Journal of Machine Learning Research 3 (2002) 803-832 Submitted 12/01; Published 12/02

Lyapunov Design for Safe 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, USA

Editors: Carla E. Brodley and Andrea Danyluk

Abstract

Lyapunov design methods are used widely in control engineering to design controllers that achieve qualitative objectives, such as stabilizing a system or maintaining a system's state in a desired operating range. We propose a method for constructing safe, reliable reinforcement learning agents based on Lyapunov design principles. In our approach, an agent learns to control a system by switching among a number of given, base-level controllers. These controllers are designed using Lyapunov domain knowledge so that *any* switching policy is safe and enjoys basic performance guarantees. Our approach thus ensures qualitatively satisfactory agent behavior for virtually any reinforcement learning algorithm and at all times, including while the agent is learning and taking exploratory actions. We demonstrate the process of designing safe agents for four different control problems. In simulation experiments, we find that our theoretically motivated designs also enjoy a number of practical benefits, including reasonable performance initially and throughout learning, and accelerated learning.

Keywords: Reinforcement Learning, Lyapunov Functions, Safety, Stability

1. Introduction

Practitioners of artificial intelligence are paying increasing attention to the reliability and safety of approximate solutions to sequential decision problems (Singh et al., 1994; Weld and Etzioni, 1994; Schneider, 1997; Neuneier and Mihatsch, 1999; Gordon, 2000; Perkins and Barto, 2001a,b). These issues are particularly important for learning agents. Such agents may come to behave in ways not expected by their designers, and the process of learning and exploring alternative behaviors may itself be costly or dangerous. Where safety is not a concern or is easily achieved, reinforcement learning techniques have generated impressive solutions to difficult, large-scale stochastic decision problems. Examples include such diverse problems as backgammon playing (Tesauro 1994, 1995), job-shop scheduling (Zhang and Dietterich, 1996), elevator dispatching (Crites and Barto, 1998), and option pricing (Tsitsiklis and Van Roy, 1999). However, there are obstacles to applying reinforcement learning to problems in which it is important to ensure reasonable system performance and/or respect safety constraints at all times. Most reinforcement learning algorithms offer no guarantees on the quality of control during learning. In practice, the initial performance of learning systems is often poor. Even after the agent has started to perform well, exploratory actions, which are necessary for learning, can degrade performance. For many algorithms, performance does not always improve monotonically as learning progresses (Bertsekas and Tsitsiklis, 1996). If learning stops at some point, and the agent begins behaving according to a fixed policy, then exploratory actions and learning errors cease to threaten the agent's performance. However, one still may not know if the behavior that the agent has learned is safe and successful in all possible situations. Depending on the precise criteria used, this may be difficult or impossible (undecidable) to determine.

Safety and reliability have always been important issues in control engineering. Often, the problems studied in control engineering are not formulated as optimal control problems. Commonly stated goals for controlling a system include: stabilizing a system around some operating point, tracking a desired system trajectory, or keeping the system's state in some range (see, for example, Sontag, 1990; Levine, 1996; and Vincent and Grantham, 1997, for general treatments and Craig, 1989, for a discussion of robotics problems in particular). For a number of individual problems and abstract classes of problems, provably correct solutions have been developed for achieving qualitative goals such as these. Of course, the number of problems that have yielded to analysis is not as large as one would like. And in implementing such a solution, one must worry about whether the real system to be controlled behaves enough like the abstract problem that the solution is valid. Still, analytical solutions to qualitative formulations of control problems have been invaluable in a wide array of disciplines, including robotics, networking, and numerous branches of engineering.

In this paper we show that by using qualitative control technologies in a reinforcement learning framework, one can design *qualitatively safe and reliable* agents that can learn approximate solutions to *optimal* control problems. We find that appropriate use of qualitative domain knowledge greatly improves the initial performance of a learning system; ensures reasonable performance throughout learning; and extends the range of problems that can reasonably be solved to include, for example, problems requiring high-precision control.

The primary qualitative technique we use is Lyapunov control design. Lyapunov functions are a fundamental tool used in control theory, and other disciplines, for studying the stability of systems evolving through time. Intuitively, a Lyapunov function is a scalar function of the system state on which system trajectories descend monotonically. Often, a Lyapunov function is thought of as a kind of "energy" which the system continually dissipates. In physical systems, a Lyapunov function may correspond to some real, natural notion of energy. For example, a ball rolling on a horizontal plane subject to friction is continually sapped of its kinetic energy, slowing the ball down. A mechanical wristwatch winds down as the potential energy stored in its spring is expended in driving the motion of the hands. In other cases, a Lyapunov function may be more abstract, such as an error or distance measure between the system state and a desired system state or trajectory. A Lyapunov function can be viewed as a form of domain knowledge that qualitatively describes certain aspects of the dynamics of the system. We know, for example, that the rolling ball and the wristwatch will eventually slow to a halt, though we may not know precisely where or when.

The notion of a control Lyapunov function generalizes the Lyapunov function concept to systems that receive control inputs. A control Lyapunov function is a scalar function of the system state on which some control inputs cause trajectories that descend. Intuitively, our main proposal is to achieve safe, reliable reinforcement learning control by constraining the action choices of the agent so that *all* actions cause the system to descend on an appropriate control Lyapunov function. Qualitative performance guarantees follow directly from the descent constraint. We do not need to make any assumptions about the internals of the learning agent. It may be actively learning to control the system, choosing actions randomly, or simply executing a previously-learned policy. Thus, our approach is consistent with virtually any choice of reinforcement learning algorithm.

Our approach is primarily applicable to the same kinds of problems to which Lyapunov design applies: stabilization, regulation, and tracking problems. In other words, our techniques are relevant to problems in which the agent's task is to optimally bring the state of its environment to some region of state space, possibly a goal region, to maintain the state there, or to cause the state to follow some desired path. For a given optimal control problem of one of these types, finding an appropriate Lyapunov function may not be an easy task. Further, it may be that in restricting the agent to behave in ways that are provably qualitatively satisfactory, we may be ruling out other qualitatively satisfactory behaviors that are superior according to the cost function of the optimal control problem. Nevertheless, we expect that when Lyapunov domain knowledge is available, the theoretical assurances we gain and the ability to readily solve new classes of problems will outweigh such drawbacks.

In Section 2 we define the Markov decision process framework that we use to formulate optimal control problems. We also describe a number of qualitative safety and reliability properties that can be established using Lyapunov-based action constraints. Section 3 begins with background on Lyapunov functions and concludes with definitions and theorems necessary for establishing qualitative system properties. Following this is a sequence of four example problems in which we demonstrate Lyapunov analysis, demonstrate the process of designing learning agents based on Lyapunov domain knowledge, and present empirical evaluations of learning both with and without Lyapunov domain knowledge. In Section 7 we describe related work, and in Section 8 we conclude and discuss directions for future work.

2. Markov Decision Processes and Qualitative Solution Properties

We model the agent's environment as a Markov decision process (MDP), evolving on an arbitrary state set S. The state of the environment at time $t \in \{0,1,2,\ldots\}$ is denoted by s_t. The initial state, s_0, is determined according to an initial state distribution S_0 (or probability measure, if one prefers). At each time t, the agent chooses an action, a_t, from the set of allowed actions, $A(s_t)$. The immediate cost incurred, $c_t \in \mathbb{R}$, and the next state of the environment, $s_{t+1} \in S$, are determined stochastically according to a joint distribution depending on the state-action pair (s_t, a_t).

In some MDPs there is a set of goal states, $G \subset S$. When the environment enters a goal state, the agent incurs a terminal cost but no further costs are incurred and no further actions are taken. As far as the optimal control problem is concerned, we say that the trial, or trajectory, ends once a goal state is reached. However, real systems do not necessarily stop once a goal state is reached. For example, once a robot arm is moved to a goal configuration, we may consider the optimal control task to be completed. However, in practice we may be concerned with whether the arm stays in the goal configuration. In our demonstration problems, and for the qualitative properties described below, it is sometimes useful to consider what happens to the system after it reaches a goal state.

A stochastic policy π maps each state $s \in S$ to a distribution over $A(s)$. The expected (un)-discounted return of policy π is defined to be

$$V^{\pi} = \lim_{\tau \to \infty} E\left\{ \sum_{t=0}^{\tau} \gamma^t c_t \right\},$$

where the expectation is with respect to the initial state distribution, the stochastic transitions of the environment, and the stochastic action selections of the policy. The factor γ is a discount rate in the range [0,1]; in the case $\gamma = 1$, V^{π} is referred to as the undiscounted return. The agent's task is to find a policy that minimizes the expected discounted return.

In general, we cannot expect a reinforcement learning controller to obtain an optimal or even near optimal policy to an optimal control problem. During learning, suboptimal behavior is a given. Yet in some settings, it may be important for us to ensure reasonable performance at all times. For example, we may want to know that excessive costs will not be incurred or that safety constraints will not be violated. Below, we define a number of qualitative system properties that can be used to express safety or performance objectives.

Let $T \subset S$. One of the simplest questions we may ask is, "Does the agent keep the environment in set T, if it starts there?" If T is taken to be a set of "safe" states, then this property can be used to express a safety constraint that we want the controller to satisfy. Such a property may hold for all possible trajectories of the system, or for a probability one[1] set of system trajectories.

Property 1 (Remain in T) *For certain/with probability 1, if $s_0 \in T$, then for all $t \in \{1,2,\ldots\}$, $s_t \in T$.*

Another natural question is, "Does the controller cause the system to enter T from any initial state?" When $T = G$, this property expresses a guarantee of goal achievement.

Property 2 (Reach T) *For certain/with probability 1, there exists $t \geq 0$ such that $s_t \in T$.*

Combining the two previous properties expresses the property that the agent is guaranteed to bring the environment to a part of state space and keep it there.

Property 3 (Reach and remain in T) *For certain/with probability 1, there exists $\tau \in \{0,1,2,\ldots\}$ such that for all $t \geq \tau$, $s_t \in T$.*

For some MDPs it is impossible to maintain the state in a given set T. A less restrictive requirement is that the environment spend an infinite amount of time in T—always returning to T if it leaves.

Property 4 (Infinite time in T) *For certain/with probability 1, for infinitely many distinct $t \in \{0,1,2,\ldots\}$, $s_t \in T$.*

The final property we describe is convergence to T. For this we require a distance-to-T function $\delta_T : S \to \mathbb{R}^+$. At the least, this function should satisfy $\delta_T(s) = 0$ for $s \in T$ and $\delta_T(s) > 0$ for $s \notin T$.

Property 5 (Converge to T) *For certain/with probability 1, $\lim_{t\to\infty} \delta_T(s_t) = 0$.*

Questions such as the first four are often addressed in the model-checking literature. Indeed, Gordon (2000) has demonstrated the relevance of model-checking techniques for verifying learning systems. In her approach, policy changes recommended by a learning component of an agent are verified for safety by a model-checking component. Only if the changes are safe are they implemented. Model-checking techniques are most successful in finite-state domains. It remains to be seen whether such approaches can be usefully applied to the continuous, infinite state-set problems upon which we focus.

Property 5 is often studied in control theory, typically taking T to contain a single state. Since our MDP framework allows for problems with infinite state sets, all of these properties are undecidable. (See Blondel and Tsitsiklis, 2000, for a survey of complexity results from a control theory perspective. Interestingly, even quite innocent-looking finite-dimensional continuous-state systems can implement Turing machine computations.) Our situation differs from the usual control theory setting because of the presence of the learning agent acting on the system. Nevertheless, control-theoretic techniques can be useful in establishing the qualitative properties listed above.

1. By "probability one", we mean probability one with respect to the initial state distribution, the stochastic transitions of the environment, and the potentially-stochastic action selections of the reinforcement learning agent.

3. Lyapunov Functions and Design

Lyapunov's so-called "second" method was originally proposed as a tool for studying the stability of systems of differential equations. The central task is to identify a *Lyapunov function* for the system—a scalar function of the system's state with negative derivative along system trajectories. Identifying a Lyapunov function for a system establishes that the system is stable. Lyapunov functions are used in control theory to validate control strategies by showing that the resulting behavior of the controlled system is stable. The vast majority of the applications are to problems whose dynamics are modeled by systems of differential equations. Examples include robot manipulator control (Craig, 1989), robot motion planning (Rimon and Koditschek, 1992; Connolly and Grupen, 1993), coordination of distributed agents (for example, Narendra et al., 2001), orientation control of ships, airplanes, and spacecraft; stability of networks and queueing systems; and chemical process control (see, for example, Levine, 1996, for further references). In artificial intelligence, perhaps the best-known applications of Lyapunov functions are in showing the stability of neural network computations (Grossberg and Cohen, 1983; Hopfield and Tank, 1985).

The basic idea of a Lyapunov function has been extended beyond systems of differential equations to quite general settings, including discrete-time deterministic control problems (Kalman and Bertram, 1960b) and Markovian systems with general state sets evolving stochastically in discrete time (Meyn and Tweedie, 1993). We must further extend these approaches because of the reinforcement learning agent in the control loop. We make no assumptions about how the agent chooses actions; we treat the agent as a black box that non-deterministically selects actions in a manner unknown to us. Thus, we must ensure reasonable performance for any possible behavior of the agent. In the Section 3.1, we present and prove two Lyapunov-style theorems that provide guidance for designing reinforcement learning agents. These general theorems, along with specific domain knowledge in the form of a Lyapunov function, can be used to establish qualitative safety and performance guarantees of the types described above.

3.1 Lyapunov-Style Theorems for Reinforcement Learning Agents

Let $T \subset S$ and let $L : S \to \mathbb{R}$ be a function that is positive on $T^c = S - T$. We use s' to denote any state that may result when the state of the environment is s and the agent chooses an action $a \in A(s)$. Let $\Delta > 0$ be a fixed real number.

Theorem 1 *If for all $s \notin T$, all $a \in A(s)$, and all possible next states s', either $s' \in T$ or $L(s) - L(s') \geq \Delta$, then from any $s_t \notin T$, the environment enters T within $k = \lceil L(s_t)/\Delta \rceil$ time steps.*

To paraphrase, if on every time step the environment either enters T or ends up in a state at least Δ lower on the Lyapunov function L, regardless of the agent's choice of action, then the environment enters T in bounded time.

Proof: Suppose the environment is in state s_t at time t and that the agent chooses a sequence of $k = \lceil L(s_t)/\Delta \rceil$ actions without the environment entering T. At each successive time step, the environment enters a state at least Δ lower on L than at the previous time step. After k time steps, we have $L(s_{t+k}) \leq L(s_t) - k\Delta = L(s_t) - \lceil L(s_t)/\Delta \rceil \Delta \leq L(s_t) - L(s_t) = 0$. But L is positive for all states not in T, so $s_{t+k} \in T$, contradicting the assumption that the environment does not enter T by that time. ■

In stochastic systems, guaranteeing descent can be unrealistic. However, if there is always some probability of descending, then intuitively there is some "attraction" towards the set T. If L can be bounded above, this attraction is sufficient to ensure reaching T eventually. An interpretation of the upper bound is that the system never gets "too far away" from T. Let Δ, p, and q be positive reals.

Theorem 2 *If $\sup_{s \notin T} L(s) = U \in \mathbb{R}$, and if, on every timestep t for which $s_t \notin T$, with probability at least q (independent of any prior events) the agent chooses an action for which either $s' \in T$ or $L(s) - L(s') \geq \Delta$ holds with probability at least p, then from any $s_t \notin T$ the environment enters T eventually with probability 1. The probability that the environment does not enter T within k time steps is bounded above by a function that decays exponentially to zero in k.*

Proof: At each time step, there is probability at least $pq > 0$ that the property $\mathcal{P} \equiv s' \in T$ or $L(s) - L(s') \geq \Delta$ holds. Let $n = \lceil U/\Delta \rceil$. If $\mathcal{P}$ holds for n steps in a row, then the environment certainly enters T, by an argument similar to that in the proof of Theorem 1. The probability that $\mathcal{P}$ holds for n steps in a row is at least $p^n q^n > 0$. Thus, the probability that $\mathcal{P}$ fails to hold during at least one of the first n steps after t is no more than $(1 - p^n q^n) < 1$. The probability that $\mathcal{P}$ fails to hold during at least one of the first n steps after t and at least one of the second n steps after t is no more than $(1 - p^n q^n)^2$. More generally, the probability that $\mathcal{P}$ fails to hold for at least one time step in each of the first m blocks of n time steps after t is no more than $(1 - p^n q^n)^m$. This establishes the second claim. The first claim follows since the probability that $\mathcal{P}$ never holds for n time steps in a row is no more than $\lim_{m\to\infty}(1 - p^n q^n)^m = 0$. ■

3.2 Lyapunov Design for Reinforcement Learning Agents

The general approach we propose is to identify a candidate Lyapunov function for a given control problem and then *design* or *restrict* the action choices of the agent so that one of the theorems above applies. In turn, these theorems allow us to establish the properties described in the previous section. The details of how this is done, and the sorts of safety and reliability properties that can be established, depend on the domain. For example, consider a goal-based task for which one can construct a set of actions and a Lyapunov function, L, that satisfy Theorem 1. Then, from any initial state s_0, one is certain that the agent will bring the environment to G (the Reach property, with $T = G$), and will do so within $\lceil L(s_0)/\Delta \rceil$ time steps. In doing so, the environment will also Remain (Property 1) in the set $\{s : L(s) \leq L(s_0)\}$, which may represent a notion of safety; depending on the initial state of the environment, the trajectory produced cannot deviate too far from the goal. If the environment is stochastic, and only the conditions of Theorem 2 can be met, then reaching G can be guaranteed with probability one. However, no absolute bound can be put on the time at which the agent brings the environment to the goal. Below, we demonstrate the general approach we propose in four different domains, using Lyapunov domain knowledge to design reinforcement learning agents and to establish performance guarantees.

4. Deterministic Pendulum Swing-Up and Balance

Pendulum control tasks have been used as demonstration problems in theoretical control research for many years, in part because pendulum dynamics are simply stated yet highly nonlinear. Many researchers have discussed the role of mechanical energy in controlling pendulum-type systems (for example, Spong, 1995; Boone, 1997a,b; DeJong, 2000; Perkins and Barto, 2001a,b). The standard

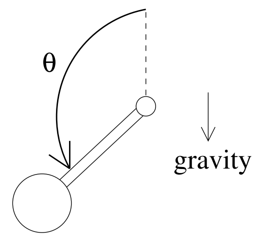


Figure 1: A single-link pendulum.

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 than the initial 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. We first demonstrate our approach on a deterministic pendulum swing-up and balance task.

4.1 Problem Definition and Controllers

Figure 1 depicts the pendulum. The state of the pendulum is specified by an angular position, θ, which is measured counter-clockwise from upright, and an angular velocity, $\dot{\theta}$. The angular acceleration of the pendulum obeys the dynamics equation

$$\ddot{\theta} = \sin\theta + u ,$$

where the sine term is due to gravity and u is a control torque. This equation assumes the pendulum is of length one, mass one, and gravity is of unit strength. Since the pendulum's dynamics are identical at the states $(\theta + 2k\pi, \dot{\theta})$, for $k \in \{0, \pm 1, \pm 2, \ldots\}$, we assume that θ stays in the range $[-\pi, \pi]$. In our simulations, if θ leaves that range, our code returns it to that range by adding or subtracting 2π, as appropriate. We also artificially restricted $\dot{\theta}$ to the range $[-6, 6]$. This was important only for the worst-behaved, naive reinforcement learning agents, which sometimes developed control policies that tended to drive the pendulum's velocity to $+\infty$ or $-\infty$.

The particular problem we study is mimimum-time control to a goal set comprising near-upright, near-stationary states: $G_1 = \{(\theta, \dot{\theta}) : \|(\theta, \dot{\theta})\|_2 \leq 0.01\}$. We assume that the control torque is bounded in absolute value by $u_{max} = 0.225$. With this torque limit, the pendulum cannot be directly driven to G_1. Instead, the solution involves swinging the pendulum back and forth repeatedly, pumping energy into it until there is enough energy to swing upright.

Most reinforcement learning algorithms are designed for discrete-time control. To bridge the gap between the learning algorithms and the continuous-time dynamics of the pendulum, we define a number of continuous-time control laws for the pendulum. Each control law chooses a control torque based on the state of the pendulum. The discrete-time actions of the agents in our experiments correspond to committing control of the pendulum to a specific control law for a specific period of time. Discrete-time control of continuous-time systems is often done in this manner (see Huber and Grupen, 1998b; Branicky et al., 2000, for other examples). In all, we introduce five control laws for the pendulum. The first two are simple constant-torque control laws.

$$P_1(\theta, \dot{\theta}) = +u_{max} , \qquad\qquad P_2(\theta, \dot{\theta}) = -u_{max} .$$

A third control law we use is a saturating linear-quadratic regulator design for the linear approximation of the pendulum dynamics centered at the origin. (See Vincent and Grantham, 1997, for details.) For a region of state space surrounding the origin, this controller is sufficient to pull the pendulum into the origin and hold it there.

$$P_3(\theta,\dot{\theta}) = \max(\min(-2.4142\,\theta - 2.1974\,\dot{\theta}, u_{max}), -u_{max})\ .$$

In other words, P_3 is the function $-2.4142\,\theta - 2.1972\,\dot{\theta}$ clipped so as to remain in the range $[-u_{max}, +u_{max}]$. The constants 2.4142 and 2.1972 come from numerical solution of a matrix equation that is part of the LQR design process; they have no special meaning by themselves.

None of these control laws rely on Lyapunov analysis, and none alone can bring the pendulum to an upright, balanced state from all initial states. We now develop two controllers based on a Lyapunov analysis of the pendulum.

The mechanical energy of the pendulum is $\mathrm{ME}(\theta,\dot{\theta}) = 1 + \cos(\theta) + \frac{1}{2}\dot{\theta}^2$. The first two terms correspond to gravitational potential energy, and the last term corresponds to kinetic energy. At the origin, the mechanical energy of the pendulum is precisely 2. For any other state with $\mathrm{ME} = 2$, if u is taken to be zero, the pendulum will naturally swing up and asymptotically approach the origin. So, the pendulum swing-up and balance problem can be reduced to the problem of achieving any state with a mechanical energy of 2.

The time derivative of the pendulum's mechanical energy is $\dot{\mathrm{ME}}(\theta,\dot{\theta}) = -\sin(\theta)\dot{\theta} + \dot{\theta}\ddot{\theta} = \dot{\theta}u$. So a natural control law for rapidly increasing the pendulum's energy is to choose u to be of magnitude u_{max} and with sign matching $\dot{\theta}$. It turns out that this control law has potential difficulties near positions where the control torque of $\pm u_{max}$ is at equilibrium with the effects of gravity (Perkins, 2002). However, a modification of this rule can avoid the equilibrium point problem. We call this strategy MEA for "modified energy ascent." Parameterized by w, this rule supplies a control torque of magnitude w in the direction of $\dot{\theta}$ "most of the time." When that choice is close to equilibrium with gravity, however, the control law switches down to a torque of magnitude $w/2$, which is not near equilibrium with gravity. The precise formulation is

$$\mathrm{MEA}(w,\theta,\dot{\theta}) = \begin{cases} \mathrm{sgn}(\dot{\theta})w & \text{if } |\dot{\theta}| > \dot{\varepsilon} \\ & \text{or } (0 < \dot{\theta} < \dot{\varepsilon} \text{ and } \theta \notin E_1) \\ & \text{or } (-\dot{\varepsilon} < \dot{\theta} < 0 \text{ and } \theta \notin E_2) \\ \frac{1}{2}\mathrm{sgn}(\dot{\theta})w & \text{if } (0 < \dot{\theta} < \dot{\varepsilon} \text{ and } \theta \in E_1) \\ & \text{or } (-\dot{\varepsilon} < \dot{\theta} < 0 \text{ and } \theta \in E_2) \\ \mathrm{sgn}(\theta)w & \text{if } \dot{\theta} = 0\ , \end{cases}$$

where $\mathrm{sgn}(x) = \{+1 \text{ if } x \geq 0 \text{ and } -1 \text{ if } x < 0\}$, $\dot{\varepsilon} > 0$ is a constant (we use $\dot{\varepsilon} = 0.1$ in our experiments), and E_1 and E_2 are sets of states surrounding the equilibrium points of $\pm w$ torque with gravity. In particular, letting $\theta_w = \sin^{-1}(w)$ and $\varepsilon = \frac{1}{2}(\sin^{-1}(w) - \sin^{-1}(\frac{1}{2}w))$, then $E_1 = [-\pi+\theta_w-\varepsilon, -\pi+\theta_w+\varepsilon) \cup [-\theta_w-\varepsilon, -\theta_w+\varepsilon)$ and $E_2 = (\theta_w-\varepsilon, \theta_w+\varepsilon] \cup (\pi-\theta_w-\varepsilon, \pi-\theta_w+\varepsilon]$. We use MEA to define two final control laws.

$$P_4(\theta,\dot{\theta}) = \mathrm{MEA}(u_{max},\theta,\dot{\theta})\ ,$$

$$P_5(\theta,\dot{\theta}) = \mathrm{MEA}(\tfrac{1}{2}u_{max},\theta,\dot{\theta})\ .$$

Experiment	Control laws	Goal set	Summary
1	P_4, P_5	G_2	Descent on L certain.
2	P_1, P_2, P_4, P_5	G_2	Descent on L possible, not guaranteed.
3	P_1, P_2	G_2	No guarantees.
4	P_1, P_2, P_3	G_1	No guarantees.

Figure 2: Control laws and goal sets for Experiments 1–4.

Theorem 3 *If the pendulum is controlled continuously by P_4 or P_5 from any initial state, then the mechanical energy of the pendulum increases monotonically and without bound. Further, from any initial state, if the pendulum is controlled by either P_4 or P_5 for a period of time δ, then the mechanical energy of the pendulum increases by at least some amount Δ. The amount Δ differs for P_4 and P_5 and depends on δ, but it does not depend on the initial state of the pendulum.*

Proof sketch: A full proof was presented by Perkins (2002). We sketch the main idea here, focusing on the second claim, from which the first claim follows easily. Recall that $\dot{\text{ME}} = \dot{\theta}u$. Because P_4 and P_5 apply at least $\frac{1}{4}u_{max}$ torque in the direction of $\dot{\theta}$, we know that the change in ME over a period of δ time is at least $\frac{1}{4}u_{max}\int_{t=0}^{\delta}|\dot{\theta}(t)|dt$, where $\dot{\theta}(t)$ denotes the angular velocity of the pendulum at time t. Since $\dot{\theta}(t)$ can be zero, it is not immediately obvious that the integral should be bounded above zero. Conceivably, $\dot{\theta}(t)$ could stay arbitrarily close to zero, and the integral could be arbitrarily small. Besides keeping $\dot{\text{ME}}$ nonnegative, a key property of MEA is that it keeps $\ddot{\theta}$ bounded away from zero, regardless of the state of the pendulum. Thus, MEA ensures that if $\dot{\theta}$ is near zero at some point, it will quickly move away from zero; either the pendulum accelerates rapidly, or it slows rapidly to a stop and accelerates the other way. Thus, $\dot{\theta}$ cannot stay near zero for the whole δ time period, and the integral can be bounded above zero across all possible initial states. ■

The pendulum can be brought to the upright balanced position by using either P_4 or P_5 to increase the pendulum's energy to precisely 2, and then letting $u = 0$ so that the pendulum swings up and asymptotically approaches the origin. Is this an optimal strategy? It is not. As we see in the next section, a reinforcement learning controller that learns to switch between P_4 and P_5 can produce significantly faster swing-up. Allowing switching between other controllers enables even faster swing-up. That P_4 is significantly suboptimal may be surprising. P_4 applies magnitude u_{max} torque most of the time, and thus more or less maximizes the instantaneous rate at which energy is pumped into the system. Reasons for its suboptimality, and an explanation of how the learning agents do better, are provided in Section 4.3.

4.2 Experiments

We performed four learning experiments in the deterministic pendulum domain. In each experiment, the agent had different actions to choose from—that is, different sets of control laws with which it could control the pendulum—as summarized in Figure 2. Choosing an action meant that the pendulum was controlled according to the corresponding control law for a period of one second, or until the pendulum entered the goal set.

In all experiments, the task was to get the pendulum to the goal set, G_1, in minimum time. However, in the first three experiments, we used the fact that from any state with mechanical energy

2, fixing $u = 0$ results in a trajectory which asymptotically approaches the origin and thus enters G_1. For these experiments, we defined a surrogate goal set, $G_2 = \{(\theta, \dot{\theta}) : \mathrm{ME}(\theta, \dot{\theta}) = 2\}$. The cost of a non-terminal action (one that did not bring the pendulum into G_2) was one. The cost of a terminal action was equal to the time from the start of the action until the pendulum reached G_2, plus the time it took the pendulum to swing up to G_1 from there. With this cost function, an optimal policy reflects minimum time control to the set G_1, even though we used the goal set G_2. For Experiment 4, action costs were simply equal to the duration of the action—one for non-terminal actions, and one or less for actions which caused the pendulum to enter G_1.

We used the Sarsa(λ) algorithm with $\lambda = 0.8$ to learn control policies. The action value functions were represented using separate CMAC function approximators for each action. Each CMAC covered the range of states $-\pi \leq \theta \leq \pi$ and $-6 \leq \dot{\theta} \leq 6$. Each CMAC had 10 layers, and each layer divided each dimension into 24 bins, for a total of 576 tiles per layer. Offsets were random. The learning rate for the k^{th} update of a tile's weight was $1/\sqrt{k}$. (See Sutton and Barto, 1998, for details and references on Sarsa and CMACs.) We performed 25 independent runs of 20,000 learning trials. Agents chose actions according to the ε-greedy strategy, with $\varepsilon = 0.1$. That is, on each time step, with probability 0.1 an agent took an action selected uniformly randomly. Otherwise, the agent chose an action with best estimated action value. Ties were broken randomly. After each learning trial we performed a test trial, in which there was no learning and no exploration, in order to evaluate the current policy. All trials started from state $(\theta, \dot{\theta}) = (\pi, 0)$. Trials were terminated if they exceeded 900 time steps, which is approximately 50 times longer than the time required to reach G_1.

The pendulum dynamics were simulated using 4^{th}-order Runge-Kutta integration with a fixed time step of 0.01 seconds. Because of the fixed time-step size, exactly hitting the G_2 goal set in simulation is very unlikely; slightly overshooting $\mathrm{ME} = 2$ is the norm. To make our simulation behave more like our theoretical model, we set the pendulum torque to $u = -100\,\mathrm{sgn}(\dot{\theta})(\mathrm{ME}(\theta, \dot{\theta}) - 2)$ instead of $u = 0$ once $\mathrm{ME}(\theta, \dot{\theta})$ reached or exceeded 2. This shed any slight excess energy the pendulum had, usually on the order of 0.0001, and ensured that the pendulum reached G_1 correctly.

In the first experiment, the agent had two actions to choose from, corresponding to control laws P_4 and P_5. By Theorem 3, on any time step, both of the actions for the agent in Experiment 1 either cause the pendulum to enter G_2 or increase the mechanical energy by at least some amount $\Delta > 0$. Using this fact, we were able to establish a number of qualitative performance guarantees for the agent in Experiment 1. Defining $L(\theta, \dot{\theta}) = 2 - \mathrm{ME}(\theta, \dot{\theta})$ and letting $T = G_2$, we see that the conditions of Theorem 1 are satisfied. This means that the agent was guaranteed to bring the pendulum to G_2, and by extension, to G_1 on every trial (Property 2 with $T = G_1$). If we imagine the pendulum continuing to evolve under $u = 0$ after reaching G_1, we know that it would stay in G_1 forever and asymptotically approach the origin (Property 3 with $T = G_1$ and Property 5 with $T = \{(0,0)\}$). Further, the pendulum was guaranteed to remain in the set $T = \{(\theta, \dot{\theta}) : \mathrm{ME}(\theta, \dot{\theta}) \leq 2\}$ at all times, which can be viewed as a safety constraint. The agent could not pump an arbitrarily large amount of energy into the pendulum, which in reality might result in damage to the equipment or dangerously fast motions. In short, we were able to establish many reassuring performance guarantees for the agent by virtue of the actions we designed for it.

In Experiment 2, the agent chose from four actions, corresponding to control laws P_1, P_2, P_4, and P_5. Using P_4 or P_5, the agent was able to continuously increase the pendulum's energy, until it reached G_2. However, the agent did not have to do so. For example, it could repeatedly select action P_1, corresponding to $+u_{max}$ torque. This puts the pendulum on a cyclic trajectory that does not intersect G_2. Thus, during test trials, there was no guarantee that the agent would bring the pendulum to the goal. During learning trials, reaching the goal was guaranteed. Recall that during learning trials agents chose a random action on each time step with probability 0.1. This was done to ensure that the agents tried out and learned about all actions. For the agent of Experiment 2, it also ensured some probability of choosing P_4 or P_5 on each time step. Since $L(\theta,\dot{\theta}) = 2 - \mathrm{ME}(\theta,\dot{\theta})$ is bounded above on the set $\{(\theta,\dot{\theta}) : \mathrm{ME}(\theta,\dot{\theta}) < 2\}$, Theorem 2 applies with $q = 0.1$ and $p = 1$. The agent in Experiment 2, then, was guaranteed to bring the pendulum to G_1 on every learning trial with probability 1, as long as the trials lasted long enough. (Because trials were limited to 900 seconds, it was possible in principle that the pendulum would not reach the goal in some learning trials. In the next section, however, we observe that no learning trials in Experiment 2 were even one-tenth that long.) The other guarantees established for Experiment 1, that energy stays bounded, and that the pendulum converges to the origin after reaching G_2, apply in Experiment 2 as well.

In Experiment 3, there were just two actions, corresponding to control laws P_1 and P_2. By virtue of using the G_2 goal set, the pendulum was guaranteed to remain in the set $T = \{(\theta,\dot{\theta}) : \mathrm{ME}(\theta,\dot{\theta}) \leq 2\}$ (Property 1). And the pendulum was guaranteed to approach the origin after reaching G_2. But there was no guarantee, during learning or test trials, that the pendulum would reach G_2, regardless of the length of those trials.

In Experiment 4, the agent had three actions to choose from, corresponding to control laws P_1, P_2, and P_3. In this case, we used G_1 as the goal set. Initially, we experimented with an agent that chose between just the P_1 and P_2 control laws. However, it turned out to be extremely difficult to bring the pendulum to the small G_1 goal set by switching between $+u_{max}$ and $-u_{max}$ torque in discrete time. We added the P_3 control law in Experiment 4 to make the problem tractable.

We have described the experiments in this order because it represents decreasing dependence on Lyapunov domain knowledge. In Experiment 1, the agent was constrained to make the pendulum ascend on mechanical energy and reach the G_2 goal set. In Experiment 2, the agent had actions based on the Lyapunov-designed controllers P_4 and P_5, and used goal set G_2. But the agent was not constrained to increase the pendulum's mechanical energy at all times. In Experiment 3, we dropped the actions based on the Lyapunov-designed control laws, and in Experiment 4 we dropped the G_2 goal set. Experiment 4 might be considered the formulation of the control problem that uses the least prior knowledge while still allowing a solution to be learned with reasonable effort.

4.3 Results and Discussion

Figure 3 displays learning curves for the agents in Experiments 1 through 4, plotted on the same scale to facilitate comparison. The dark lines represent the time to G_1 during test trials, averaged across the 25 independent runs. The lighter lines represent the shortest and longest test trials across runs. For readability purposes, the curves are smoothed by averaging blocks of 5 trials together; the mean-time-to-goal curve omits trials that time out without reaching G_1. The horizontal axis shows the trial number, going up to 1,000 of the 20,000 total trials. By way of comparison, the strategy of using P_4 to pump energy into the pendulum until it reaches G_2 and then letting it swing to G_1 takes 24.83 seconds.

Most striking is the high-quality, low-variability initial performance of the "safe" learning agent in Experiment 1. This contrasts sharply with the results of Experiment 4, which show a more typical profile for a reinforcement learning system solving a dynamical control task; initial performance is poor and highly variable. In Experiments 2 and 3, we see intermediate behavior. Because the trials

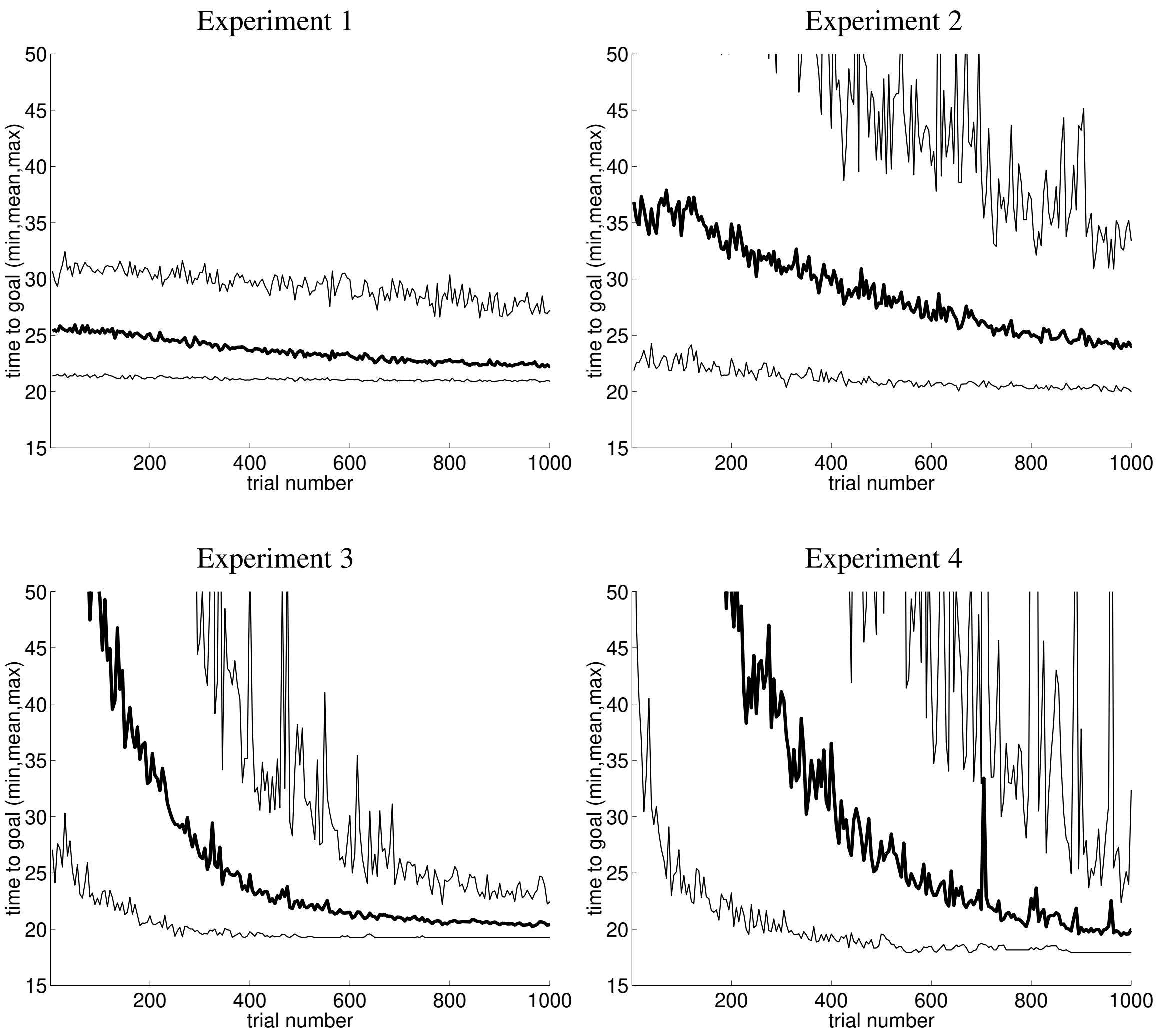


Figure 3: Mean, min, and max time to G_1 across runs during test trials.

		Experiment 1	Experiment 2	Experiment 3	Experiment 4
% of first 10	Learn	100	100	100	88.4
reaching G_1	Test	100	96.0	82.4	50.4
Time to G_1	Learn	$25.4 \pm .2$	$35.8 \pm .7$	62.1 ± 2.2	317 ± 38
first 10	Test	$25.4 \pm .3$	70.7 ± 22	222 ± 33	584 ± 39
Time to G_1	Learn	$21.2 \pm .05$	$21.0 \pm .1$	$21.4 \pm .1$	$22.3 \pm .2$
last 1000	Test	$21.0 \pm .06$	$19.8 \pm .1$	$19.5 \pm .1$	$18.7 \pm .2$
Longest	Learn	34.6	80.9	171	timeout
trial	Test	35.2	timeout	timeout	timeout

Figure 4: Summary statistics for Experiments 1–4.

are so short in Experiment 1, that agent also learns significantly faster (in terms of total time steps) than the agents in the other experiments. Figure 4 gives more detailed statistics summarizing the experiments. The table reports the percentage of the first ten trials which make it to G_1 before timing out at 900 time steps, the mean duration of the first ten learning and test trials, the mean duration of the last 1000 learning and test trials, and the longest trials observed in each experiment. The means are accompanied by 95% confidence intervals.

The initial performance of the agent in Experiment 1 was clearly the best, an order of magnitude better than that of the agent in Experiment 4, which incorporated the least prior knowledge. Importantly, only in Experiment 1, for which the strongest theoretical guarantees were established, did the agent bring the pendulum to the goal on every learning and test trial. The longest trial generated in that experiment was 35.2 seconds, whereas all the other experiments included much worse learning trials and test trials that timed out. In this domain, ensuring descent on the Lyapunov function, and not the mere presence of Lyapunov-based actions, is necessary for ensuring good performance at all times.

The performance figures for the final 1000 trials reveal significant improvements from initial performance in all experiments. In the end, all the learning agents also outperform the simple strategy of using P_4. However, the numbers reveal a down-side to using Lyapunov domain knowledge. The constraint to descend on the Lyapunov function limits the ability of the agent in Experiment 1 to minimize the time to G_1. In the final test trials, it takes an average of 21.0 seconds to swing the pendulum upright. The agents in Experiments 2 and 3 do better because they are not constrained in this way. Their final test trials take 19.8 and 19.5 seconds on average. Why is this? Recall that the time derivative of the pendulum's mechanical energy is $\dot{ME}(\theta,\dot\theta) = \dot\theta u$. In Experiment 1, the agent always chose a u that matched $\dot\theta$ in sign, thus constantly increasing mechanical energy. Note, however, that when $\dot\theta$ is close to zero, mechanical energy necessarily increases only slowly. Every time the pendulum changes direction, $\dot\theta$ must pass through zero, and at such times the agent in Experiment 1 did not make much progress towards reaching G_2. It turns out that a better policy in the long run is to accelerate against $\dot\theta$ and turn the pendulum around rapidly. This way, the pendulum spends more time in states where $\dot\theta$ is far from zero, meaning that energy can be added to the system more quickly. The agent in Experiment 1 was able to achieve this to some degree, by switching from P_4 to the lower-torque P_5, allowing more rapid turn-around. But this did not match the freedom afforded in Experiments 2 and 3. The agent in Experiment 4 had yet another advantage. It was not constrained to choose $u = 0$ once the pendulum had reached G_2 and before it had reached G_1. The agent was able to continue accelerating the pendulum towards upright and then decelerate it as it approached, resulting in even faster trajectories to G_1.

5. Stochastic Pendulum Swing-Up and Balance

In this section, we consider a swing-up and balance problem for a pendulum with stochastic dynamics. We formulate the problem as a continuing task, not a goal-based task. The agent seeks to swing the pendulum upright and maintain it upright for as much of the time as possible. The pendulum's position variable is subject to random disturbances which tend to push the pendulum away from the balanced state, making the task more challenging.

5.1 Dynamics and Controllers

We model the pendulum's dynamics by the system of stochastic control differential equations:

$$d\theta = \dot{\theta}dt + 0.1dW \ ,$$

$$d\dot{\theta} = sin(\theta)dt + udt \ ,$$

where W denotes a Wiener process with mean zero and standard deviation one (see, for example, Kushner, 1971). The Wiener process is the standard model of a continuous-time Gaussian disturbance, or "white noise." The equations above describe a Gaussian disturbance to the position variable, with mean zero and standard deviation 0.1.

As mentioned above, there are no goal states for this problem. The agent's task is to keep the pendulum near upright as much as possible, where "near upright" is defined by the set $S_{\text{up}} = \{(\theta, \dot{\theta}) : |\theta| \leq 0.5\}$. The agent incurs unit cost per time as long as the pendulum is not in S_{up}, and incurs no cost while the pendulum is in S_{up}. S_{up} contains all states up to approximately 30 degrees away from upright.

Because of the random disturbances to the position variable, no controller can maintain the pendulum in the set S_{up} indefinitely. Indeed, because the disturbance to θ is Gaussian, with infinite tails, there is some probability that the pendulum will "jump" out of S_{up} on any given time step, regardless of how it is controlled. On the other hand, the same disturbances insure that, regardless of how the pendulum is controlled, with probability one it will eventually enter S_{up} from any initial state. Thus, under any controller the pendulum reaches S_{up} with probability one (Property 2) and spends an infinite amount of total time there (Property 4). In Section 5.3, however, we will see that these guarantees are very weak, and that controllers that are explicitly designed to return the pendulum to S_{up} can have a much stronger influence.

We define four new control laws for this problem. One is just the zero torque control law.

$$P_6(\theta, \dot{\theta}) = 0 \ .$$

The other three control laws, P_7, P_8, and P_9, are designed to return the pendulum to S_{up}. They behave identically when the pendulum is outside S_{up}, and differ only in that they apply $-u_{max}$, 0, or $+u_{max}$ torque when the pendulum is in S_{up}. To bring the pendulum to S_{up}, the control laws use a strategy that depends in part on the MEA control law defined above. In particular, when the pendulum's energy is below 2, each control law matches P_4. This pumps energy into the pendulum, which ultimately sends it toward S_{up}. When the energy is above 2, the pendulum is at risk of

Experiment	Control laws	Summary
5	P_7, P_8, P_9	Designed to return to S_{up}; choice of $\pm u_{max}$ and 0 in S_{up}.
6	P_1, P_2, P_6	Choice of $\pm u_{max}$ and 0 torques.

Figure 5: Design of Experiments 5 and 6.

swinging upright, passing through S_{up} quickly, and swinging down again on the other side. In this case, the control laws apply a braking torque to the pendulum, with strength proportional to the amount of excess energy the pendulum has. Intuitively, when the pendulum is outside S_{up}, these control laws "aim for" the upright, stationary position in the center of S_{up}. Letting $B(\theta,\dot{\theta}) = \max(\min(-100\text{sgn}(\dot{\theta})(\text{ME}(\theta,\dot{\theta})-2), u_{max}), -u_{max})$ be the braking force applied to a pendulum with energy in excess of 2, we define

$$P_7(\theta,\dot{\theta}) = \begin{cases} +u_{max} & \text{if } (\theta,\dot{\theta}) \in S_{\text{up}} \\ P_4(\theta,\dot{\theta}) & \text{if } (\theta,\dot{\theta}) \notin S_{\text{up}} \text{ and } \text{ME}(\theta,\dot{\theta}) < 2 \\ B(\theta,\dot{\theta}) & \text{if } (\theta,\dot{\theta}) \notin S_{\text{up}} \text{ and } \text{ME}(\theta,\dot{\theta}) \geq 2\ , \end{cases}$$

$$P_8(\theta,\dot{\theta}) = \begin{cases} 0 & \text{if } (\theta,\dot{\theta}) \in S_{\text{up}} \\ P_4(\theta,\dot{\theta}) & \text{if } (\theta,\dot{\theta}) \notin S_{\text{up}} \text{ and } \text{ME}(\theta,\dot{\theta}) < 2 \\ B(\theta,\dot{\theta}) & \text{if } (\theta,\dot{\theta}) \notin S_{\text{up}} \text{ and } \text{ME}(\theta,\dot{\theta}) \geq 2\ , \end{cases}$$

$$P_9(\theta,\dot{\theta}) = \begin{cases} -u_{max} & \text{if } (\theta,\dot{\theta}) \in S_{\text{up}} \\ P_4(\theta,\dot{\theta}) & \text{if } (\theta,\dot{\theta}) \notin S_{\text{up}} \text{ and } \text{ME}(\theta,\dot{\theta}) < 2 \\ B(\theta,\dot{\theta}) & \text{if } (\theta,\dot{\theta}) \notin S_{\text{up}} \text{ and } \text{ME}(\theta,\dot{\theta}) \geq 2\ . \end{cases}$$

5.2 Experiments

Figure 5 summarizes the two experiments we performed in the stochastic pendulum domain. In Experiment 5, the agent chose from actions corresponding to P_7, P_8, and P_9. This agent did not need to learn how to get the pendulum into S_{up}. Outside of S_{up}, all three of its control laws are identical and are designed to do that task. All that the agent had to learn was how best to keep the pendulum in S_{up} once it was there, by switching among $-u_{max}$, 0, and $+u_{max}$ torques. In Experiment 6, the agent chose from control laws P_1, P_2, and P_6, corresponding to torques $-u_{max}$, 0, and $+u_{max}$. This agent had to learn how to get the pendulum into S_{up}, as well as how to keep it there. Neither agent could keep the pendulum in S_{up} indefinitely and, for either agent, the pendulum was guaranteed to return to S_{up} regardless of the choices the agent made. However, the Lyapunov-based actions in Experiment 5 strongly biased that agent toward bringing the pendulum to S_{up}. At the least, superior initial performance was expected in Experiment 5.

We performed 25 independent learning runs with 1,000 learning and 1,000 test trials in each run. Since there are no goal states to determine an end to a trial, each trial was run for 999 simulated seconds. We instituted a discount rate of $\gamma = 0.95$ so that expected discounted returns would be finite and the agents' action value estimates would not diverge to $+\infty$. Action selection was ε-greedy with $\varepsilon = 0.1$. We used the same learning algorithm, Sarsa(λ) with $\lambda = 0.8$, and CMACs as in Experiments 1 through 4. In pilot experiments, we found that for an agent to perform well on this task, it had to be allowed to choose new actions more frequently than once per second. We present results of experiments in which the agents were allowed to choose a new action every 0.3 seconds.

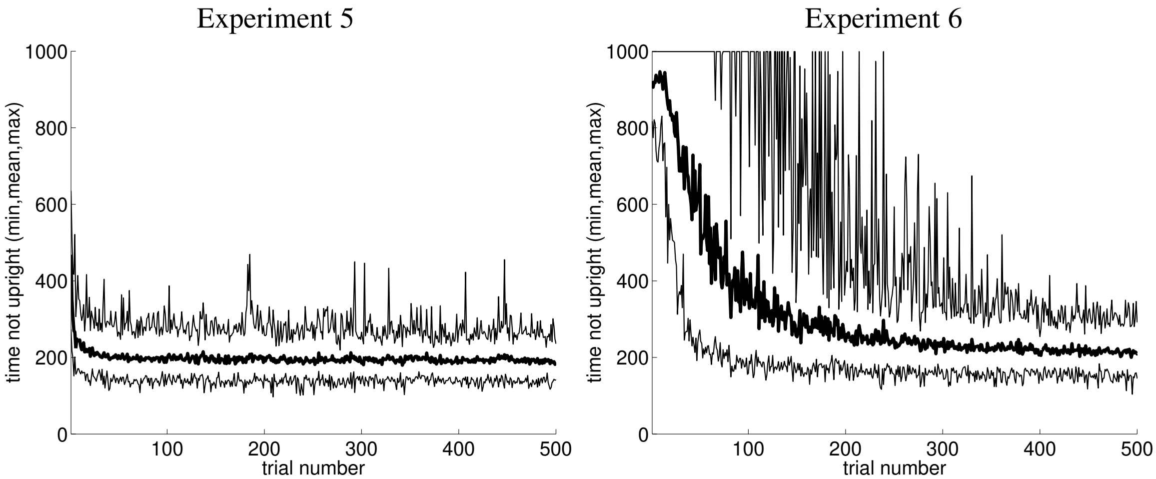


Figure 6: Mean, min, and max across runs of time spent not near upright during test trials.

		Experiment 5	Experiment 6
Cost of first trial	Learn	523 ± 11	848 ± 9
	Test	468 ± 44	922 ± 28
Mean cost in first 10 trials	Learn	366 ± 8	841 ± 4
	Test	287 ± 10	925 ± 8
Mean cost in last 100 trials	Learn	279 ± 5	301 ± 9
	Test	193 ± 7	203 ± 11
Highest cost trials	Learn	555	964
	Test	635	999

Figure 7: Summary statistics for Experiments 5 and 6.

The pendulum dynamics were simulated using a time step size of 0.1 second. On each time step, 4^{th}-order Runge-Kutta integration was used to compute the deterministic dynamics, after which a mean zero, standard deviation $0.1\sqrt{0.1}$ Gaussian disturbance was added to the position variable.

5.3 Results and Discussion

Figure 6 displays learning curves representing the minimum, mean, and maximum test trial costs across the 25 runs. Note that trial cost is the same as time the pendulum spends outside of S_{up}. These curves are not smoothed, and are plotted for the first 500 of the 1,000 total test trials. By way of comparison, the single best-performing control law for the task is P_8, which keeps the pendulum in S_{up} approximately 43% of the time, corresponding to an average trial cost of around 570.

The agent in Experiment 5 had better initial and final performance than the agent in Experiment 6, with its naive, constant-torque action formulation. There is less variance in its performance, and it completes most of its learning in the first few tens of trials. By contrast, the agent in Experiment 6 was still learning its policy after 300 trials. Even after 200 learning trials, in some test trials the pendulum spend no time in S_{up}.

Figure 7 presents detailed statistics supporting these observations. The difference in performance between the first trial and first ten trials of Experiment 5 indicates how much learning happened early on. Of course, trials were 999 seconds long, almost 4,000 decision steps, so it is not surprising that significant learning could take place in the first few trials. The performance figures for the final 100 test trials indicate that the agent in Experiment 5 did better than the agent in Experiment 6. However, the difference is on the edge of statistical significance. Although the shapes of the learning curves suggest that the agents had reached an asymptotic level of performance, it is possible that with further learning or tweaking of learning parameters, greater performance could be obtained. At any rate, the restriction in Experiment 5 to push the pendulum's energy towards 2 whenever it is not in S_{up} appears not to have negatively impacted the agent's performance. The figures for the highest-cost trials demonstrate once again the benefits of Lyapunov-based designs in ensuring good performance. Even in the worst test trial in Experiment 5, the pendulum was in S_{up} for more than one third of the 999 seconds. In the worst learning trial, the pendulum was near upright for almost half of the time. Experiment 6 included many test trials in which the pendulum spent no time in S_{up} and some learning trials that were almost as bad.

6. Robot Arm Movement and Stabilization

Robot arm, or manipulator, control is possible through the well-known technique of "feedback linearization," described below. Feedback linearization is an important tool in control theory and widely used in control practice as well. For robot arm control and other problems, however, control based on feedback linearization is notorious for being inefficient, slow, and "robotic", and/or for exerting unnecessarily high torques. Feedback linearization does not take advantage of the natural dynamics of the arm. Indeed, the main idea is to use control torque to eliminate most of the natural dynamics, making analytical solution of the problem possible. From an optimal control point of view, then, robot arm control remains an interesting and unsolved problem.

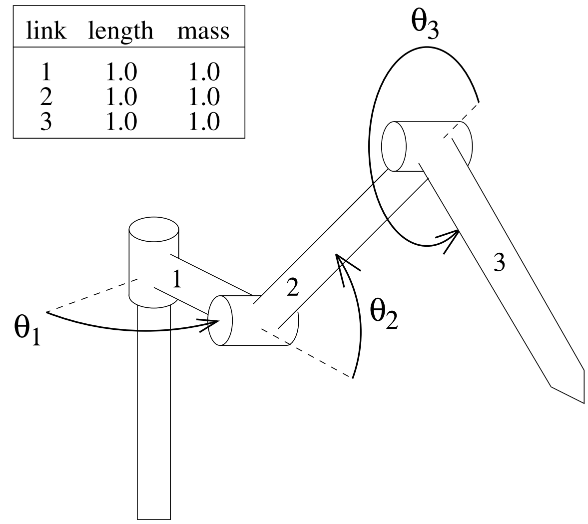
The last two problems we consider are simulated robot arm control problems. We treat the two largely simultaneously, as the only difference between them is that in one case the arm's dynamics are deterministic and in the other case the dynamics are stochastic. The same goal set, cost function, and sets of control laws are used for both versions of the dynamics.

6.1 Dynamics and Controllers

The robot arm is depicted in Figure 8. Its state is specified by three angular joint positions and three angular velocities. We use x to denote the state vector. The vector of joint positions and the vector of joint velocities are denoted by Θ and $\dot{\Theta}$ respectively. For the deterministic version of the dynamics, we use a standard frictionless model (Craig, 1989):

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

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. For the stochastic version of the dynamics, we perturb the joint positions by noise that is



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

Figure 8: Diagram of the 3-link arm.

multiplicative in the joint velocities.[2]

$$d\begin{bmatrix}\Theta\\ \dot{\Theta}\end{bmatrix}=\begin{bmatrix}\dot{\Theta}(dt+0.2dW)\\ H^{-1}(\Theta,\dot{\Theta})(\tau-V(\Theta,\dot{\Theta})-\mathcal{G}(\Theta))dt\end{bmatrix}.$$

Here, W should be interpreted as a vector of three independent Wiener processes with mean zero and standard deviation one. We assume joint positions are limited to the range $[-\pi,+\pi]$. If the dynamics try to push a joint's position out of this range, its position is held at the edge of the range (that is, at $+\pi$ or $-\pi$) and the joint's velocity is set to zero. We do not explicitly bound joint velocities.

The task is to move the arm to the goal set $G_{arm}=\{x:\|x\|\leq 0.01\}$. This is a set of low-velocity states near to a fully-extended horizontal configuration, $\Theta=0$. Rather than having a single initial state, we use an initial state distribution that is uniform over the configurations $\Theta_0=(a,b,-b)$ for $a\in\{-\pi,-\frac{2}{3}\pi,-\frac{1}{3}\pi\}$ and $b\in\{-\frac{1}{2}\pi,0,+\frac{1}{2}\pi\}$, with zero velocity.

As in Experiments 1 through 4, agents' actions correspond to choosing control laws to apply to the arm for a period of one second. Letting $\tau(t)$, $t\in[0,1]$, denote the torques chosen by a control law during one second of control, and letting $\Theta(t)$ denote the joint positions the arm goes through during that time, we define the cost of choosing that control law to be $\int_0^1\|\Theta(t)\|^2+\|\tau(t)-\tau_0\|^2dt$, where τ_0 is the amount of torque needed to hold the arm steady at the origin. Intuitively, this cost function penalizes the agent to the extent that the state is far from G_{arm} and to the extent that large control torques, beyond the unavoidable τ_0, are applied to the arm.

In pilot experiments, we found that it was virtually impossible for an agent to bring the arm to G_{arm} using naive action formulations. With three joints to actuate and a range of torques that need to be applied to reach G_{arm}, a sufficient set of constant-torque actions would be unmanageably large. We also found that a control law based on a linear approximation to the arm dynamics at the origin brought the arm into G_{arm} for only a fairly small set of surrounding states. It is possible that learning algorithms explicitly designed to handle continuous action spaces could be successful on

2. There is no special reason for choosing multiplicative instead of additive noise, except to demonstrate another common noise model.

the problem (for example, Gullapalli, 1992). However, we used the same approach that we found successful on the pendulum—defining a small set of well-chosen control laws with which an agent can learn to control the arm.

The standard method for controlling a robot arm is to combine feedback linearization with some form of linear system control (Craig, 1989). A simple approach to feedback linearization in the robot arm 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 simple linear form

$$\frac{d}{dt}\begin{bmatrix}\Theta \\ \dot{\Theta}\end{bmatrix} = \begin{bmatrix}\dot{\Theta} \\ u\end{bmatrix},$$

for the deterministic case, and

$$d\begin{bmatrix}\Theta \\ \dot{\Theta}\end{bmatrix} = \begin{bmatrix}\dot{\Theta}(dt + 0.2dW) \\ udt\end{bmatrix},$$

for the stochastic case. Since H is always non-singular, this transformation is lossless; any τ can be achieved by the proper choice of u. Once the dynamics are expressed in linear form, linear control design methods apply. We use a linear-quadratic regulation (LQR) approach. This results in controllers that are able to move the arm to any desired configuration, and also provides the Lyapunov function which constrains one of the action sets we define below. We briefly describe the LQR approach. (See, for example, Vincent and Grantham, 1997, for a more thorough discussion.)

Consider a linear system of the form $\dot{x} = Ax + Bu$, where x is a state vector, u is a control vector, and A and B are constant matrices. Suppose we want to move this system to a target state x_T from any initial state x_0, and suppose we specify a control law for performing this task. Let $x(t)$ be a system trajectory that results from that control law. The "quadratic" part of LQR says that the quality of that control law should be measured by the cost function $\int_0^\infty [(x(t) - x_T)'Q(x(t) - x_T) + u(x(t))'Ru(x(t))]dt$, where Q and R are symmetric positive-definite matrices of our choosing. That is, a quadratic penalty is associated with the deviation of the state, $x(t)$, from the target state, x_T, and a quadratic penalty is associated with the control effort u. If we restrict ourselves to control laws of the form $u = -K(x - x_T)$, where K is a constant "gain" matrix, then the optimal K can be found by solving a matrix algebraic Ricatti equation. This equation is typically solved numerically, though in the present case it can readily be solved by hand (Perkins, 2002). By varying our choices of Q and R, we can produce K's that represent different tradeoffs between the speed of approaching x_T and the size of the control effort, u, that is applied.

Note that the cost function of our optimal control problem and the cost function of the LQR design are similar in form. Both include a quadratic penalty on the deviation of the state from the target and a quadratic penalty on control effort. A key difference is that the LQR design penalizes u, which is just a part of the total torque, $\tau = Hu + V + \mathcal{G}$, applied by a feedback linearization-LQR design. This is one reason that robot arm control based on feedback linearization tends to be unsatisfactory from an optimal control point of view. When one designs a controller for an arm in feedback-linearized form, one treats u as if it were the only control effort being applied. Really, the control effort that ought to be minimized is τ, or $\tau - \tau_0$, and the dynamics that should be exploited, given by H, V, and $\mathcal{G}$, are all hidden by the feedback linearization transformation. All the control laws we design for the arm are based on the linear feedback transformation. By specifying a cost function that accounts for the true effort of controlling the arm, however, we hope to re-instill some sensitivity to the arm's natural dynamics. By trying to minimize this cost function, our reinforcement learning agents are able to discover policies for controlling the arm that use less real effort and/or less time than a standard LQR design.

We define five controllers based on the feedback linearization transformation and LQR design. That is, each controller chooses $\tau = Hu + V + \mathcal{G}$, where u is chosen by an LQR design. Let LQR(Q,R) stand for the gain matrix resulting from LQR design with penalty matrices Q and R; let Q_0 be the matrix with diagonal $\{1,1,1,0.01,0.01,0.01\}$; let R_0 be the 3×3 identity matrix; let K=LQR(Q_0,R_0); and let Θ_4, and Θ_5 respectively be the joint position vectors $[-\frac{1}{2}\pi, \frac{1}{2}\pi, -\pi]$, and $[-\frac{1}{2}\pi, -\frac{1}{2}\pi, \pi]$. Our first five control laws for the arm are

$$A_1(\Theta,\dot{\Theta}) = -H(\Theta,\dot{\Theta})K \begin{bmatrix} \Theta \\ \dot{\Theta} \end{bmatrix} V(\Theta,\dot{\Theta}) + \mathcal{G}(\Theta) \,,$$

$$A_2(\Theta,\dot{\Theta}) = -H(\Theta,\dot{\Theta})K_2 \begin{bmatrix} \Theta \\ \dot{\Theta} \end{bmatrix} V(\Theta,\dot{\Theta}) + \mathcal{G}(\Theta) \text{ where } K_2\text{=LQR}(4Q_0,R_0) \,,$$

$$A_3(\Theta,\dot{\Theta}) = -H(\Theta,\dot{\Theta})K_3 \begin{bmatrix} \Theta \\ \dot{\Theta} \end{bmatrix} V(\Theta,\dot{\Theta}) + \mathcal{G}(\Theta) \text{ where } K_3\text{=LQR}(Q_0,4R_0) \,,$$

$$A_4(\Theta,\dot{\Theta}) = -H(\Theta,\dot{\Theta})K \begin{bmatrix} \Theta - \Theta_4 \\ \dot{\Theta} \end{bmatrix} V(\Theta,\dot{\Theta}) + \mathcal{G}(\Theta) \,,$$

$$A_5(\Theta,\dot{\Theta}) = -H(\Theta,\dot{\Theta})K \begin{bmatrix} \Theta - \Theta_5 \\ \dot{\Theta} \end{bmatrix} V(\Theta,\dot{\Theta}) + \mathcal{G}(\Theta) \,.$$

The intuitions behind these control laws are as follows. The first three control laws move the arm to the origin, which is in the center of the goal set, G_{arm}. They move the arm there at different rates, however. Q_0 primarily penalizes the arm's position for being far from the zero position, putting relatively little weight on the arm's velocity being different from zero. This reflects the fact that we want the arm to be moved to the goal position, but we are not particularly concerned about the velocity of the arm as it goes to the goal. R_0 puts a unit penalty on control effort, u, so the A_1 design represents an equal balance between getting the arm to the zero position and exerting control effort. A_2 uses a higher penalty on the distance to the goal position; this control law tends to move the arm to the origin more quickly than A_1 by using a larger control effort, u. A_3 is biased in the opposite direction. The final two control laws, A_4 and A_5, do not move the arm into G_{arm} at all. They move it to the configurations Θ_4 and Θ_5 respectively, which are 90 degrees from G_{arm} with the outer two links of the arm folded in. This can be useful because when the arm is folded, its moment of inertia around the first joint is reduced, allowing it to be swung around that joint with little actual torque.

Under the deterministic dynamics, each of the control laws A_1, A_2, and A_3 individually cause the arm to asymptotically approach the origin, and thus enter G_{arm}, from any initial state. The standard proof that an LQR design causes the controlled system to asymptotically approach the target point uses a Lyapunov argument. Any LQR controller causes system trajectories that descend on a simple positive-definite quadratic function, $(x - x_T)'V(x - x_T)$. The time derivative of this function along system trajectories is $(x - x_T)'U(x - x_T)$, where U is a symmetric negative-definite matrix. The matrices V and U are readily constructed from the computations that are required to generate the gain matrix K (Vincent and Grantham, 1997). We take the matrix V corresponding to A_1 to define our Lyapunov function $L_{arm} = x'Vx$. Since the time derivative of L_{arm} is negative-definite under A_1, it is bounded below zero outside of G_{arm}. Thus from any initial state, during any one second period, A_1 causes the arm to either enter G_{arm} or descend on L_{arm} by an amount at least Δ.

Control laws A_2 through A_5 do not always cause descent on L_{arm}, but we can readily define similar controllers that do. Let C be any control law. We can define an L_{arm}-descending version as

$$\text{LD}(C,\Theta,\dot{\Theta}) = \begin{cases} C(\Theta,\dot{\Theta}) & \text{if } L_{arm}(\Theta,\dot{\Theta}) \geq 0.1 \text{ and } C(\Theta,\dot{\Theta}) \text{ causes } \frac{d}{dt}L_{arm}(\Theta,\dot{\Theta}) \leq -0.1 \\ A_1(\Theta,\dot{\Theta}) & \text{otherwise.} \end{cases}$$

We define four more control laws for the arm as the L_{arm}-descending versions of control laws A_2 through A_5.

$$A_6(\Theta,\dot{\Theta})=\text{LD}(A_2,\Theta,\dot{\Theta}) \, ,$$

$$A_7(\Theta,\dot{\Theta})=\text{LD}(A_3,\Theta,\dot{\Theta}) \, ,$$

$$A_8(\Theta,\dot{\Theta})=\text{LD}(A_4,\Theta,\dot{\Theta}) \, ,$$

$$A_9(\Theta,\dot{\Theta})=\text{LD}(A_5,\Theta,\dot{\Theta}) \, .$$

Theorem 4 *For the deterministic arm dynamics, there exists $\Delta > 0$ such that for any $s \notin G_{arm}$, if the arm is controlled for one second by A_1,A_6,A_7,A_8, or A_9, then either the arm enters G_{arm} or the resulting state is at least Δ lower on L_{arm}. For the stochastic arm dynamics, there exists $\Delta > 0$ and $p > 0$ such that for any $s \notin G_{arm}$, if the arm is controlled for one second by A_1,A_6,A_7,A_8, or A_9, then with probability at least p either the arm enters G_{arm} or the resulting state is at least Δ lower on L_{arm}. For either set of dynamics, in any trajectory created by switching among any of the controllers $A_1,\ldots,A_9$, the system state stays bounded at all times.*

Proof sketch: First, we argue boundedness. The Θ component of the state is bounded at all times in the region $[-\pi,\pi]^3$. The only concern, then, is with the velocity component of the state, $\dot{\Theta}$. All of the control laws above are LQR controllers or combinations of LQR controllers for the feedback-linearized arm. Each LQR controller has feedback gains on the joint velocities which tend to slow each joint down. That is, the acceleration of joint i follows the rule: $\ddot{\Theta}(i) = -a(\Theta(i) - \Theta_T(i)) - b\dot{\Theta}(i)$, where a and b are positive constants and $\Theta_T(i)$ is the i^{th} component of the target configuration for the LQR controller. Since Θ is bounded, for velocities larger in magnitude than some $\mathcal{V} \in \mathbb{R}$, this rule results in braking force which reduces the joint's velocity regardless of its position. The only way for a joint velocity to be greater than $\mathcal{V}$ is if it starts that way—that is, if $\|\dot{\Theta}_0\|_\infty > \mathcal{V}$. Thus, at all times in a trajectory, the joint velocities are bounded by $\max(\mathcal{V}, \|\dot{\Theta}_0\|_\infty)$.

Next, we establish descent under controllers A_1,A_6,A_7,A_8, and A_9. For the deterministic dynamics, A_1 keeps $\frac{d}{dt}L_{arm}$ bounded below zero off of G_{arm}. Specifically, we have $\frac{d}{dt}L_{arm} \leq \sup_{x \notin G_{arm}} x'Ux$, where U is the negative-definite matrix giving the time derivative of the Lyapunov function at state x. A_6,A_7,A_8, and A_9 also keep $\frac{d}{dt}L_{arm}$ bounded below zero, by construction. It is immediate, therefore, that for any of these control laws, one second of control either brings the arm into G_{arm} or results in a state lower on L_{arm} by at least some Δ. The same property holds for the stochastic dynamics, but with probability p. With zero-mean noise, one can argue that there is some probability p that one second of control results in a state within ε of the state that would result under the deterministic dynamics. Thus, there is at least probability p that the arm enters G_{arm} or results in a state lower on L_{arm} by $\Delta/2$ (for example), where Δ is the amount of descent guaranteed under the deterministic dynamics. ■

Experiment	Dynamics	Control laws	Summary
7	deterministic	A_1, A_6, A_7, A_8, A_9	Descent on L_{arm} guaranteed.
8	deterministic	A_1, A_2, A_3, A_4, A_5	Descent possible, not guaranteed.
9	stochastic	A_1, A_6, A_7, A_8, A_9	Probabilistic descent on L_{arm}.
10	stochastic	A_1, A_2, A_3, A_4, A_5	Descent possible, not guaranteed.

Figure 9: Design of Experiments 7 through 10.

6.2 Experiments

We performed four experiments in the robot arm domain, as outlined in Figure 9. Experiments 7 and 8 used the deterministic version of the dynamics, while Experiments 9 and 10 used the stochastic version. In Experiments 7 and 9, the agent chose from five actions, corresponding to the control laws A_1, A_6, A_7, A_8, and A_9. In Experiments 8 and 10, the agent chose from actions corresponding to the control laws A_1, A_2, A_3, A_4, and A_5.

For each experiment, we ran 25 independent runs of Sarsa(λ) with $\lambda = 0.8$. Each run consisted of 1,000 blocks, or suites, of 18 trials—one learning trial from each of the states in our initial state distribution, followed by one test trial from each initial state. Action selection was ε-greedy with $\varepsilon = 0.1$. Action value functions were represented using a separate CMAC function approximator for each action. Each CMAC covered the region of state space: $[-\pi, \pi]^3 \times [-5, 5]^3$. Each CMAC had 20 layers, dividing each dimension into 5 equal-sized bins, with offsets being random. The learning rate for the k^{th} update of a tile's weight was $1/\sqrt{k}$. Trials were terminated after 250 time steps if they did not reach G_{arm}.

The deterministic arm dynamics were simulated using 4^{th}-order Runge-Kutta integration with a fixed time step size of 0.1 seconds. The stochastic dynamics were simulated by computing the deterministic dynamics and then multiplying the change to each joint position by one plus a Gaussian random disturbance with mean zero and standard deviation $0.2\sqrt{0.1}$.

By Theorem 4, the formulation of Experiment 7 satisfies the conditions of Theorem 1 with $T = G_{arm}$. Thus, that agent was guaranteed to control the arm to G_{arm} on every trial (Property 2). If we assume that, once the arm enters G_{arm}, it continues to be controlled according to A_1, then the arm remains in G_{arm} and asymptotically approaches the origin (Properties 3 and 5). Since we know that all of the controllers available in Experiment 7 descend on L_{arm}, a trajectory that starts at state x_0 is guaranteed not to leave the region $\{x : L(x) \leq L(x_0)\}$ (Property 1), which can be viewed as a safety property. For Experiment 9, Theorem 4 implies that the conditions of Theorem 2 are satisfied, thus this agent had a probability one guarantee of controlling the arm to G_{arm} on every trial. Under the stochastic dynamics, no control law can guarantee keeping the arm in G_{arm}. However, one can show using arguments based on martingale theory that A_1, for example, has some chance of keeping the arm in G_{arm} forever. This is possible because the position disturbance, which is multiplicative in $\dot{\Theta}$, vanishes as $\dot{\Theta} \to 0$. We do not delve into these arguments here, but see, for example, the book by Kushner (1971).

The agents in Experiments 8 and 10 were not guaranteed to choose descending actions during testing trials, hence there was no guarantee that these agents would bring the arm to G_{arm}. During learning trials, however, the ε-greedy action selection strategy meant that there was at least probability 0.1 on every time step that these agents would choose an action that causes descent (for certain,

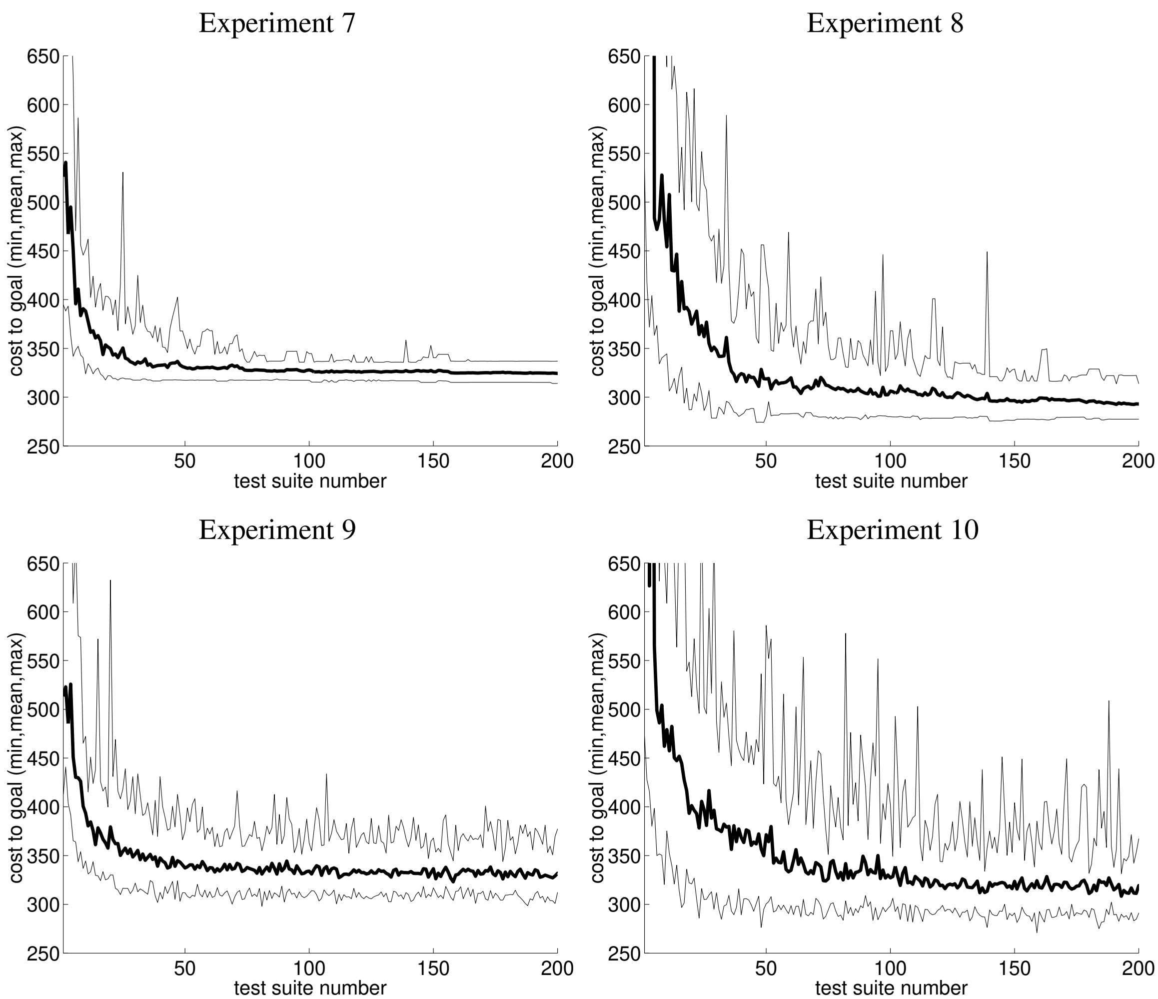


Figure 10: Mean, min, and max test suite costs across runs.

or with some probability). Thus, from Theorems 2 and 4, these agents were guaranteed to bring the arm to G_{arm} during learning trials, if given enough time.

6.3 Results and Discussion

Learning curves for the four experiments are displayed in Figure 10. We define the cost of a test suite (a set of nine test trials, one from each initial state) to be the average of the costs of the nine trials. The curves plot the minimum, mean, and maximum test suite costs, across runs, for the first 200 of 1,000 test suites. Using control law A_1 alone generates a test suite cost of 434 under the deterministic dynamics and 448, on average, under the stochastic dynamics.

Results for the two agents with the Lyapunov-based theoretical guarantees appear in the left column. Compared to the corresponding results in the right column, the Lyapunov-based designs exhibit superior initial performance, lower-variability performance, and more rapid learning. The agents in Experiments 8 and 10, however, show better performance in the long term. Comparing the curves of the deterministic-dynamics experiments in the top row with the curves in the bottom row

		Expt. 7	Expt. 8	Expt. 9	Expt. 10
Mean trial cost in first 10 suites	Learn	478 ± 7	806 ± 19	491 ± 6	862 ± 24
	Test	445 ± 12	$2{,}219 \pm 1{,}900$	458 ± 9	$1{,}168 \pm 950$
Mean trial cost in last 100 suites	Learn	353 ± 2.5	442 ± 4.3	360 ± 3.0	474 ± 4.9
	Test	320 ± 1.8	284 ± 1.5	327 ± 2.7	308 ± 1.5
Highest cost suites	Learn	749	2,301	818	1,952
	Test	820	161,710	768	55,364
Highest cost single trials	Learn	2,525	7,920	2,814	7,424
	Test	2,412	244,510	2,488	244,580
Longest trials (seconds)	Learn	11.7	83.8	12.4	63.9
	Test	10.9	timeout	12.4	timeout

Figure 11: Summary statistics for Experiments 7–10.

shows that the position disturbance in the stochastic dynamics has essentially no qualitative effect on learning. Its primary effect is to make performance during learning and test trials more noisy.

Summary statistics are presented in Figure 11. For the most part, these statistics support what is visible in the learning curves—better initial performance and lower variability performance for the agents with a Lyapunov-descent constraint, but better final performance for the unconstrained agents. One thing not readily observed from the learning curves is the powerful effect Lyapunov-descent constraints have on worst-case behavior. In the worst test suites and the worst individual test trials, agents without Lyapunov constraints incurred two orders of magnitude more cost than the constrained agents. There was a similar disparity in the durations of the longest trials for the two types of agents. As in the other experiments above, Experiments 7–10 demonstrate that constraining an agent to descend on a Lyapunov function is important in ensuring reasonable performance. Merely providing actions that are designed using Lyapunov analysis does not ensure good behavior.

7. Related Work

The methods for safe agent design that we propose in this paper rely on qualitative analysis ideas that have been under development for more than a century. Lyapunov originally developed the idea of a Lyapunov function late in the 19^{th} century. The idea that there is more than one way for a system to descend on a given Lyapunov function, and that this freedom can be used to optimize secondary criteria, dates back at least to Kalman and Bertram's early survey of Lyapunov function methods (1960). Subsequent work has used this observation in various ways, sometimes to optimize a cost function as we do here, and sometimes to produce controllers with improved stability or robustness properties. (See, for example, Krstić et al., 1995; Freeman and Kokotović, 1996; and Sepulchre et al., 1997, for work of this sort and further references.) Viewed in this light, the primary novelty of our work is the application of these ideas to learning systems, in which we are concerned with on-line learning behavior and cannot necessarily trust the behavior of a learned policy.

In the field of reinforcement learning there has been increasing interest in issues of safe agent design. In work quite similar to ours, Singh et al. (1994) propose an action formulation for a reinforcement learning agent in which action choice is a convex combination of the gradient directions of two different Lyapunov functions. In their robot motion planning setting, there are two natural candidates for Lyapunov functions, and the agent can follow arbitrary convex combinations of their two gradients and still enjoy collision-free control. In general, however, this method of combining Lyapunov functions does not retain the beneficial properties of either. Huber and Grupen (1997, 1998a,b) apply reinforcement learning to robotics problems in which the system state is defined by the status of a set of lower-level closed-loop controllers. Reinforcement learning is used to choose which low-level controllers to activate, but, from the start, actions that violate basic safety constraints are eliminated from the set of admissible actions. In spirit, this is quite similar to our approach; domain knowledge is used to constrain the agent to a set of safe behaviors. One important difference with our approach is that we do not stop at ensuring safety. Successful behavior, in terms of bringing the environment to a goal state or other desirable states, is also guaranteed.

Kretchmar (2000) has recently proposed a method that combines reinforcement learning with robust control theory to achieve safe learning of controllers for systems with partially-unknown dynamics. In simulation, his approach works well for realistic regulation problems. However, the forms of system dynamics and learning agents that are allowed by his theory are limited. Our approach applies to a much broader range of agents and environments. In its present form, however, our approach requires explicit knowledge of a Lyapunov function, making it difficult to treat problems in which the environment dynamics are not known a priori. We suggest some means to address this issue in our discussion of future work.

Gordon (2000) has recently proposed learning methods in which each control improvement suggested by learning is first verified for safety using model-checking techniques. Like the work of Kretchmar, and unlike ours, her approach uses on-line verification to ensure learning never results in an unsafe system.

Schneider (1997) and Neuneier and Mihatsch (1999) propose learning methods that pay attention to the variability of outcomes, resulting in learned controllers that are risk-averse. Although there are no theoretical guarantees on the performance of these learning algorithms, the goal of more reliable, safer learning is the same as ours.

8. Conclusions

We have shown that by relying on qualitative control design methods, one can construct reinforcement learning agents that provably enjoy basic safety and performance guarantees. Our primary approach relies on constructing sets of base-level control laws using Lyapunov domain knowledge. These control laws are designed so that any policy of switching among them enjoys basic safety and performance guarantees. With this approach, any reinforcement learning algorithm can be applied to the problem of learning a high-quality control policy, and reasonable performance is maintained at all times. Our work has focused on Lyapunov-based design methods, which are primarily applicable to stabilization, regulation, and tracking problems. However, we have also demonstrated the utility of other control design methods, such as approximating nonlinear dynamics by linear functions, feedback linearization, and linear-quadratic regulator control. Our general contention is that, in approaching a reinforcement learning problem, it is a mistake to ignore the wide variety of analytical tools and design methods that have been developed by control theorists, engineers, and roboticists. Incorporating these ideas into the designs of reinforcement learning agents allows for safer, more reliable, and more rapid learning, and renders tractable problems that are very difficult or impossible to solve with naive agent designs.

9. Future Work

In Section 3.1 we presented two theorems that are useful for designing safe reinforcement learning agents. One of these is mainly relevant to deterministic environments, while the other is applicable to stochastic environments as well. This second theorem is somewhat atypical in comparison with the stochastic stability literature (for example, Kushner, 1971, Meyn and Tweedie, 1994). In particular, our assumption of a global upper bound on the Lyapunov function is unusual. We have found this theorem useful for our example domains—its conditions are easy to check—and it seems a natural way of extending Theorem 1 to the stochastic case. However, other types of descent conditions might be more appropriate for other domains. For example, the supermartingale property requires that the expected value of the Lyapunov function at the next state is less than or equal to its value at the current state. Kushner (1971) and Meyn and Tweedie (1993) describe this and a number of other stochastic descent conditions from which useful qualitative safety and stability properties can be deduced. Developing versions of these conditions that can be applied to reinforcement learning agents is one topic for future investigation.

Being able to construct a safe space of behaviors for an agent opens up new possibilities for anytime approximate optimal control. In particular, we are presently looking at the grid-based approximate dynamic programming approaches that are often applied to continuous-state control problems of the sort we considered in this paper (Kushner and Dupuis, 1992, Fleming and Soner, 1993). Using fine grids, one is able to compute near-optimal policies, but there is difficulty in scaling these approaches to high-dimensional problems. Using a variety of tricks, including intelligently-allocated variable-resolution grids, Munos and Moore (2002) recently reported being able to solve problems with at most four- or, to a degree, five-dimensional state spaces. With coarse grids, one can approach problems of higher dimension. But coarse grids may result in no solution at all (for example, if a goal set appears unreachable from some grid point) or may result in a policy of very poor quality, possibly one that incurs infinite cost (Gordon, 1999). If, however, one can use qualitative domain knowledge to restrict attention to a space of qualitatively satisfactory policies, then we know that a grid-based approximate dynamic programming algorithm would produce a working solution for any resolution grid. We are presently exploring the possibilities of solving high-dimensional problems with coarse grids and of solving problems in anytime fashion using a sequence of grids of increasing resolution.

In the examples we presented, we assumed that the system dynamics were known so that Lyapunov analysis and other control methods could be applied. When the dynamics are known, on-line learning is not strictly necessary. Learning can be done in simulation and then safety during learning is not an issue. (Of course, we still may be interested in determining whether a policy resulting from off-line learning satisfies safety properties.) Lyapunov methods are desirable, in part, because of their robustness to certain kinds of unmodeled dynamics or disturbances. Robust control theory, however, explicitly deals with problems in which the system dynamics are not completely known. Robust control Lyapunov functions, in particular, are functions that are simultaneously Lyapunov for a whole set of control problems (Freeman and Kokotović, 1996). In situations where an agent does not know, a priori, what problem it will face, on-line learning is desirable because the agent can tune its behavior to perform well on the particular problem it encounters. If one knows that the problem the agent will face comes from some class, and if one can identify a robust control Lyapunov function for that class, then we conjecture that it is possible to design a safe reinforcement learning agent for that problem class using essentially the same approach we have proposed here.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant No. ECS-0070102 and ECS-9980062. 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 was also supported by a graduate fellowship from the University of Massachusetts Amherst. We are greatly indebted to Sascha Engelbrecht, who initiated this line of research while he was a postdoc associated with our laboratory. We also thank Michael Rosenstein, Doina Precup, and Daniel Bernstein for many useful discussions and for reading drafts of this paper and previous papers on this work, and we thank the anonymous reviewers for their feedback on the submitted version of the paper.

References

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

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

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

M. S. Branicky, T. A. Johansen, I. Peterson, and E. Frazzoli. On-line techniques for behavioral programming. In *Proceedings of the IEEE Conference on Decision and Control*, 2000.

C. I. Connolly and R. A. Grupen. The applications of harmonic functions to robotics. *Journal of Robotics Systems*, 10(7):931–946, 1993.

J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley, 1989.

R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262, 1998.

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

W. H. Fleming and H. M. Soner. *Controlled Markov Processes and Viscosity Solutions*. Springer-Verlag, 1993.

R. A. Freeman and P. V. Kokotović. *Robust Nonlinear Control Design: State-Space and Lyapunov Techniques*. Birkhäuser, Boston, 1996.

D. F. Gordon. Asimovian adaptive agents. *Journal of Artificial Intelligence Research*, 13:95–153, 2000.

G. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, 1999.

S. Grossberg and M. A. Cohen. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:815–826, 1983.

V. Gullapalli. *Reinforcement Learning and its Application to Control*. PhD thesis, University of Massachusetts Amherst, 1992.

J. Hopfield and D. W. Tank. ‘Neural’ computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

M. Huber and R. A. Grupen. A feedback control structure for on-line learning tasks. *Robots and Autonomous Systems*, 22(2–3):303–315, 1997.

M. Huber and R. A. Grupen. A control structure for learning locomotion gaits. In *Proceedings of the Seventh International Symposium on Robotic and Applications*, 1998a.

M. Huber and R. A. Grupen. Learning robot control—using control policies as abstract actions. In *NIPS’98 Workshop: Abstraction and Hierarchy in Reinforcement Learning, Breckenridge, CO*, 1998b.

R. E. Kalman and J. E. Bertram. Control system analysis and design via the “second method” of Lyapunov I: Continuous-time systems. *Transactions of the ASME: Journal of Basic Engineering*, pages 371–393, 1960a.

R. E. Kalman and J. E. Bertram. Control system analysis and design via the “second method” of Lyapunov II: Discrete-time systems. *Transactions of the ASME: Journal of Basic Engineering*, pages 394–400, 1960b.

R. M. Kretchmar. *A Synthesis of Reinforcement Learning and Robust Control Theory*. PhD thesis, Colorado State University, 2000.

M. Krstić, I. Kanellakopoulos, and P. Kokotović. *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., New York, 1995.

H. Kushner. *Introduction to Stochastic Control*. Holt, Rinehart and Winston, Inc., New York, 1971.

H. J. Kushner and P. Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time*. Springer-Verlag, 1992.

W. S. Levine, editor. *The Control Handbook*. CRC Press, Inc., Boca Raton, Florida, 1996.

S. Meyn and R. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, New York, 1993.

R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning Journal*, 49(2–3):291–323, 2002.

K. S. Narendra, N. O. Oleng', and S. Mukhopadhyay. Decentralized adaptive control. In *Proceedings of the Eleventh Yale Workshop on Adaptive and Learning Systems*, pages 59–68, 2001.

R. Neuneier and O. Mihatsch. Risk sensitive reinforcement learning. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 1031–1037, Cambridge, MA, 1999. MIT Press.

T. J. Perkins. *Lyapunov Methods for Safe Intelligent Agent Design*. PhD thesis, University of Massachusetts Amherst, 2002.

T. J. Perkins and A. G. Barto. Heuristic search in infinite state spaces guided by Lyapunov analysis. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 242–247, San Francisco, 2001a. Morgan Kaufmann.

T. J. Perkins and A. G. Barto. Lyapunov-constrained action sets for reinforcement learning. In C. E. Brodley and A. P. Danyluk, editors, *Machine Learning: Proceedings of the Eighteenth International Conference*, pages 409–416, San Francisco, 2001b. Morgan Kaufmann.

Elon Rimon and Daniel E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–517, 1992.

J. G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1047–1053, Cambridge, MA, 1997. MIT Press.

R. Sepulchre, M. Janković, and P. V. Kokotović. *Constructive Nonlinear Control*. Springer-Verlag, 1997.

S. P. Singh, A. G. Barto, R. Grupen, and C. Connolly. Robust reinforcement learning in motion planning. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 655–662, Cambridge, MA, 1994. MIT Press.

E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer-Verlag, New York, 1990.

M. W. Spong. The swing up control problem for the acrobot. *Control Systems Magazine*, 15(1): 49–55, 1995.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press/Bradford Books, Cambridge, Massachusetts, 1998.

G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 1995.

G. J. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

J. N. Tsitsiklis and B. Van Roy. Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44(10):1840–1851, 1999.

T. L. Vincent and W. J. Grantham. *Nonlinear and Optimal Control Systems*. John Wiley & Sons, Inc., New York, 1997.

D. Weld and O. Etzioni. The first law of robotics (a call to arms). In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1042–1047, 1994.

W. Zhang and T. G. Dietterich. High performance job-shop scheduling with a time-delay TD(λ) network. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1024–1030, Cambridge, MA, 1996. MIT Press.