Data-driven Design of Energy-Shaping Controllers for Swing-Up Control of Underactuated Robots

Wankun Sirichotiyakul and Aykut C. Satici

Abstract We propose a novel data-driven procedure to train a neural network for the swing-up control of underactuated robotic systems. Our approach is inspired by several recent developments ranging from nonlinear control theory to machine learning. We embed a neural network indirectly into the equations of motion of the robotic manipulator as its control input. Using familiar results from passivity-based and energy-shaping control literature, this control function is determined by the appropriate gradients of a neural network, acting as an energy-like (Lyapunov) function. We encode the task of swinging-up robotic systems through the use of transverse coordinates and goal sets; which drastically accelerates the rate of learning by providing a concise target for the neural network. We demonstrate the efficacy of the algorithm with both numerical simulations and experiments.

1 Introduction

Motivation. The swing-up control design of pendulum systems has been the subject of innumerable research papers until today. Design approaches ranging from bang-bang control, energy-based approaches, and many others have been extensively studied [1,8,9, 11]. These approaches require precise knowledge of the mathematical model describing the system. For systems with high complexity, development of controllers using these methods can be intractable.

Data-driven approaches are more flexible in the sense that the same framework can be applied to many different systems. In this category, the main idea is to use data to find a control policy such that the closed-loop behavior of the system optimizes a certain objective given by some notion of accumulative reward/cost. One of the most commonly used techniques in this area of research is reinforcement learning [18], which seeks a direct mapping from the system states to the control inputs.

Problem Statement. The majority of data-driven approaches treat the closed-loop system as a black-box, i.e. containing no predetermined structure from the governing laws of physics. While this is a flexible solution to many control problems, the vast amount of training data required for training is often prohibitive. Recent advances in machine learning research have shown that it is advantageous to combine available knowledge of the system with the learning framework. In this paper, we discuss the data-driven control design which directly incorporates the dynamics of the system, drastically increasing the rate of learning. The quintessential control problem that we tackle is the swing-up control of pendulum systems.

Related Work. In [5], the dynamics of the system is first learned and then later incorporated into a policy search. This addresses the poor sample complexity of model-free reinforcement learning, but does not allow physical structures of the system to be directly incorporated. Neural ordinary differential equation (ODE) [2], is a recently developed framework that connects deep neural networks to continuous-time dynamical systems. This approach has provided researchers with a modeling basis for incorporating physical structures into their machine learning problems, e.g. using neural ODEs to learn the Hamiltonian dynamics of physical systems [19].

The very recent extension to neural ODE [14] enables the direct incorporation of a neural network into the differential equation governing the evolution of system. We combine this neural-network-embedded ODE with passivity-based control techniques to tackle the swing-up control design problem. The main difference in our work and [19] is that we do not learn the Hamiltonian dynamics using a neural ODE. Rather, we leverage the known dynamics and train a neural network to learn an energy-like function, from which the controller is derived, thereby imposing desirable characteristics on the closed-loop system.

2 Technical Approach

The form of the controllers we design in this paper is based on passivity-based/energy-shaping control. We will borrow from [10, 15] to describe the general outline of this method. Let $x \in \mathbb{R}^{2n}$ denote the state of the robot. The state x may be represented in terms of the generalized positions and velocities $x = (q, \dot{q})$ or positions and momenta $x = (q, p)$. The system's equations of motion is

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix} \begin{bmatrix} \nabla_q H \\ \nabla_p H \end{bmatrix} + \begin{bmatrix} 0 \\ G(q) \end{bmatrix} u.$$

where $H(q,p) = \frac{1}{2} p^\top M^{-1}(q) p + V(q)$ is the Hamiltonian of the robot, M is the mass matrix, and $V(q)$ represents the potential energy. Energy-shaping control, which is closely tied to passivity-based control [15], uses the control input $u \in \mathcal{U} \subseteq \mathbb{R}^m$ to impose a desired (closed-loop) energy-like function: $H_d : \mathbb{R}^{2n} \to \mathbb{R}$. In [10], this function was chosen to be a quadratic function of the system momenta:

$$H_d(q,p) = \frac{1}{2} p^\top M_d^{-1}(q) p + V_d(q),$$

however, in this work we do not constrain ourselves to desired Hamiltonians of this particular form. Rather, we use a neural network to represent H_d and devise a learning framework to obtain the desirable characteristics automatically. From H_d, we then derive the controller, comprised of an energy-shaping term and a damping injection term as follows

$$u(x) = u_{es}(q,p) + u_{di}(q,p) = -(G^\top G)^{-1} G^\top \frac{\partial H_d}{\partial q} - K_d G^\top \frac{\partial H_d}{\partial p}. \qquad (1)$$

2.1 Main Learning Problem

To train the neural net, we formulate the control design for swinging-up robotic mechanisms as an optimization problem of $H_d(x;\theta)$. In particular, we have

$$\begin{aligned} \underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad & J(\theta) = \int_{t_0}^{t_f} r(\phi, u(\phi;\theta))\, dt, \\ \text{subject to} \quad & \dot{x} = f(x) + g(x)u(x;\theta), \quad x(t_0) = x_0. \end{aligned} \tag{2}$$

where $\theta \in \mathbb{R}^p$ denotes the neural network parameters, $\phi = \phi(t; x_0, \theta)$ is the flow of the equations of motion, the time horizon $t_f \in (0, \infty)$ is a hyperparameter and $r : \mathbb{R}^{2n} \times \mathbb{R}^m \to \mathbb{R}$ is a running cost function.

The running cost function $r(\phi, u)$ that yields the loss function $J(\theta)$ consists of 1) distance, $r_\perp$, to some preferred orbit, 2) set distance, r_{set}, between the current prediction and the goal set, 3) Double-hinge loss, r_{hinge}, to penalize multiple swings of the first link, and 4) the regularization term, r_{reg}, to avoid overtraining the neural network parameters. The running cost is found by the sum of these components:

$$r(\theta) := r_\perp(\phi, u) + r_{\text{set}}(\phi, u) + r_{\text{hinge}}(\phi, u) + r_{\text{reg}}(\theta),$$

where each term depends on θ through the dependence of the trajectory ϕ on θ. We now elaborate on the details of each component of r as follows.

1) Distance to preferred orbit: The first term that makes up the running loss is the distance to a preferred orbit, $\phi^\star$. In Section 3, where we apply this framework to the simple pendulum and inertial wheel pendulum, this preferred orbit was chosen to be the homoclinic orbit of the pendulum. In general, such a homoclinic orbit may not be available in closed-form. In this case, the preferred orbit may be computed using one of several methods that are available in the literature, such as trajectory optimization [4], virtual holonomic constraints [16], etc.

Once the preferred orbit has been chosen, the next step is to compute the *transverse coordinates* along this orbit, using the ideas outlined in [7, 17]. This method reparametrizes the states of the dynamical system by assigning one state, τ, to be a parameter that determines where along the specific trajectory the state of the system is. The remaining states, $x_\perp \in \mathbb{R}^{2n-1}$, are chosen transversal to the trajectory, with the condition $\phi \to \phi^\star$ if and only if $x_\perp \to 0$ as $t \to \infty$. We penalize the the prediction trajectory by incurring a quadratic loss on $x_\perp$ and on the control effort u:

$$r_\perp(\phi, u) = \frac{1}{2} x_\perp^\top Q x_\perp + \frac{1}{2} u^\top R u \tag{3}$$

2) Set distance: The ultimate goal of the framework is to swing-up the robotic system to an upward (unstable) equilibrium point. We encourage the controller to move the system states to a set around the upward equilibrium point by penalizing whenever the prediction trajectory spends time away from this set.

The penalty, r_{set} is constructed by defining a convex open neighborhood $\mathcal{S}$ of the upward equilibrium point and computing the set distance of the predicted closed-loop trajectory $\phi(t)$ to $\mathcal{S}$, i.e.

$$r_{\text{set}}(\phi) = \text{dist}(\gamma, \mathcal{S}) = \inf\{\|x - y\| : x \in \phi(t),\ t \in (t_0, t_f),\ y \in S\}. \tag{4}$$

For instance, the set $\mathcal{S}$ can be selected as a ball around the upward equilibrium point of radius $d > 0$ in the standard norm topology. In this case, d is to be selected by the user before the training. If any point along the prediction $\phi(t)$ gets closer than d to the upward equilibrium point, no additional loss is incurred.

3) Double-hinge loss: We want penalize those prediction trajectories ϕ which overshoot the upward equilibrium point and require the system to swing multiple times before reaching the set S using a double-hinge loss term, r_{hinge}. With q_i corresponding to the angle of the i^{th} link, we have

$$r_{\text{hinge}}(\phi, u) = \begin{cases} 0, & |q_i| \le 2\pi, \\ |q_i| - 2\pi, & \text{otherwise.} \end{cases} \tag{5}$$

2.2 Imposing Symmetry Constraints

For dynamical systems that are naturally symmetric about the origin, i.e. $f(x) = -f(-x)$, it often is desirable to maintain symmetry in the closed-loop system by designing a symmetric control policy. To obtain this in the learned control function of the form in eq. (1), where H_d is a neural network, we encode this information directly into the neural net by applying a symmetric function h to the input. To demonstrate, a two-layer neural net and its gradient with respect to the input x is are

$$H_d(x) = W_2\,\sigma(W_1 h(x) + b_1) + b_2,$$
$$\nabla_x H_d(x) = W_2\,\sigma'(W_1 h(x) + b_1) \odot W_1 \odot h'(x).$$

Selecting $h(x) = h(-x)$ and hence $h'(x) = -h'(-x)$, we obtain the desired symmetry, i.e. $\nabla_x H_d(x) = -\nabla_x H_d(-x)$, which is directly reflected in the control $u(x)$ in Eq. (2).

Imposing this symmetry constraint in the control function reduces the sample space over which the neural network has to train. This speeds up the training process and increases consistency in the behavior of the closed-loop system.

2.3 Gradients Computation through ODE Solutions

Solving the optimization problem (2), requires the gradient computation of the $J(\theta)$ through the solution $\phi(t; x_0, \theta)$ of an ODE. The main challenge ultimately lies in the computation of $\frac{\partial \phi}{\partial \theta}$. In the recent literature [2], adjoint sensitivity methods have been used to compute this derivative by first solving a backwards ODE, known as the adjoint problem:

$$\frac{\mathrm{d}\lambda}{\mathrm{d}t} = -\lambda \frac{\partial\left(f(x) + g(x)u\right)}{\partial x}.$$

Then, the desired gradient of the loss function is computed through the expression

$$\nabla_\theta J(\theta) = \lambda(t_0)\frac{\partial\left(f(x) + g(x)u\right)}{\partial x}.$$

This approach is based on Pontryagin's maximum principle [12] and is quite mathematically elegant. However, it requires multiple forward solutions of the ODE to implement; hence, it quickly becomes quite costly to execute.

Recent advances in automatic differentiation (AD) and differentiable programming has allowed more efficient computations of the desired gradients. In [14], a number of existing automatic differentiation schemes and adjoint methods are generalized to enable efficient gradient computation of solutions to ODEs with neural-net architectures embedded in them. We use this very recent framework, which is implemented in the programming language Julia within the package `DiffEqFlux.jl`, to compute the gradients of the loss function $J(\theta)$ and use it in conjunction with ADAM [6] to solve our optimization problem (2).

3 Results

We apply our approach to robots commonly used as benchmarks: the pendulum and the cart-pole. The data-driven control designs for both systems are performed and evaluated in simulations. The controllers are also tested on real systems.

We use a simple fully-connected neural network with two hidden layers and ELU activations [3]. The controller derived from H_d is used for swing up, and a standard LQR is used to stabilize the upward equilibrium. Simulation studies are conducted from a range of initial conditions near the downward equilibrium. If the linear controller is able to catch the swing and stabilize the upward equilibrium, then the simulation is considered successful. This requires the swing-up controller to bring the state into the region of attraction of the linear controller. Performance of the swing-up controller is evaluated by the success rate of the simulations.

3.1 Pendulum

The dynamics of the pendulum is given as

$$I\ddot{\theta} = -mgl\sin(\theta) - b\dot{\theta} + u \tag{6}$$

where $m = 0.377$ kg is the combined mass, $l = 23.2$ cm is the length from the pivot point to the center of gravity of the mechanism, $I = 0.02583$ kg-m^2 is the moment of inertia of the whole mechanism, g is the gravitational constant, u is the torque generated by the actuator and is limited by $|u| \leq 0.57$ Nm. The position $\theta = 0$ corresponds to the downward equilibrium.

The architecture of the neural net used to represent H_d for this system is $(2, 48, 36, 1)$. The first number indicates the dimension of the input layer. The last number indicates the dimension of output layer. The dimensions of each hidden layer are shown in between.

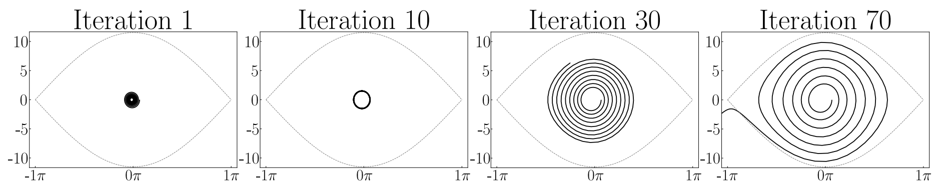

Fig. 1 Convergence to the homoclinic orbit (dashed line) of the learned controller for the physical pendulum. The horizontal and vertical axes are the pendulum angle and angular velocity, respectively.

The preferred orbit for calculation of $r_\perp$ in Eq. (3) is the homoclinic orbit characterized by

$$\dot{\theta}^\star(\theta) = \sqrt{\frac{2}{I}\left(E_0 - mgl(1-\cos\theta)\right)}, \tag{7}$$

where $E_0 = 2mgl$ is the potential energy of the pendulum at the unstable equilibrium. With this orbit, the transverse coordinate is $x_\perp = \min(|\dot{\theta} \pm \dot{\theta}^\star|^2)$. The goal set $\mathcal{S}$ used to compute the set distance loss r_{set} described is defined as

$$\mathcal{S} = \left\{(\theta, \dot{\theta}) : \left\| \begin{bmatrix}\theta \ \dot{\theta}\end{bmatrix}^\top - \begin{bmatrix}\pm\pi \ 0\end{bmatrix}^\top \right\| \leq 0.1 \right\}.$$

A demonstration of the training progress is shown in Figure 1. The controller learns to bring the trajectories toward the homoclinic orbit and swing up the pendulum. The learned controller is evaluated by simulating the pendulum for 15 seconds from a range of initial states. We perform 2,500 simulations with initial angles in the interval $[-0.9\pi, 0.9\pi]$ rad, and initial velocities in the interval $[-10, 10]$ rad/s. The data-driven controller swings up the system with a 100% success rate.

We further bolster the efficacy of our approach by implementing the learned controller on an experimental system. We start the pendulum at a random initial condition close the origin and record the angle and the commanded torque. Figure 2 shows the results of the experiment. These results demonstrate the robustness of the swing-up controller, as the system parameters in Eq. 6 are estimated. Further, the viscous friction model is only an approximation of the true friction in the bearings of the real system. Velocity measurements are also not directly available. They are estimated through the use of a digital low-pass filter.

3.2 Cart-pole

The cart-pole experiment is conducted using the Linear Servo Base Unit with Inverted Pendulum, manufactured by Quanser [13]. It consists of an unactuated pendulum mounted (via revolute joint) to a linear cart that is actuated. The position of the pendulum is denoted by the angle θ, with the upright position as $\theta = 0$. The position of the cart,

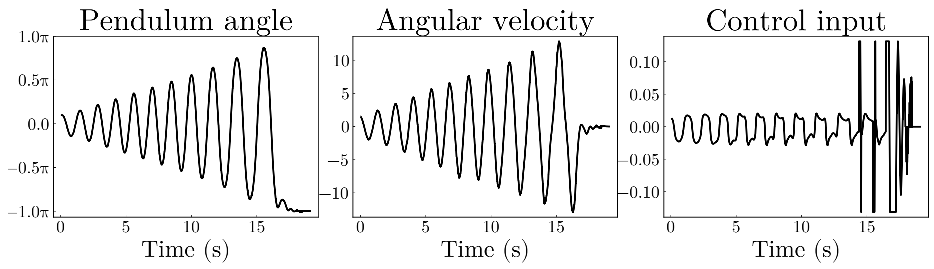

Fig. 2 Experimental results for the physical pendulum. The angle measurement contains a small amount of noise, and the velocity measurement is approximated. The learned controller is able to swing up despite these inaccuracies.

denoted by x, is constrained to $|x| < 0.4$ m. The equations of motion are

$$\begin{aligned} I_2\ddot{x} &= I_1\left(F_c - B_{eq}\dot{x} - ml\dot{\theta}^2 s_\theta\right) + ml\left(mglc_\theta s_\theta - B_p\dot{\theta}c_\theta\right), \\ I_2\ddot{\theta} &= (M+m)\left(mgls_\theta - B_p\dot{\theta}\right) + ml\left(F_c c_\theta - B_{eq}\dot{x}c_\theta - ml\dot{\theta}c_\theta s_\theta,\right), \end{aligned} \tag{8}$$

where s_θ and c_θ denote sine and cosine of θ, respectively; $I_1 = I + ml^2$; $I_2 = (M + m)I + ml^2\left(M + m\sin^2\theta\right)$; F_c is the actuator force acting on the cart; and g is gravity. The model parameters can be found on the manufacturer's website.

We use a neural network with the following dimensions: $(4, 64, 64, 1)$. Since the cart-pole system is symmetric, we apply the procedure described in Section 2.2 to impose symmetry. In particular, we process the input with $g(x) = x^2$ before feeding it to the neural network. The goal set S in the computation of the set distance loss is chosen as

$$S = \left\{\left(x, \dot{x}, \theta, \dot{\theta}\right) : \sqrt{(1-\cos\theta)^2 + \dot{\theta}^2 + \dot{x}^2} \le 0.125\right\}.$$

We evaluate a total of 2,500 simulations starting from a range of initial states with $\theta \in \pm[30, 150]$ degrees, $\dot{\theta} \in [-8, 8]$ rad/s, and zero in other dimensions. The swing-up controller achieves a 100% success rate for the cart-pole system. The contours of $H_d(x)$ and the controller $u(x)$ derived from it are shown in Figure 3. To demonstrate the advantages of our approach over with traditional energy-shaping control, we compare the accumulated quadratic cost to perform the swing-up on the cart-pole in Figure 4. The same cost matrices are used for both the learned controller and the energy-shaping controller. Our approach resulted in lower accumulated cost and successful swing-up from a wide variety of initial conditions.

The results from an experimental trial are shown in Figure 5. A video accompanying this submission also shows a recording of this specific trial. The learned controller performs the swing up without going over cart track limits. A standard LQR controller successfully catches the swing and stabilizes the upright position. These results show that the learned controller is robust against the inaccuracies in velocity approximation, system parameters, and friction model.

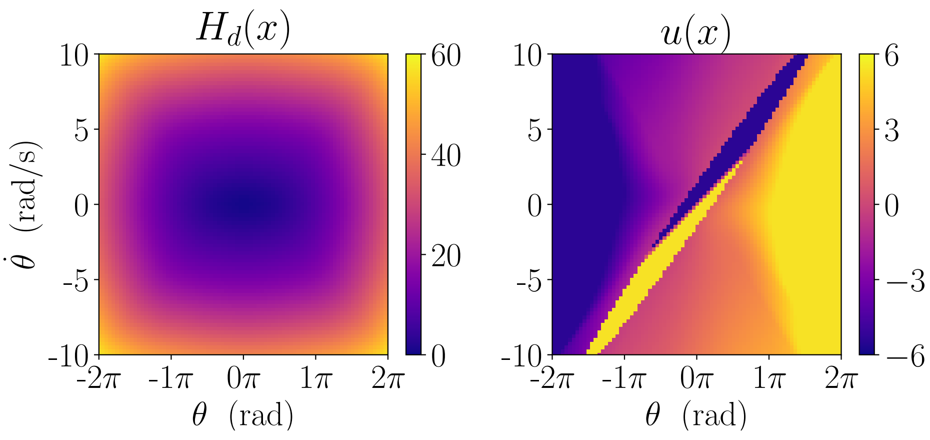

Fig. 3 (Left) The learned desired Hamiltonian $H_d(x)$, and (right) the resulting control policy $u(x)$ for the cart-pole system, projected onto the θ-$\dot{\theta}$ plane with $x = \dot{x} = 0$. The neural network used to learn H_d consists of approximately 4,500 parameters to be trained, which is relatively small in the world of machine learning.

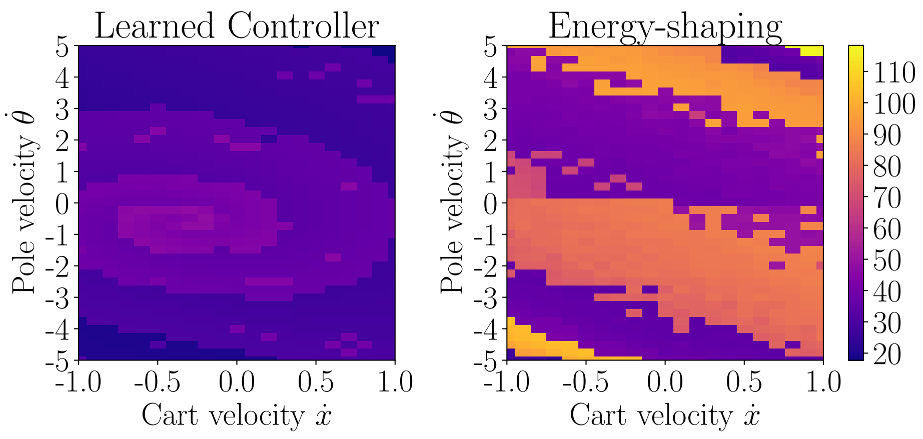

Fig. 4 Each grid represents the accumulated quadratic cost over an 8-second simulation from the corresponding initial cart and pole velocities. The initial cart position is $x = 0$ and the pole starts at the downward position. The learned controller performed the swing-up with lower cost than the traditional energy-shaping controller.

4 Main Experimental Insights

The successful experiments bolster the efficacy and robustness of our approach. In both the pendulum and the cart-pole experiments, the system parameters and the viscous friction model used during training are approximations of the true system models. The use of a digital low-pass filter to approximate the velocities further deviates the the system from the conditions used during training. The experimental results show that our approach is robust against these uncertainties.

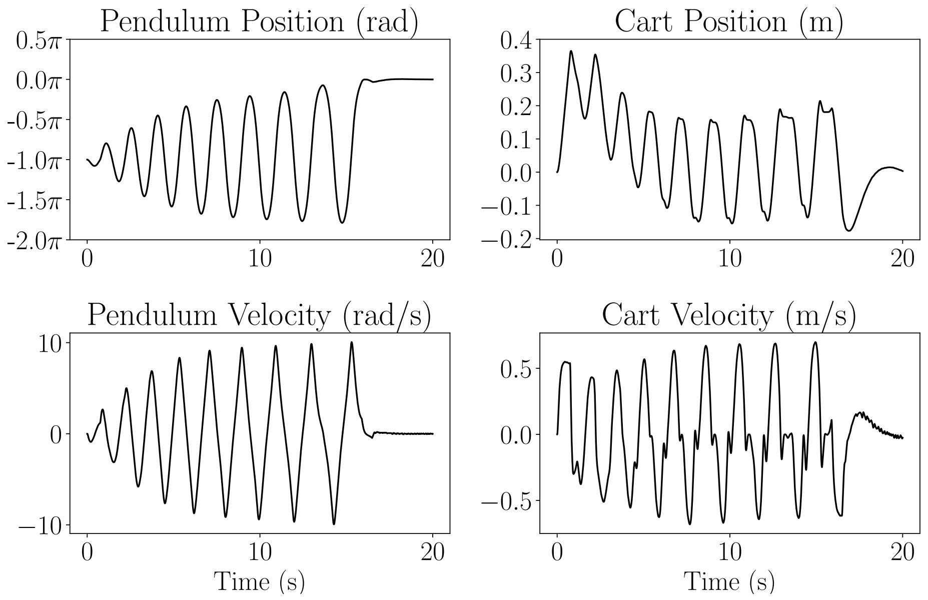

Fig. 5 Experimental results showing state evolution of the cart-pole. The learned controller is able to swing up the pendulum without violating the cart track limits.

Implementation of the traditional energy-shaping control requires careful tuning of the swing-up gain. On the cart-pole system, a gain that is too large resulted in the cart going over track limits, while one that is too small did not swing up the pole. Similar behavior was observed in the simple pendulum experiment. Our data-driven designs do not require gain tuning, simplifying the implementation of the controllers.

One challenge revealed during experiments is the computation of the gradient of H_d. This computation can be more costly than that of traditional swing-up controllers. In real-time applications where update frequency of a kilohertz or higher, the computation power requirement is a valid concern. This issue can be minimized by avoiding large neural network structures or by the use of parallel computing.

We show, through both simulations and experiments, that our data-driven approach can achieve performance levels similar to or exceeding that of traditional swing-up control designs. The incorporation of physical knowledge about the system combined with techniques from dynamics and control theory, such as transverse dynamics and energy-shaping control, lead to relatively quick training sessions. Our approach also takes into account the symmetry of the system's dynamics, resulting in a more consistent closed-loop behavior across the state space. We plan to extend this framework to accommodate systems with contacts in future work.

References

1. Åström, K.J., Furuta, K.: Swinging up a pendulum by energy control. Automatica **36**(2), 287–295 (2000)
2. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations (2018)
3. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
4. Conway, B.: Practical methods for optimal control using nonlinear programming (2002)
5. Deisenroth, M., Rasmussen, C.E.: Pilco: A model-based and data-efficient approach to policy search. In: Proceedings of the 28th International Conference on machine learning (ICML-11), pp. 465–472 (2011)
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
7. Manchester, I.R.: Transverse dynamics and regions of stability for nonlinear hybrid limit cycles. arXiv preprint arXiv:1010.2241 (2010)
8. Mason, P., Broucke, M., Piccoli, B.: Time optimal swing-up of the planar pendulum. IEEE Transactions on Automatic Control **53**(8), 1876–1886 (2008)
9. Mori, S., Nishihara, H., Furuta, K.: Control of unstable mechanical system control of pendulum. International Journal of Control **23**(5), 673–692 (1976)
10. Ortega, R., Spong, M.W., Gómez-Estern, F., Blankenstein, G.: Stabilization of a class of underactuated mechanical systems via interconnection and damping assignment. IEEE transactions on automatic control **47**(8), 1218–1233 (2002)
11. Park, M.S., Chwa, D.: Swing-up and stabilization control of inverted-pendulum systems via coupled sliding-mode control method. IEEE transactions on industrial electronics **56**(9), 3541–3555 (2009)
12. Pontryagin, L.S., Mishchenko, E., Boltyanskii, V., Gamkrelidze, R.: The mathematical theory of optimal processes (1962)
13. Quanser: Linear Servo Base Unit with Inverted Pendulum (2013). Rev. 1.0
14. Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A.: Universal differential equations for scientific machine learning (2020)
15. van der Schaft, A.J., Van Der Schaft, A.: L2-gain and passivity techniques in nonlinear control, vol. 2. Springer (2000)
16. Shiriaev, A., Perram, J.W., Canudas-de Wit, C.: Constructive tool for orbital stabilization of underactuated nonlinear systems: Virtual constraints approach. IEEE Transactions on Automatic Control **50**(8), 1164–1176 (2005)
17. Shiriaev, A.S., Freidovich, L.B.: Transverse linearization for impulsive mechanical systems with one passive link. IEEE Transactions on Automatic Control **54**(12), 2882–2888 (2009)
18. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
19. Zhong, Y.D., Dey, B., Chakraborty, A.: Symplectic ode-net: Learning hamiltonian dynamics with control. arXiv preprint arXiv:1909.12077 (2019)