# Comparing the performance of controllers based on Reinforcement Learning and Nonlinear Analysis

Arish Alreja, Pavel Komarov, and Prabhudev Prakash

*Abstract*— **Tools from Nonlinear analysis, Optimal Control and Statistical learning feature prominently in the analysis and control of complex systems ranging from robotics to power grids. The motivation for this work was to compare and contrast these toolsets. We realized this goal by developing Nonlinear and model-free Reinforcement Learning (RL) controllers for an inverted pendulum on a cart system, with the control objective of keeping the pendulum upright.**

**Both controllers achieve the control objective under noiseless conditions: The Nonlinear controller achieves the task perfectly, whereas the RL Controller suffers from wobble rooted in the quantization noise that arises from its need for state discretization. Thus, by the standards of accuracy and efficiency, the Nonlinear controller outperforms the RL controller. Finer discretization of state space produces an RL controller with improved performance but extends learning time. Interestingly, the RL's discretization renders it robust to noise, so it consistently does better than the NonLinear controller under sufficiently turbulent conditions.**

**Our most substantial finding is that Q-values over the discretized state space approximate the negative of our analytically-derived Lyapunov Function. Aside from achieving control objectives, learning Lyapunov Functions with a stochastic model-free technique can be powerful and may inform nonlinear controller design for systems where analytical derivations of Lyapunov Functions are challenging. Our conclusion is that the tools of Nonlinear analysis, Optimal Control and Statistical Learning are complementary, not competing approaches.**

## I. INTRODUCTION

Nonlinear analysis, Optimal Control and Statistical learning are ubiquitous in analysis and control contexts. Robots, neuro-prosthetics, power grids, spacecraft, and countless other complex systems rely on the theory from these fields.

Advances in analyzing, modeling, and controlling Nonlinear Systems have led to significant advances in solving new, ever-more-challenging controls problems, while parallel advances in optimization, accompanied by inexpensive and accessible computational power, have led to broad adoption of machine learning techniques, with numerous applications to problems like navigation or ambulation.

At first glance, the contrast between deterministic analytic approaches and the approximate nature of probabilistic machine learning models may suggest competition between the methods. This impression provided the original motivation for this project.

Comparing and contrasting the performance of a nonlinear and a learned controller demanded that we combine knowledge from ECE 6552 with unfamiliar material well outside the scope of the course.

We chose to use Reinforcement Learning (RL) as the method for our learned controller because unlike most machine learning algorithms, RL is fairly general-purpose and can be initialized simply, without the requirement for millions of training examples. Apart from initializing state spaces and reward functions along with environmental interaction, it demands minimal (if any) customization/programming for individual problems. A crucial difference between RL and Nonlinear Analysis techniques is that it can be model-free, that is have no fore-knowledge of how the external world works and devote no internal memory to guessing the environment's state. By contrast, most analytic techniques are entirely reliant on a model of the system dynamics.

The mathematical underpinnings of RL are non-trivial, and thorough understanding is required in order to use the method effectively. To give ourselves time to understand this half of our project and to ensure that we had time to go beyond building a controller and focus on a deeper comparative analysis, we chose a well-understood Nonlinear System: the inverted pendulum on a cart. The RL algorithm is fairly generic and can scale without much need for customization to more complex systems.

Our experience validated some of our initial notions but also exposed us to new insights about how these different approaches to studying and controlling complex systems might interact with each other in a complementary way.

## II. THE INVERTED PENDULUM ON A CART

Before we could begin leveraging tools from Nonlinear Control and elsewhere to formulate controllers, we needed to put our system of interest in to a mathematical form recognizable to control engineers and researchers. That is, we had to express the dynamics of the system according to a set of differential equations and compose them in to a state space model.

To begin, we used a free-body diagram to discern the physics of the mechanical system.

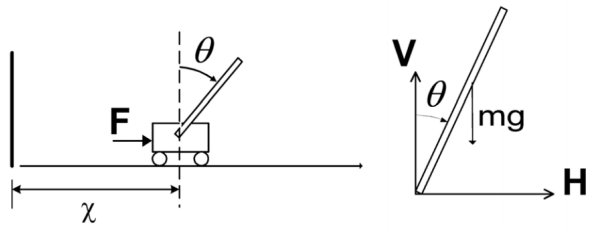



Fig. 1: The inverted pendulum on a cart system and relevant forces [4].

Given this picture, expressions for the forces on the pendulum can be written as

$$H = m\frac{d^2}{dt^2}(x + L\sin\theta)$$

$$V = m(\frac{d^2}{dt^2}(L\cos\theta) + g)$$

where $H = \Sigma$ forces in the horizontal direction, $V = \Sigma$ forces in the vertical direction, $m$ = the mass of the pendulum, $\theta$ = the angle of the pendulum from the vertical, and $L$ = distance from the pendulum's pivot to its center of mass.

Next, expressions for all torques around the pendulum's pivot and all horizontal forces on the cart can be written as

$$I\ddot{\theta} = VL\sin\theta - HL\cos\theta - c\dot{\theta} \tag{1}$$

$$M\ddot{x} = F - H - k\dot{x} \tag{2}$$

where $I$ = the pendulum's inertia around its center of mass ($\frac{1}{3}mL^2$ for a uniform rod), $M$ = the mass of the cart, $F$ = force applied to the cart, and $c, k$ = damping coefficients for rotation of the pendulum and movement of cart, respectively.

To obtain a state space model, we needed expressions for higher-order terms in terms of only lower-order terms. Because we care about cart and pendulum positions, we must solve the physics equations for $\ddot{x}$ and $\ddot{\theta}$.

Substituting for H and V in 1 and 2 and rearranging yields

$$\begin{bmatrix} I + mL^2 & mL\cos\theta \\ mL\cos\theta & M + m \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} mLg\sin\theta - c\dot{\theta} \\ F + mL\sin\theta\dot{\theta}^2 - k\dot{x} \end{bmatrix}$$

Then solving for $\ddot{\theta}$ and $\ddot{x}$ gives

$$\begin{bmatrix} \ddot{\theta} \\ \ddot{x} \end{bmatrix} = \frac{1}{d(x_1)} \begin{bmatrix} M + m & -mL\cos\theta \\ -mL\cos\theta & I + mL^2 \end{bmatrix} * \begin{bmatrix} mLg\sin\theta - c\dot{\theta} \\ F + mL\sin\theta\dot{\theta}^2 - k\dot{x} \end{bmatrix} \tag{3}$$

where $d(x_1) = (I + mL^2)(M + m) - m^2L^2cos^2\theta$

This is all that is necessary to express how the system evolves. These two equations are the basis for our state space model in terms of the lower-order terms $\begin{pmatrix} \theta & \dot{\theta} & x & \dot{x} \end{pmatrix}^T$ and for the forward-Euler computations in our MATLAB simulations. Both the Nonlinear controller and the RL make use of this same model.

## III. CONTROL SYNTHESIS

### A. REINFORCEMENT LEARNING

Since RL is not part of the ECE 6552 syllabus a brief note about its theoretical underpinnings is included in the Appendix. An excellent introduction by Sutton and Barto [1] can also be found online for free. The content in this section assumes familiarity with basic terminology associated with RL. We discuss specifics pertaining to the RL algorithm (Q-Learning) we used, its implementation and the learning process required to train an effective controller.

*Q-Learning–The Algorithm:* We chose to use the Q-learning algorithm because the Q function includes action choices as an input. This gives an RL Agent control over action selection and makes it easier to introduce Algorithm 2 for action selection. Algorithm 2 supports state space exploration during the learning process by allowing us to configure (using exploration probability $\epsilon$) how often a random action is chosen.

Owing to its Markov Decision Process roots, the algorithm requires state and action space discretization which ensures that Q-values must be learned for only a finite number of state-action pairs. Effectively, the algorithm implements a Gradient Descent learning step for the Q-values associated with each state-action pair.

It is interesting to recognize that instead of a dataset, the Agent executing this algorithm receives training samples that are determined by its environment (physical dynamics of the system), the Agent's initial conditions $(s_0, a_0)$ and the Agent's course of actions $a_1....a_n$ as its state trajectory evolves $(s_1...s_n)$. Upon observing this, we came to think of RL as a form of supervised (by the environment) learning conducted online (1 sample at a time). Given a sufficient number of samples for a given state-action pair, its Q-Value is guaranteed to 'eventually' converge, which means the Agent/Controller should continue to get better at keeping the pendulum upright. From Step 7,8 of Algorithm 1 it is worth noting and easy to imagine that Q-values for state-action pairs that are adjacent to each other (in terms of transition probabilities) are dependent upon each other, so this is how new information propagates through an Agent's Q-table. When all new information is assimilated, the algorithm will approximate the Q function well, and all Q-values will converge.

**Algorithm 1** Q–learning

1: Initialize $Q(s, a)$ randomly
2: **for** each episode **do**
3: $s \leftarrow$ random state
4: **for** each step **in** the episode **until** termination **do**
5: Choose $a$ per Action Selection Policy
6: Take action $a$, observe $(r, s')$
7: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max a' Q(s', a') - Q(s, a)]$
8: $s \leftarrow s'$

**Algorithm 2** Action selection policy

1: $n \leftarrow \mathbb{U}[0, 1]$
2: **if** $n > \epsilon$ **then**
3: Select $a$ with maximum Q value
4: **else**
5: Select $a$ randomly from available actions

*Q-Learning–The Implementation:* Q-Learning can be implemented using a Q-table or a deep neural network. We developed both implementations, though only the Q-table implementation worked successfully. Much easier to debug and visualize than black-box neural nets, a literal table was much less problematic. All our code was developed in MATLAB and uses standard MATLAB toolboxes.

*Q-Table:* The Q-table is a look-up table indexed by (s,a) that contains randomly initialized Q values for each state-action pair. The learning steps described in the Algorithm are implemented every time the environment gives feedback $(r, s')$ to the Agent. To set up the learning problem, the concept of an absorbent state is necessary. An absorbent state means that the system has gone into an unacceptable state and the simulation (or actual physical system) must terminated (and reset) to an acceptable initial condition. Figure 2 depicts a Q-table and how the absorbent (badly terminated) states affect the other states. In our case, we considered "the cliff" to be the pendulum exceeding

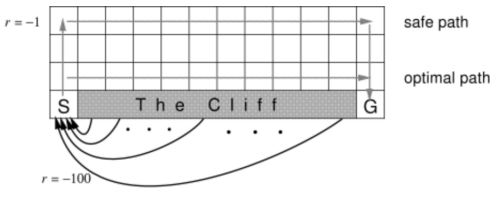
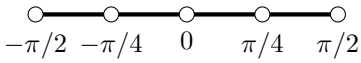



Fig. 2: Depiction of absorbing states (The Cliff) influencing the optimal action selection policy. [1]

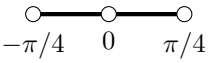
$|\theta| > \pi/2$ or $|\dot{\theta}| > \pi/4\ rad\ s^{-1}$

State Space Discretization: Two states $(\theta, \dot{\theta})$ are discretized in our implementation. While we found that RL could also additionally learn to keep the cart's traversal and velocity $(x, \dot{x})$ within some bounds, we did not pursue additional states under constraint because they were only peripherally related to our motivating questions. The coarsest discretization which allowed the controller to learn is given below. We eventually moved on to finer-grained state discretization for comparative analysis.

Discretization for $\theta$ (4 bins)




Discretization for $\dot{\theta}$ (2 bins)




Action Space Discretization: The force applied to the cart was the only action. We chose pairs of forces, which had equal magnitude but opposing direction (sign). The most bare-bones configuration that met the objective included only $F = -10N, 10N$. We eventually moved to more granular and numerous action choices for comparative analysis.

Together this means our initial RL Controller had to learn Q values for $4 \times 2$ states $\times$ 2 actions $= 16$ different state-action pairs.

Rewards: We initially tried 'Lyapunov-like' reward functions $(-\theta^2, -\theta^2 - \dot{\theta}^2)$, but upon reading an analysis of the problem in [1] we found an approach suggesting a reward of 0 or -1 depending upon whether the pendulum had survived (not fallen over) or entered an absorbent state (below the horizontal). This reward policy was attractive because it did not make any assumptions about the system. This 'minimalist' reward function suffices, since the negative reward gathered at absorbent states propagates back up to other states which are closer to the absorbent ones (in terms of transition probability) because of the nature of the Q-learning update rule (Step 7. Algorithm 1). Some of our eventual results (discussed later) are exciting particularly because of this *system-agnostic reward policy*.

*Training A Q-Learner:* Our standard training configuration included individual simulation episodes limited to 300 seconds followed by a successful (reward =0) termination. We used an initial $\epsilon = 0.5$ and let it decay exponentially, a learning rate $\alpha = 0.25$, and a discounting factor, $\gamma$, which balances the relative im-

portance of immediate rewards versus future cumulative rewards, of $0.8$. In this configuration, the RL Controller usually learned how to hold the pendulum upright for an episode after 50 to 70 episodes of learning. To speed convergence, we began training by initializing the system only marginally away from the upright position, increasing that margin as the learner mastered the last. Also for speed, we initialized the Q-Table to 0 rather than randomly, since our reward function should eventually push any entry to [0,-1] anyway.

While training numerous models with different levels of state discretization, we discovered several ways to hasten learning which have meaningful implications for real world systems (but may not be feasible for all systems). For example, if the controller can survive (the pendulum does not drop below $|\pi/2|$) for 20-30 real seconds, then it is almost certain to survive for 300 seconds. So, in order to save compute time, we restricted episode length to 20 seconds. We also noticed that randomly initializing each episode with $\theta, \dot{\theta}$ spanning the state space rather than relying upon the 'exploration' from Algorithm 2 could achieve stable operation five times faster than the initial scheme. So, we turned off Algorithm 2 by setting $\epsilon = 0$. We also varied $\alpha$ but did notice a substantial influence on learning rate. When dealing with more complex systems, some (or all) of these intuitions about state space exploration could save hours of training time.

The final RL controller demonstrated during our presentation was based on 77 unique states $(\theta, \dot{\theta})$ and 11 different Actions (10 non-zero forces of opposing signs for 5 different magnitudes and a 0 force), which gives a Q-table with 847 Q-values. Although we trained it extensively, it converged to stable operation within about 45 episodes limited to a maximum of 20 seconds of real time per simulation. This underscores the power of this technique.

### *B. NONLINEAR CONTROL*

To develop a nonlinear controller, we explored the mathematics, the theory upon which nonlinear control is founded, in the context of our system of interest.

To begin, we had to go one step beyond the two differential equations in 3 to formulate a state-space model in the familiar affine form with an output expressing the goal. Letting $\begin{pmatrix}\theta & \dot{\theta} & x & \dot{x}\end{pmatrix}^T = \begin{pmatrix}x_1 & x_2 & x_3 & x_4\end{pmatrix}^T$, letting $F$ be the control input $u$, and expressing the goal as driving $y = \theta$ to zero, we get the result 4 at the top of the next page.

Now the begins the road to a controller.

#### *Feedback Linearization*

The idea here is that the output dynamics $\eta$, which are the output $(y)$ and its derivatives, should be forced to zero as time advances. After taking the derivatives, we can rewrite the output dynamics in a linear form as

$$\dot{\eta} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}}_{F \in \mathbb{R}^{\gamma \times \gamma}} \eta + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}}_{G \in \mathbb{R}^{\gamma \times 1}} v$$

where $\gamma$ is the "relative degree" of the system and $v$ is the control input to this new system.

Choosing $v = -\begin{bmatrix}\alpha_1 & \alpha_2 & \dots & \alpha_\gamma\end{bmatrix}\eta$, where $\alpha_x$ are the coefficients of a polynomial $s^\gamma + \alpha_{\gamma-1}s^{\gamma-1} + \dots \alpha_1 s + \alpha_0$ with zeros in the open left-half plane, makes the system in to the form $\dot{\eta} = A\eta$, where all of A's eigenvalues have negative real part. In other words, choosing v in this way ensures the output dynamics are globally exponentially stable.

If we choose $u$ such that it cancels the nonlinearities of the system and enforces these linear output dynamics, then we can achieve stability. Or that's the idea.

So, let's take those derivatives.

$$y = h(x) = x_1 = \eta_1$$

$$\dot{y} = \frac{\partial h(x)}{\partial x}\dot{x} = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}(f(x) + g(x)u)$$

$$= \underbrace{x_2}_{L_f h(x) = \eta_2} + \underbrace{0}_{L_g h(x)} u$$

Since $L_g h(x)$ is not nonzero, we have to differentiate again

$$\ddot{y} = \frac{\partial L_f h(x)}{\partial x}\dot{x} = \begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}(f(x) + g(x)u)$$

$$= \underbrace{f_2(x)}_{L_f^2 h(x)} + \underbrace{g_2(x)}_{L_g L_f h(x)} u$$

Now, since we have reached a nonzero coefficient of $u$ in two differentiations, we know the relative degree $(\gamma)$ of the system is 2 and can formulate the control law that enforces linear, exponentially stable output dynamics:

$$u = \frac{1}{L_g L f h(x)}(-L_f^2 h(x) + v)$$

where, as stated earlier, $v$ is chosen such that the output dynamics are stable. $v = -\begin{bmatrix}1 & 2\end{bmatrix}\eta = -y - 2\dot{y} = -x_1 - 2x_2$ works, and the control law is complete.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \underbrace{\begin{bmatrix} x_2 \\ \frac{1}{d(x_1)}(M+m)(mLg\sin x_1 - cx_2) + \frac{1}{d(x_1)}(-mL\cos x_1)(mL\sin x_1 x_2^2 - kx_4 \\ x_4 \\ \frac{1}{d(x_1)}(-mL\cos x_1)(mL\sin x_1 - cx_2) + \frac{1}{d(x_1)}(I+mL^2)(mL\sin x_1 x_2^2 - kx_4 \end{bmatrix}}_{f(x)} + \underbrace{\begin{bmatrix} 0 \\ \frac{-mL\cos x_1}{d(x_1)} \\ 0 \\ \frac{I+mL^2}{d(x_1)} \end{bmatrix}}_{g(x)} u \quad (4)$$

$$y = x_1$$

*Zero Dynamics*

But that is not the whole story. The law guarantees the output dynamics are exponentially stable, but the system as a whole may still not be. Unless the system is full-state linearizable ($\gamma = n$), there are other dynamics at play that the controller can not affect. These are called the "zero dynamics", z, and are "orthogonal" to the control input. That is $Dz(x)g(x) \equiv 0$.

Frobenius' theorem establishes the existence of more than $n - \gamma$ solutions to that condition, so we can find enough of them to complete a transformation of coordinates from the original system to the ($\eta$, $z$) domain. It is not necessary to always find these solutions, but they can be informative, so we have done so.

Since $g(x)$ is 0 at its first and third indices, a natural choice of $z(x)$ might be $\begin{bmatrix} x_1 & x_3 \end{bmatrix}^T$. This yields

$$\Phi(x) = \begin{bmatrix} \eta \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 \\ x_3 \end{bmatrix} \rightarrow D\Phi(x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

But since $D\Phi(x)$ is not full rank, $\Phi(x)$ can not be a diffeomorphism, so this is not a valid coordinate transformation! On a second attempt, choose $z(x) = \begin{bmatrix} x_3 & (I+mL^2)x_2 + (mL\cos x_1)x_4 \end{bmatrix}^T$ This yields

$$D\Phi(x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -mLx_4 \sin x_1 x_2 & (I+mL^2) & 0 & mL\cos x_1 \end{bmatrix}$$

which is full rank everywhere $\cos x_1 \neq 0$, that is $\theta \in (-\pi/2, \pi/2)$, the entire upward-facing region.

Now, since we know $\eta \rightarrow 0$, we know the only thing left is whatever evolution the system is undergoing orthogonal to the control input. That is, we have stabilized the system (exponentially) to the zero dynamics. So what do those dynamics do as time advances? A bit of algebra yields

$$\lim_{t\to\infty} \dot{z} = \frac{\partial z}{\partial x}\dot{x}\bigg|_{\eta=0} = \begin{bmatrix} x_4 \\ 0 \end{bmatrix}$$

That is, $\dot{z_1}$, the position of the system keeps changing by $x_4$, the velocity, and $\dot{z_2}$, which is related to velocity, goes to zero. So we expect the controller to drive acceleration to zero and balance the pendulum in an inertial frame going a constant velocity.

*Control Lyapunov Functions*

But that's not all! It is possible too to control the system with a Control Lyapunov Function (CLF), and for feedback linearizable systems, finding the right Lyapunov function is formulaic.

We know that the output dynamics of the system are given by a linear system. Let the Hurwitz matrix that keeps that system stable be called $F_{CL}$. A solution, $P = P^T > 0$, to the Continuous Time Lyapunov Equation

$$F_{CL}^T P + P F_{CL} = -Q$$

where Q is also positive definite and symmetric, yields a CLF $V(x) = \eta(x)^T P \eta(x)$.

For our system, a valid $F_{CL}$, the one that puts both eigenvalues at -1, is given by

$$\begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix}$$

Choosing Q to be a multiple of the identity matrix, a solution to the CTLE is given by

$$Q = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \rightarrow P = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}$$

This gives

$$V(x) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 3x_1^2 + 2x_1x_2 + x_2^2$$

We know that for a Lyapunov function to prove exponential stability, it (1) must be between two class-k functions in $\|\eta\|$ and (2) must have a derivative less than some negative constant times itself. The first condition is clearly true in our case since $\frac{1}{2}(x_1^2 + x_2^2) < V(x) < 100(x_1^2 + x_2^2)$. The second relies upon the control input because

$$\dot{V}(x) = \frac{\partial V(x)}{\partial x}\dot{x} = \begin{bmatrix} 6x_1 + 2x_2 & 2x_1 + 2x_2 & 0 & 0 \end{bmatrix} \dot{x}$$

$$= (6x_1 + 2x_2)x_2 + (2x_1 + 2x_2)f_2(x) +$$

$$(2x_1 + 2x_2)g_2(x)u < \frac{\lambda_{min}(Q)}{\lambda_{max}(P)} * V(x)$$

We can find the "best" control law that satisfies this condition by formulating the problem as a quadratic program (QP) with a cost function that encodes some notion of what it means for a controller to be good. Often, the best controller is the one that achieves its aim with the least effort, so we can write the QP as

$$u^*(x) = \underset{u \in \mathbb{R}^m}{\operatorname{argmin}} \; u^T u$$

$$s.t.\; L_f V(x) + L_g V(x) u < -\alpha V(x)$$

This is not the only possible cost function. Actually, it is often preferrable to minimize $v^T v$ instead, transforming to a QP in $u$ by the relationship $u = A^{-1}(x)(-L_f^* h(x))$. But we tried that for this problem and found no benefit, and it is much easier solve $u^T u$ alone because that QP has a closed-form solution given by (when $\phi_0$ and $\phi_1$ are scalar)

$$u^*(x) = \begin{cases} \frac{-\phi_0}{\phi 1}, & \text{if } \phi_0 > 0 \\ 0, & \text{otherwise} \end{cases}$$

where $\phi_0 = L_f V(x) + \alpha V(x)$ and $\phi_1 = L_g V(x)$.

This controller may be considered better than that derived by simple feedback linearization because it encodes the notion of minimum effort. Whereas a feedback linearizing controller will always override the system's nonlinearities to enforce $\eta \to 0$, the CLF-based controller will drops to zero when the dynamics of the system work in its favor.

In our case, the dynamics never work in favor of the control objective, so the two analytically-derived controllers act nearly exactly the same. So in our later analysis we have used the CLF-based controller exclusively. All references to "the nonlinear controller" refer to this one.

## IV. RESULTS

We compared the analytically-derived controller and reinforcement-learned controllers according to two metrics: Resilience to noise and efficiency.

All simulations were performed using a 1 kg cart and a 0.1 kg pendulum of length 0.5 meters. The results were collected from simulated episodes limited to a maximum of 300 seconds of real time. The timestep used for forward-Euler integration was 0.01 seconds.

*Simulation Videos:*

- Reinforcement Learner-based Controller
- Non-Linear, CLF-based Controller
- 360 degree visualization of Learned Q-Function alongside negative of the derived Lyapunov Function.

The first result is unsurprising: It is clear that the CLF controller is more efficient than the RL controller. This is supported by 100 simulations of each controller, which show the magnitude of the average force utilized to balance the inverted pendulum is 1.8827 Newtons for the CLF controller and 4.3881 Newtons for an RL controller with 11 discrete forces. This gap is partially due to that discretization and the associated quantization noise, and, as we have seen that less-fine discretization yield worse results, we postulate that finer discretization might shrink the gap. But the CLF also does better than the RL because the RL hasn't been trained for infinite time and so has imperfect Q-values. Since optimal Q-values become more time-consuming to obtain as the size of the state-space is increased, there is a tradeoff between good Q-values and a large action space.

The second result is that noise, specifically process noise, is handled better by the RL controller than by the CLF controller. Figure 3 shows the average mean-squared-error of $(\theta)$ for both controllers over different values of $\beta$, where $\beta$ is the scale of a noise-term added at model update-steps as

$$x = x + \dot{x} * dt + \beta * randn * scales$$

where $scales$ is a vector of arbitrary weights $\begin{bmatrix} \frac{\pi}{4} & \frac{\pi}{18} & 0.2 & 0.5 \end{bmatrix}^T$ corresponding to the state $\begin{pmatrix} \theta & \dot{\theta} & x & \dot{x} \end{pmatrix}^T$ and $randn$ is a random number from a normal distribution with mean of 0 and standard deviation of 1.

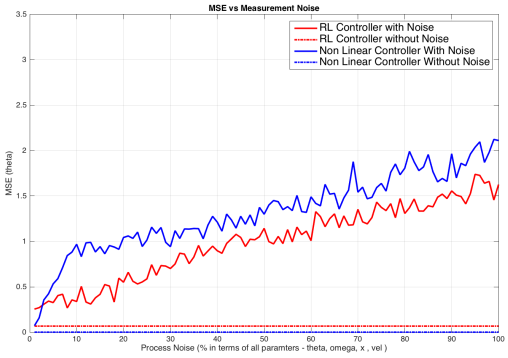



Fig. 3: MSE $(\theta)$ vs Process Noise

Figure 5 depicts the average episode duration, that is how long a simulation survives, versus $\beta$. In both cases, the CLF-based controller fares better under little

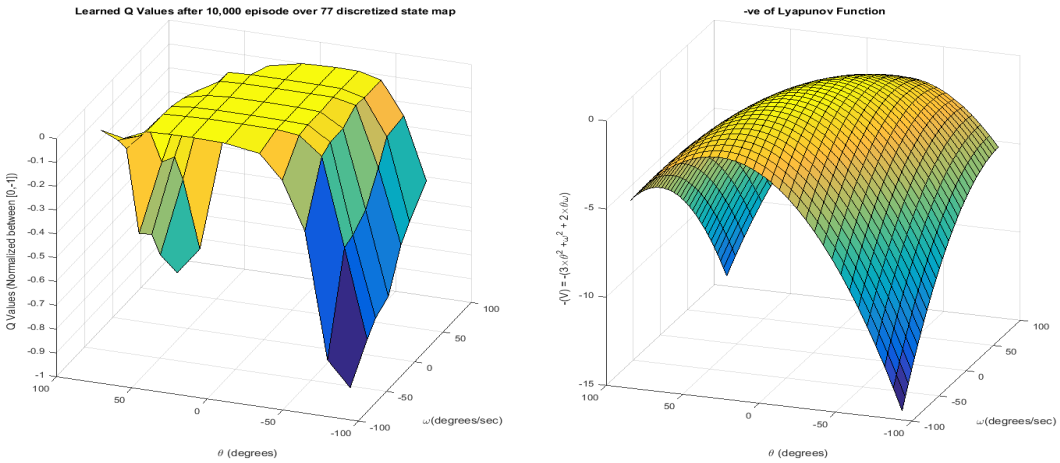



Fig. 4: The Q-values and -V plotted side by side over the state space $(\theta, \dot{\theta})$

noise, but its performance degrades faster than the RL controller's as beta increases. A possible explanation for this, is that the RL controller discretizes the state space into bins, bins that are likely to describe the system's evolution plus noise, whereas the nonlinear controller operates entirely precisely.

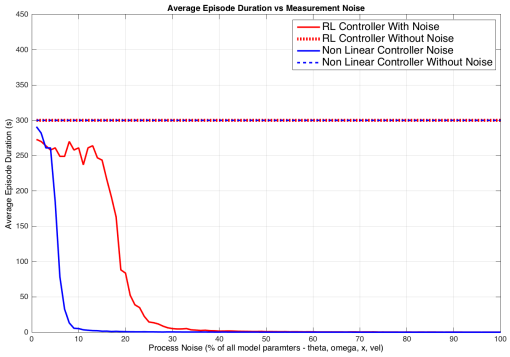



Fig. 5: Average Episode Duration(s) vs Process Noise

Lastly, perhaps the most compelling result from our comparative study is that the Q-values learned via RL resemble the Lyapunov Function derived using the methods of Nonlinear Analysis as shown in Figure 4. If this generalizes beyond our simple-system, curve-fitting a function to an RL's Q-table could be a powerful tool to guide design CLF-based controllers for systems that resist analytical tractablility. The best part of this result for us was that it emerged from a system-agnostic reward scheme: The learner knew nothing of the natural environment aside from the fact that letting the pendulum fall was bad, yet it was able to distribute that information over the whole state-space in such a way that its notion of "goodness" for a state is related to that state's energy. This suggests that Lyapunov Stability theory is a truth commonly reflected in nature.

## V. CONCLUSIONS

With a very simple reward scheme, the RL controller converges to an optimal value function, which makes Reinforcement Learning-based implementation attractive. The value function generated by our Reinforcement learning scheme takes the shape of Lyapunov function (Figure 4) without any coercion on our part.

Because the Reinforcement Learner is based upon bucketing and must learn to withstand quantization noise by design, the RL controller is more robust to noise. But in less-noisy environs, this advantage quickly turns to disadvantage, as an analytically-derived controller is far more efficient and accurate.

Thus, we can conclude that analytical approaches are superior in domains requiring accuracy and efficiency, but RL-based methods are powerful under noisy conditions and, since they require no deep analysis, can be useful in domains where such analysis is impractical.

Moving beyond a mere comparison of these approaches, we would like to suggest they can be used conjointly to solve a single problem. Particularly, for control objectives, at least those requiring stabilization to an energetically-critical point, it appears possible to find a CLF by curve-fitting the Q-values of an RL. Such a CLF could be used to very simply find a controller for the system and would not require any of the complicated analysis or guessing involved in deriving one directly. Future work might focus on the questions like "Do our observations generalize to higher dimensional, more complex systems?" or "What kinds of control objectives can be achieved by this hybrid approach?"

## REFERENCES


[1] Sutton, Richard S and Barto, Andrew G, Introduction to reinforcement learning,vol. 135, MIT Press Cambridge, 1998.

[2] MistWiz (Wikimedia Commons),Markov Decision Process example, 2006, https://commons.wikimedia.org/wiki/File:Markov_Decision_Process_example.png

[3] Khalil, H. K., & Grizzle, J. W. (1996). Nonlinear systems (Vol. 3). New Jersey: Prentice hall.

[4] Bugeja, M. (2003, September). Non-linear swing-up and stabilizing control of an inverted pendulum system. In EUROCON 2003. Computer as a Tool. The IEEE Region 8 (Vol. 2, pp. 437-441). IEEE. https://www.researchgate.net/publication/4045336_Non-linear_swing-up_and_stabilizing_control_of_an_inverted_pendulum_system


## APPENDIX

This section provides a summary of our understanding of the theoretical underpinnings of Reinforcement Learning. It is included here for completeness because it is not part of the ECE 6552 Syllabus. For interested readers, an excellent textbook by Sutton and Barto [1] can also be found online for free. Here we discuss the basic framework of Markov Decision Processes, relate them to reward maximization problems and discuss how the Bellman Inequality helps arrive at recursive forms that can be used to determine optimal solutions in a computationally feasible manner. We end with describing how Reinforcement Learning descends from these principles to provide approximate solutions for very large/not fully observable Markov Decision Processes.

Markov Decision Processes provide a modeling framework for decision-making when facing stochastic outcomes. A decision making agent can transition its way across discrete states based on its choice of actions while accumulating rewards associated with state transitions. Markov Decision Processes adhere to the Markov Property, which ensures that the process is memory less i.e Transition from state $s$ to $s'$ via action $a$ has probability $P_a(s, s')$ and is independent of preceding states or actions.

Figure 6 illustrates a Markov Decision Process viewed by an agent as a a collection of states $S$. At time $t$ in state $s_t$, the agent decides what the best action $a_t$. The environment takes $< s_t, a_t >$ and produces a new state $s'$. This step simulates the world responding to the agent's action $a$ and is accompanied by a reward $r(s_t, a_t) \in \mathbb{R}$, The reward $r(s_t, a_t)$ defines how good the decision (to choose $a_t$) was. $R$ is the sum of rewards given a specific sequence of selected actions based on a policy $\pi$ given by Eq. 5. An agent focused on reward maximization will try to find an optimal policy $\pi^*$, which maximizes $R$.

$$R(\pi) = \sum_{t=0}^{\infty} R(s_t, a_t) \tag{5}$$

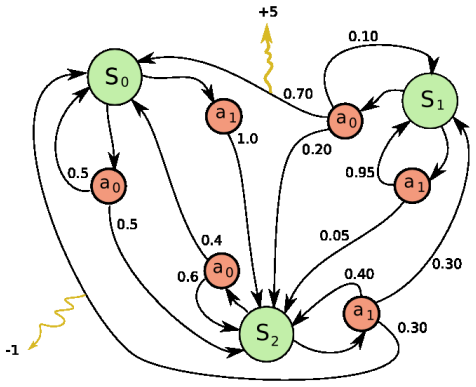



Fig. 6: Transition diagram for a small Markov decision process [2]

Markov Decision Processes provide a powerful framework for solving stochastic discrete control problems. However, their abstract form exposes some practical problems when an agent starts seeking out optimal policies to maximize reward.

Reward Accumulation and the Discounting Factor $\gamma$: Non-terminating sequences of actions and rewards can lead to reward accumulation approaching infinity over the lifetime of the process even if the goal is not achieved. This problem can manifest if there is a small reward for taking no action and staying in the same state. The Reward Accumulation problem is solved with the introduction of a discounting factor $\gamma < 1$, which guarantees that $R(\pi)$ will converge per Eq. 6.

$$R(\pi) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{6}$$

The following functions capture the notions of total reward while accounting for the probabilistic nature of Markov Decision Processes.

Value Function($V^\pi(s)$): Represents the expected value of the long term return associated with a state $s$ when acting according to a policy $\pi$. Described by Eq. 7

$$V^\pi(s) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \tag{7}$$

Q-Function($Q^\pi(s, a)$): Represents the Expected value of the cumulative return associated with performing action $a \in \mathbb{R}$ and in subsequent states $s \in \mathbb{S}$. Described by Eq. 8

$$Q^\pi(s, a) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \tag{8}$$

The Q-function encodes the rewards associated with following a certain policy at a certain point as well as

those from going off-policy at each state. As we will see later, this is crucial for Reinforcement Learning. Pursuing maximal reward will lead an agent to optimal policies $\pi^*$ which maximize for Q and V functions. The Markov property(memoryless-ness) allows relating optimal $Q$ and $V$ functions to the Bellman optimality equation [1] and a recursive relationship between $V(s)$ to $V(s')$ allows the unraveling of $V$ as shown below

$$V^{\pi}(s) = E_{\pi}\left\{R_t | s_t = s\right\}$$

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\Big| s_t = s\right\}$$

$$V^{\pi}(s) = E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\Big| s_{t+1} = s'\right\}$$

$$V^{\pi}(s) = \sum_a \pi(s,a)\sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}(\mathcal{R}^a_{ss'} + \gamma E_{\pi}(\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\Big| s_{t+1} = s'))$$

$$V^{\pi}(s) = \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right] \quad (9)$$

Where $\mathcal{P}$ is the probability of transitioning from $s$ to $s'$ when taking action $a$ and $\mathcal{R}$ is the return associated with doing the same. The Bellman equation for the Q-functions is as follows:

$$Q^{\pi}(s,a) = E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1})\Big| s_t = s, a_t = a\right\} \quad (10)$$

The solution to the reward maximization problem can be determined exactly when the entire Markov Decision Process is known. When this is not the case, only an approximate solution is possible. This can happen when a Markov Decision Process is large enough that complete knowledge is not practically possible to store.In such cases learning an approximate solution comes into play. This is called Reinforcement Learning.

With the assumption that the agent has an incomplete representation of the Markov Decision Process, it begins to explore the state space by interacting with the environment, traversing the state space and getting rewards. State Space exploration is necessary to avoid local maxima and achieving it requires exploring previous unknown spaces that are sub-optimal according to current policy. This requirement forms the basis of most Reinforcement Learning algorithms.

By discovering the optimal Q function $Q^*$ using Eq. 10 we can recover the optimal policy by picking actions which maximize Q in each state. The substructure of Eq. 10 opens the door to iterative techniques for finding the optimal Q Function.