(\omega)u$. This still increases ME, but there is no danger of being stuck approaching an equilibrium point with this smaller torque. Action a_3 corresponds to $\mathrm{EA}(\mu, 0.1, 0.1)$ and a_4 corresponds to $\mathrm{EA}(\frac{1}{2}\mu, 0.1, 0.1)$. Both a_3 and a_4 can be shown to increase the mechanical energy of the pendulum by at least some amount $\delta > 0$ regardless of the pendulum's state. (We omit proof due to space considerations.)

These four actions are organized into three different action sets: $A_{\mathrm{const}} = \{a_1, a_2\}$, $A_{\mathrm{EA}} = \{a_3, a_4\}$, and $A_{\mathrm{all}} = \{a_1, a_2, a_3, a_4\}$. Because a_3 and a_4 both cause a δ step up on ME, the function $L(\theta, \omega) = 2 - \mathrm{ME}(\theta, \omega)$ defines a Lyapunov function for the MDP induced by those two actions. Further, that MDP is descending on L, so the theorem of Section 3 applies, ensuring that an agent restricted to use these actions will always reach G. There is no such guarantee for agents using either of the other two action sets, although one might expect A_{all} to bias an RL agent towards the goal because it includes the two EA-based actions.

4.2 Q-Learning Experiments on the Pendulum

To represent action-values, we discretized the state space region $[-\pi, \pi]^2$ into uniform bins by splitting both dimensions into B intervals. We report results for $B = 2^0, 2^1, 2^2, \ldots, 2^8$. The total number of bins thus ranged from 1 to $2^{16} = 65{,}536$. For each bin and each action, we maintained an action-value. For each choice of B and each action set, we ran 30 independent runs of Q-learning (Watkins, 1989). For the kth update of an action value, we used the learning rate parameter $1/\sqrt{k}$. We found this outperformed various constant learning rates as well as the choice $1/k$. Each run contained 10,000 learning trials. The start state for every trial was $(\theta, \omega) = (-\pi, 0)$. Action selection was $\varepsilon-$greedy with $\varepsilon = 0.1$, and ties were broken randomly (Sutton & Barto, 1998). Trials were terminated if they took longer than 900 time steps. After each learning trial we performed a test trial, controlling the pendulum according to the current greedy policy with no exploration and no learning.

The results we report correspond to a choice of $\mu = 0.224$. We obtained qualitatively similar results for other choices of μ, but this choice magnified the differences between the results for the three action sets, making comparisons easier. The discount rate, γ, was 1.

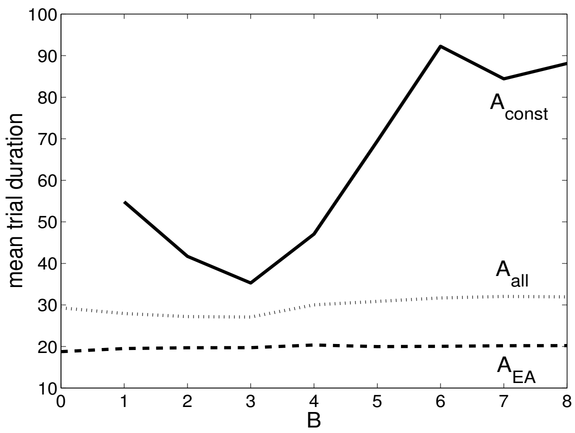

Figure 2. Early learning performance on the pendulum

4.3 Pendulum Results

One of the primary effects of having the energy-based actions a_3 and a_4 was to improve the initial performance of the learner. Figure 2 displays the mean number of steps to goal (equivalently, total cost incurred) during the first 10 learning trials, averaged over the 30 independent runs. Learning trials were significantly shorter for agents with action sets A_{EA} or A_{all} than for agents that with A_{const}. The energy-based actions strongly biased these agents to reach the goal. Agents that used A_{EA}, which guaranteed that the mechanical energy increased on every step, further outperformed the A_{all} agents. These effects were strongest at the higher values of B. When the state space was divided into a large number of bins, all the agents behaved nearly randomly because of the random tie-breaking rule and the improbability of revisiting bins. Trial times under A_{EA} were practically unaffected by the state space binning resolution. Note that there is no data point for A_{const} at $B = 0$, because almost all of those learning trials timed out. Using only the policies "$-\mu$ always" and "$+\mu$ always", it is virtually impossible swing up the pendulum, except by a lucky sequence of random exploratory actions.

Figure 3 displays the final performance under the different action sets, as measured by the mean duration of the last 100 test trials. One clear trend is that learning was most successful at the intermediate resolutions, around $B = 5$ or 6. There, and at $B = 4$ and $B = 7$ as well, A_{const} agents performed the best. The final performances for the three action sets were: 14.51 ± 0.00 for A_{const} at $B = 5$, 14.65 ± 0.05 for A_{all} at $B = 5$, and 15.44 ± 0.01 for A_{EA} at $B = 6$. It appears that the restriction of A_{EA} to descend on the Lyapunov function does limit performance. No A_{EA} run was shorter than 15 time steps. That A_{all} did not do as well as A_{const} was surprising, since the former can represent any policy that the latter can. The difference is slight, however, for intermediate values of B. At

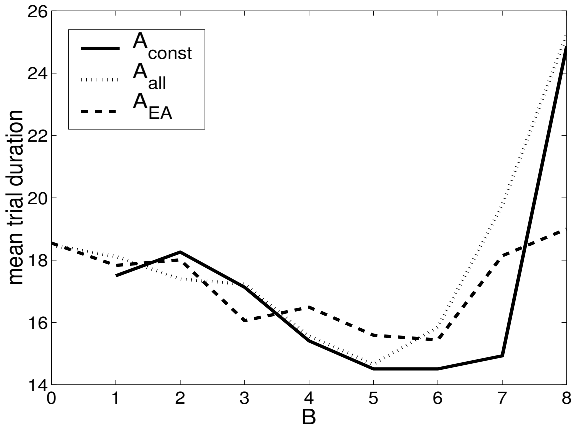

Figure 3. Final test performance on the pendulum

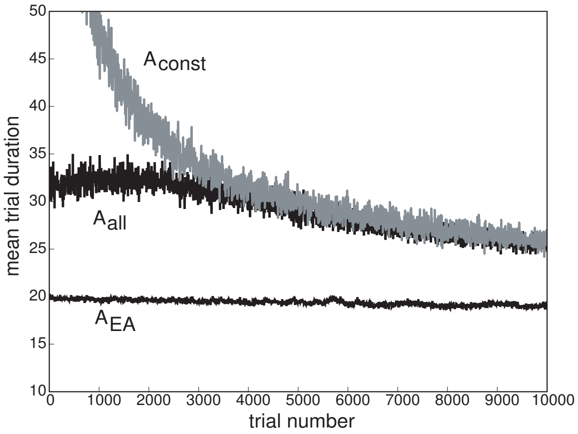

Figure 4. Learning curves for $B = 8$ on the pendulum

higher values of B, learning had not asymptoted by the end of the 10,000 trials, as is seen clearly in Figure 4. There are $2^{2B} \times$ (Number of actions) free parameters in the action value functions, so it is unsurprising that learning was slow for high B.

Lyapunov-based constraints made a big difference in the worst-case behavior of the learning agents. Over all runs, the longest learning trial under $\mathrm{A_{EA}}$ only took 33 time steps, compared to 136 time steps for $\mathrm{A_{all}}$ and 547 time steps for $\mathrm{A_{const}}$ $(B \geq 1)$. Under $\mathrm{A_{all}}$ and $\mathrm{A_{const}}$, the longest test trials timed out after 900 time steps. Thus, Lyapunov-based constraints can play an important role in keeping trials short and in maintaining performance during learning.

5. Robot Arm Control

The second problem we consider is controlling a simulated 3-link robot arm, depicted in Figure 5. The arm is a special case of a feedback linearizable system, an important class

link	length	mass
1	0.1	1.0
2	1.0	1.0
3	1.0	1.0

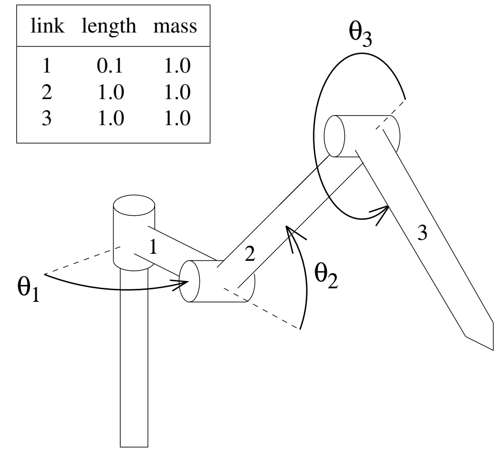

Figure 5. Diagram of the 3-link arm

of systems to which Lyapunov methods are applicable.

5.1 Arm Problem Definition

The state space of the arm is $\Re^6$, corresponding to three angular joint positions and three angular velocities. We denote the joint position and joint velocity column vectors by Θ and $\dot{\Theta}$. We define $G = \{s \in S : \|s\| \leq 0.01\}$. This is a set of low-velocity states near to a fully-extended horizontal configuration. The start state distribution is uniform over the nine configurations $\Theta_0 = (x, y, -y)^T$ for $x \in \{-\pi, -2\pi/3, -\pi/3\}$ and $y \in \{-\pi/2, 0, +\pi/2\}$, with zero velocity.

The dynamics of the arm are modeled by:

$$\frac{\mathrm{d}}{\mathrm{dt}} \begin{bmatrix} \Theta \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} \dot{\Theta} \\ H^{-1}(\Theta, \dot{\Theta})(\tau - V(\Theta, \dot{\Theta}) - \mathcal{G}(\Theta)) \end{bmatrix},$$

where H is the inertia matrix, τ is the vector of actuator torques applied at the joints, V is a vector modeling Coriolis and other velocity-dependent forces, and $\mathcal{G}$ is a vector modeling gravitational forces (Craig, 1989).

As with the pendulum, each of the RL agents' actions corresponds to running a continuous-time controller for a time interval of unit duration or until the arm reaches G. This makes optimal trajectories around 10 steps long under the actions we define below. The continuous-time controllers for the arm are based on feedback linearization and linear-quadratic regulator (LQR) methods (e.g., Vincent & Grantham, 1997).

A simple approach to feedback linearization in the robot manipulator case is to reparameterize the control torque in terms of a vector u so that $\tau = Hu + V + \mathcal{G}$. This puts the robot dynamics into the linear form $\frac{\mathrm{d}}{\mathrm{dt}}[{\Theta \atop \dot{\Theta}}] = [{\dot{\Theta} \atop u}]$. Once the dynamics are expressed in linear form, LQR design methods are applicable. LQR design produces continuous-time controllers for the arm, and also provides the Lyapunov function which constrains one of the action sets we define below.

LQR methods result in a linear control law of the form $u = -M[\begin{smallmatrix}\Theta-\Theta_0 \\ \dot{\Theta}\end{smallmatrix}]$, where Θ_0 is a desired target configuration. If the control law is applied indefinitely, it causes the state to asymptotically approach the target position. The matrix M results from the numerical solution of a matrix equation. We compute LQR controllers to minimize the cost functional $\int_{t=0}^{\infty} ||\Theta(t)||^2 + ||u||^2 dt$, where $\Theta(t)$ is the position component of the trajectory generated by the LQR controller from any starting configuration $\Theta(0)$. The resulting matrix M does not depend on the starting configuration. We define five LQR-based controllers:

C_1 An LQR controller for the feedback linearized arm with $\Theta = (0,0,0)$ as the target configuration. This configuration is within G, so C_1 alone is able to bring the arm into G from any initial configuration.

C_2 An LQR controller with target configuration $\Theta = (0,0,0)$ except that u is multiplied by 2. This tends to cause larger control torques that bring the arm towards the target configuration faster.

C_3 An LQR controller with target configuration $\Theta = (0,0,0)$ except that u is multiplied by 0.5. This tends to cause smaller control torques that bring the arm towards the target configuration more slowly.

C_4 An LQR controller that regulates towards the target configuration $(\Theta_1, \pi/4, -\pi/2)$ where Θ_1 is the current position of joint 1. This tends to fold the arm in towards the center, so it can swing around the first joint more easily.

C_5 Similar to C_4, but regulating the second two joints to the configuration $(\Theta_2, \Theta_3) = (\pi/2, -\pi)$.

The first action set for the arm, $\mathrm{A_{LQR}}$, has five actions, corresponding to the five controllers above.

A benefit of LQR design is that it produces a Lyapunov function as a side effect. Associated with each LQR controller is a quadratic Lyapunov function for the problem of getting to a small region around the controller's target configuration. In general, this takes the form $[(\Theta - \Theta_0)\ \dot{\Theta}]W[(\Theta - \Theta_0)\ \dot{\Theta}]^T$, where W is a symmetric positive definite matrix. Since C_1's target point is within G, we can use its Lyapunov function, $L = [\Theta\ \dot{\Theta}]W_1[\Theta\ \dot{\Theta}]^T$, to design a second action set.

Control law C_1 causes the arm to descend continuously on L. It is easy to show that it produces $\frac{\mathrm{d}}{\mathrm{dt}}L \leq -\delta < 0$ for some δ in any non-goal state (Vincent & Grantham, 1997). However, the other control laws do not necessarily descend on L. To produce a set of controllers that descend on L, we use the following transformation. For any controller $C_j,\ j \in \{2,3,4,5\}$ we define the L-descending version of the controller to be the one that chooses u as:

$$u = \begin{cases} C_j(s) & \text{if } L \geq 0.1 \text{ and that choice} \\ & \text{produces } \frac{\mathrm{d}}{\mathrm{dt}}L \leq -0.1 \\ C_1(s) & \text{otherwise.} \end{cases}$$

In other words, if controller C_j is not descending fast enough (or at all) on L or if the arm is already near the goal, then the control choice of C_1 is supplied instead. The second action set for the arm, $\mathrm{A_{L\text{-}desc}}$, contains five actions, corresponding to C_1 and the L-descending versions of $C_2, \ldots, C_5$. $\mathrm{A_{L\text{-}desc}}$ induces an MDP which is descending on L, and thus satisfies the theorem in Section 3.

When an action is selected, the cost incurred is the continuous-time integral, until the next action choice, of the quantity $c = ||\Theta(t)||^2 + ||\tau(t) - \tau_0||^2$. τ_0 is the torque needed to hold the arm steady in the horizontal configuration. This is a standard cost function used in control theory. The first term penalizes the agent to the extent that the arm is away from the goal configuration. The second term penalizes the use of high torques, not counting the torque τ_0, which is unavoidable. This is not a minimum-time cost function. It also differs from the cost function used in the LQR design because it penalizes $\tau - \tau_0$ instead of u. This cost function comes closer to penalizing the whole control effort applied, rather than just penalizing the control effort applied to the feedback linearized system.

5.2 Q-Learning Experiments on the Arm

Due to the choice of start states and G, we can restrict attention to the region of state space: $\Theta \in [-4,1] \times [-2,2]^2$, $\dot{\Theta} \in [-3,3]^3$. To represent action values, the region was divided into bins, with B equal intervals along each dimension. We report results for B =1, 3, 5, 7, 9, and 11. The total number of bins thus ranged between 1 and $11^6 = 1,771,561$. For each bin we maintained action-values, which were updated according to the Q-learning update rules (Watkins, 1989). For each choice of B and both action sets, we performed 20 independent runs of 10,000 learning trials. Action selection was $\varepsilon-$greedy with $\varepsilon = 0.1$. Trials were terminated if they took longer than 500 time steps. After every 25th learning trial, a test trial was run from each of the nine starting points, to evaluate the current policy. The learning rate parameter for the kth update of an action-value was $1/\sqrt{k}$. The discount rate, γ, was 1.

5.3 Arm Results

Figure 6 displays the mean cost of the first 100 learning trials for the two action sets, averaged across the 20 inde-

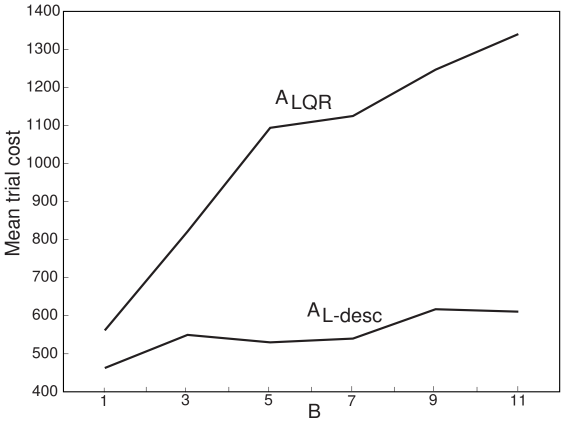

Figure 6. Early learning performance on the arm

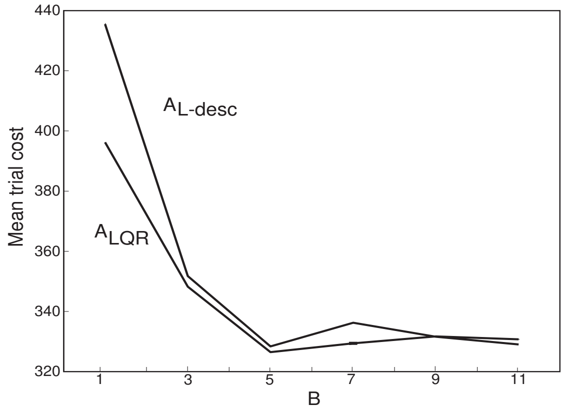

Figure 7. Final test performance on the arm

pendent runs. $A_{L\text{-desc}}$ produced significantly better initial performance than A_{LQR}. Under either action set, performance decreased significantly as the resolution of the state space binning increases. These observations are similar to the results obtained with the pendulum.

Figure 7 shows the value of the final policies discovered under each action set, as measured by the mean trial cost during the last 10 testing periods. There was little difference in asymptotic performance under the two action sets, except at $B = 1$. It appears that good policies descend on L, so that a restriction to descend on L costs little or nothing in terms of long-term performance. This is plausible, considering that the $||\Theta||^2$ component of action costs penalizes movement away from G.

Although final performance was good for both action sets, performance under A_{LQR} was sometimes quite poor during learning. The longest learning trial took 244 time steps and incurred a cost of 15,421. Some test trials timed out. By comparison, the worst learning or test trial under $A_{L\text{-desc}}$ took only 15 time steps and incurred a cost of 2,644; both

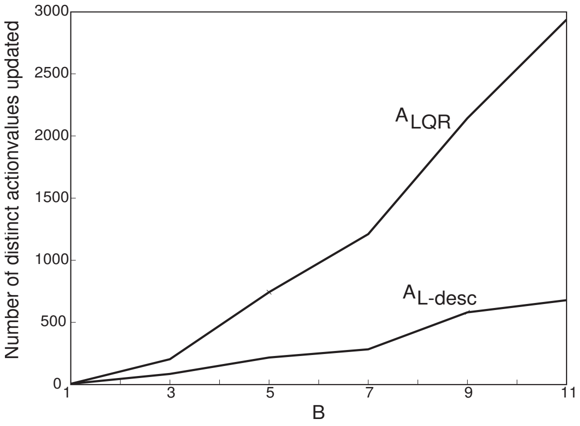

Figure 8. Mean memory usage for arm learning

statistics are an order of magnitude better than the figures for A_{LQR}. As with the pendulum, Lyapunov constraints played an important role in maintaining good performance during learning.

Another benefit of the Lyapunov-based actions was that they focused learning on a much smaller portion of state space than was explored by agents using action set A_{LQR}. Figure 8 plots the mean number of distinct action-values updated by the end of 10,000 learning trials. At most choices of B, the $A_{L\text{-desc}}$ agents updated approximately one-fifth as many distinct action-values as agents using A_{LQR}. This effect was not observed in the pendulum experiments. Reducing the portion of the state-action space explored by learning can have important implications if one uses function approximators with memory requirements that are proportional to this factor. For example, in a simple binned state space representation, one can save memory by allocating space for an action value only when it is updated for the first time. Reducing the number of states and actions explored by an agent can increase memory savings, allowing finer resolution representations or permitting easier scaling to higher dimensional problems.

6. Conclusion

Lyapunov methods allow the design of controllers that are guaranteed to bring a system to a goal state. However, Lyapunov methods do not directly address the cost of controlling the system. By formulating the actions available to an RL agent so that they descend on a Lyapunov function, one can use learning to minimize costs while retaining the strong theoretical and practical benefits of Lyapunov methods. In theory, one can guarantee that the RL agent reaches a goal state on every trial. In the two examples presented in this paper, we illustrated how Lyapunov-based actions can greatly improve a Q-learner's initial performance and help maintain good performance during learning.

Manipulating the action formulation of an RL agent is certainly not the only way to use Lyapunov domain knowledge. Ng et al. (1999) proposed a general method for changing the reward function of an MDP to encourage movement toward a goal as measured by some "potential" function. Lyapunov functions seem excellent candidates for the potential functions in such a scheme. Combining our approach with theirs might yield even better results. Alternatively, in a Bayesian framework such as the one proposed by Dearden et al. (1998), Lyapunov domain knowledge might be used to set priors about the values of actions or about which actions are more likely to be optimal ones.

Acknowledgments

This work was funded by the National Science Foundation under Grant Nos. ECS-9980062 and ECS-0070102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Theodore Perkins is also supported by a graduate fellowship from the University of Massachusetts Amherst. We thank Doina Precup and Michael Rosenstein for comments on earlier drafts of this paper.

References

Anderson, M. J., & Grantham, W. J. (1989). Lyapunov optimal feedback control of a nonlinear inverted pendulum. *Journal of Dynamic Systems, Measurement, and Control*.

Boone, G. (1997a). Efficient reinforcement learning: Model-based acrobot control. *1997 International Conference on Robotics and Automation* (pp. 229–234).

Boone, G. (1997b). Minimum-time control of the acrobot. *1997 International Conference on Robotics and Automation* (pp. 3281–3287).

Clouse, J. A. (1997). *On integrating apprentice learning and reinforcement learning*. Doctoral dissertation, University of Massachusetts Amherst.

Connolly, C. I., & Grupen, R. A. (1993). The applications of harmonic functions to robotics. *Journal of Robotics Systems*, *10*, 931–946.

Craig, J. J. (1989). *Introduction to robotics: Mechanics and control*. Addison-Wesley.

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian q-learning. *Proc. AAAI-98, Madison, Wisconsin*. AAAI Press.

DeJong, G. (2000). Hidden strengths and limitations: An empirical investigation of reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 215–222). Morgan Kaufmann.

Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Kalman, R. E., & Bertram, J. E. (1960). Control system analysis and design via the "second method" of lyapunov ii: Discrete-time systems. *Transactions of the ASME: Journal of Basic Engineering*, 394–400.

Krstić, M., Kanellakopoulos, I., & Kokotović, P. (1995). *Nonlinear and adaptive control design*. New York: John Wiley & Sons, Inc.

Maclin, R., & Shavlik, J. W. (1996). Creating advice-taking reinforcement learners. *Machine Learning*, *22*, 251–281.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 278–287). Morgan Kaufmann.

Parr, R. (1998). *Hierarchical control and learning for markov decision processes*. Doctoral dissertation, University of California, Computer Science Division, Berkely.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. Doctoral dissertation, University of Massachusetts Amherst.

Rimon, E., & Koditschek, D. E. (1992). Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, *8*, 501–517.

Singh, S. P., Barto, A. G., Grupen, R., & Connolly, C. (1994). Robust reinforcement learning in motion planning. *Advances in Neural Information Processing Systems 6* (pp. 655–662). Cambridge, MA: MIT Press.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1044). Cambridge, MA: MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, Massachusetts: MIT Press/Bradford Books.

Tesauro, G. J. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, *6*, 215–219.

Vincent, T. L., & Grantham, W. J. (1997). *Nonlinear and optimal control systems*. New York: John Wiley & Sons, Inc.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral dissertation, Cambridge University, Cambridge, England.