2021

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

Wankun Sirichotiyakul
*Boise State University*

Aykut C. Satici
*Boise State University*

## Publication Information

Sirichotiyakul, Wankun and Satici, Aykut C. (2021). "Data-Driven Design of Energy-Shaping Controllers for Swing-Up Control of Underactuated Robots". In B. Siciliano, C. Laschi, and O. Khatib (Eds.), *ISER 2020: Experimental Robotics* (Springer Proceedings in Advanced Robotics series, Volume 19, pp. 323-333). Springer. https://doi.org/10.1007/978-3-030-71151-1_29

# 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.**

## I. Introduction

In this paper, we discuss the data-driven design of controllers for a class of underactuated robotic mechanisms. The quintessential control problem that we tackle is the swing-up control of pendulum systems, such as the simple pendulum and the inertia wheel pendulum.

The swing-up control problem is to move the pendula from its stable downward position to its unstable inverted position and balance it about the vertical. The motions considered in this problem are large and therefore the swing-up problem is highly nonlinear and challenging.

This problem has been the subject of innumerable research papers until today. Control design approaches ranging from bang-bang control, sliding mode control, energy-based approaches, and many others have been extensively studied [1]–[4]. Passivity-based control [5], [6], which is a generalization of energy-shaping approach can also be applied to swing up the pendulum. 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.

The surge in popularity of machine learning has brought with it many data-driven solutions to the problem of the control design. Data-driven approaches are 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 [7], which seeks a direct mapping from the system states to the control inputs. These approaches been used in control tasks such as robot locomotion and manipulation [8], [9] and control of underactuated systems [10]. Most of data-driven approaches in the domain of control design 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 [11], the dynamics 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. The neural ordinary differential equation (ODE) [12], is a recent 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 [13].

Our aim in this paper is to combine the many of the clever techniques researchers have used to successfully swing-up pendulum systems with data-driven techniques to automatically come up with clever control laws. To this end, we develop a learning framework which incorporates the underlying physical model of the system. Our approach uses the recent development of an extension to neural ODE [14] that enables the direct incorporation of a neural network into the differential equation governing the system. We combine this neural-network-embedded ODE with passivity-based control techniques to learn an energy-like function that imposes desirable characteristics onto the closed-loop system. To train the neural net, we construct the loss function using transverse coordinates, which quantify the distance between system's trajectories and some desired orbit.

In this work, we provide novel results in the design of data-driven control laws for a class of underactuated mechanical systems. The contributions of this paper are summarized below:

- Express the controller through a neural network and incorporate it into the ODE governing the evolution of the system,
- Design of loss functions that rely on a transverse coordinate system, which speeds up training,
- Provide simulation and experimental support for the framework.

## II. Background

In this section, we summarize the technical background that is used in the remainder of the paper. Further details on the cursory exposition here, may be found in the references [5], [6], [15], [16].

### A. Transverse Coordinates

In this work, we will be concerned with learning structured controllers, that renders some chosen orbits asypmtotically stable. On the other hand, measuring the distance to a feasible trajectory of a dynamical system is a computationally difficult problem. This computational burden may be alleviated by the use of transverse coordinates. [15], [16].

This method reparametrizes the states of the dynamical system by assigning one state, $\tau$, to be a parameter that determines where along the specific trajectory the state of the system is The remaining $2n-1$ states, $x_\perp$, are chosen transversal to the trajectory and are called *transverse coordinates*. These coordinates may be thought of as a parametrization of a local moving family of Poincaré sections along the trajectory.

This family of Poincaré sections can be described as a family of hyperplanes defined as follows

$$\mathscr{S}(\tau) \;= \left\{y \in \mathbb{R}^{2n} : f(x^\star(\tau))^\top\left(y - x^\star(\tau)\right) = 0\right\},$$

where and $x^\star(\tau)$ is a specified orbit with $\dot{x}^\star(t) \neq 0$, $\forall t \in [0,\infty)$, and $f(x^\star(\tau))$ is the system dynamics along this specified orbit. We construct a coordinate system on $\mathscr{S}(\tau)$ by choosing a basis on this subspace. This procedure may be found in detail in [16]. A projection operator $\Pi(\tau)$ onto this space is then defined, which defines a change of coordinates $x \mapsto (x_\perp, \tau)$, where $\tau$ represents which of the transversal surfaces $\mathscr{S}(\tau)$ the current state $x$ inhabits, and the vector $x_\perp$ is the "transversal" state representing the location of $x$ within the hyperplane $\mathscr{S}(\tau)$, with $x_\perp = 0$ implying that $x = x^\star(\tau)$. The transverse coordinate under this construction is determined by

$$x_\perp \;= \Pi\left(y - x^\star(\tau)\right), \quad \text{for any } \; y \;\in \mathscr{S}(\tau). \tag{1}$$

### B. Energy-Shaping Control

The form of the controllers we design in this paper using data-driven methods is based on passivity-based/energy-shaping control. In this subsection , we will borrow from [5], [6] to describe the general outline of this method.

We 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 $x = (q,p)$. It is known that for hyperregular Lagrangians, the Hamiltonian and Lagrangian formulation of dynamics are equivalent [17]. We take the Hamiltonian point of view in the background section; however, we go back and forth the two formulations through the Legendre transformation, $\dot{q} \mapsto p = \frac{\partial H}{\partial \dot{q}} = M\dot{q}$, where $M$ is the symmetric, positive-definite mass matrix and $H$ is called the Hamiltonian of the robot, expressed as

$$H(q,p) = \frac{1}{2}p^\top M^{-1}(q)p + V(q),$$

where $V(q)$ represents the potential energy. The system's equations of motion can then be expressed as

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

The main idea of energy-shaping control, which is closely tied to passivity-based control [6], is to use the control input $u \in \mathscr{U} \subseteq \mathbb{R}^m$ to impose a desired (closed-loop) energy-like function: $H_d : \mathbb{R}^{2n} \to \mathbb{R}$. This energy-like function was chosen to be a quadratic function of the system momenta in [5] as follows

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

however, in this work we do not constrain ourselves to desired Hamiltonians of this particular form.

The controllers we choose in this work is comprised of an energy-shaping term and a damping injection term, i.e., $u(x) = u_{es}(q,\dot{q}) + u_{di}(q,\dot{q})$, where

$$\begin{aligned} u_{es}(q,\dot{q}) &= -(G^\top G)^{-1}G^\top\frac{\partial H_d}{\partial q},\\ u_{di}(q,\dot{q}) &= -K_d G^\top\frac{\partial H_d}{\partial p}. \end{aligned} \tag{2}$$

We invoke methods from machine learning, elaborated in the next section, to come up with the energy-like function $H_d$ automatically. We then take this function and use it in conjunction with equations (2) to determine the controller that performs the swing-up of various underactuated robotic systems. When we represent $H_d$ by a neural network, we will sometimes explicitly denote its dependence on various parameters $\theta \in \mathbb{R}^p$ by $H_d(x;\theta)$.

## III. Methods

In this section, we describe how to learn the function $H_d(x;\theta)$, that is used to derive the controller that performs the swing-up maneuver.

### A. Main Learning Problem

In this subsection, we formulate the control design for swinging-up robotic mechanisms as an optimization problem of a neural network, which estimates a Lyapunov function that is used to derive the controller.

We assume that the equations that govern the motion of the robotic mechanism takes the nonlinear, control-affine form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u(x;\theta),\\ x(t_0) &= x_0 \end{aligned} \tag{3}$$

where $x \in \mathbb{R}^{2n}$ is the state vector, $u \in \mathbb{R}^m$ is the vector of control inputs, determined by equations (2), and $x_0 \in \mathbb{R}^{2n}$ is the initial condition. Note that $u$ is a function of the parameters of the neural network as it is found by differentiating $H_d$. We assume that the system dynamics $f, g$ are known and are sufficiently smooth vector fields. As the control input $u$ is determined from equation (2), we can integrate the initial value problem (IVP) (3) to obtain a trajectory $\gamma : t \mapsto \phi(t; x_0, \theta)$. We refer to each $\gamma$ as a prediction from the initial condition $x_0$ with the current control law $u(x;\theta)$.

Our goal is to learn the parameters for $H_d$, which will be used to form the controller, so that a performance functional $J : \mathbb{R}^p \to \mathbb{R}$ is minimized. In other words, we formulate the following optimization problem, to be solved over the neural network parameters, $\theta \in \mathbb{R}^p$.

$$\begin{aligned} \underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad & J(\theta) = \int_{t_0}^{\tau} r(\phi, u(\phi;\theta))\, dt, \\ \text{subject to} \quad & \dot{x} = f(x) + g(x)u(x;\theta), \\ & x(t_0) = x_0. \end{aligned} \tag{4}$$

where $\phi = \phi(t; x_0, \theta)$ is the flow of the equations of motion, the time horizon $\tau \in (0,\infty)$ is a hyperparameter and $r : \mathbb{R}^{2n} \times \mathbb{R}^m \to \mathbb{R}$ is a running cost function, whose design we elaborate in the following subsection.

## B. Loss Function

The running cost function $r(x,u)$ that yields the loss function $J(\theta)$ consists of four main parts: 1) distance, $r_\perp$, to a preferred orbit, $\gamma^\star$, 2) set distance, $r_{\text{set}}$, between the current prediction and the goal set, 3) Double-hinge loss, $r_{\text{hinge}}$, to penalize multiple swings of the first link, and 4) the regularization term, $r_{\text{reg}}$, to avoid overtraining the neural network parameters. Since the regularization term is so well-known in machine learning literature, it will not be further delineated.

The running loss is found by suming of these components:

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

where each term depends on $\theta$ through the dependence of the whole system trajectory on $\theta$.

*1) Distance to a preferred orbit:* The first term that makes up the running loss is the distance to a preferred orbit, $\gamma^\star$. In Section IV, 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 this case, the preferred orbit may be computed using one of several methods that are available in the literature, such as, trajectory optimization [18], virtual holonomic constraints [19], 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 Section II-A. These coordinates allow us to compute the distance from a prediction $\gamma$ to $\gamma^\star$ efficiently.

This computation is performed as follows. We randomly sample $N$ points from the current prediction $\gamma$. The distance of each of these $N$ points to $\gamma^\star$ is then computed by the norm of its transverse coordinate $x_\perp$.

Notice that the prediction converges to $\gamma^\star$ if and only if $x_\perp \to 0$ as $t \to \infty$. In order to encourage this behavior, it makes sense to set the first term in our running cost as a quadratic loss on $x_\perp$. Since we also want to use reasonable control effort while having $\gamma$ converge to $\gamma^\star$, we append a cost for control effort usage:

$$r_\perp(x,u) = \frac{1}{2N} x_\perp^\top Q x_\perp + \frac{1}{2} u^\top R u \tag{5}$$

*2) Set Distance Loss:* The ultimage 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, $r_{\text{set}}$ is constructed by definining a convex open neighborhood $S$ of the upward equilibrium point and computing the set distance of $\gamma(t)$ to $S$

$$\begin{aligned} r_{\text{set}}(x) &= \operatorname{dist}(\gamma, S) \\ &= \inf\{\|x-y\| : x \in \gamma(t),\ t \in (t_0,\tau),\ y \in S\} \end{aligned}$$

For instance, we can choose the set $S$ to be a ball around the upward equilibrium point of radius $r > 0$ in the standard norm topology. In this case, $r$ becomes a hyperparameter to be selected by the user before the training. With this choice, if any point along the prediction $\gamma$ gets closer than $r$ to the upward equilibrium point, no additional loss is incurred.

*3) Double-hinge loss:* We want penalize those prediction trajectories $\gamma$ which overshoot the upward equilibrium point and require the first link to swing multiple times before reaching the set $S$ using a double-hinge loss term, $r_{\text{hinge}}$.

$$r_{\text{hinge}}(x) = \begin{cases} 0, & |x| \le 2\pi, \\ |x| - 2\pi, & \text{otherwise.} \end{cases} \tag{6}$$

where $x$ is the angle of the first link.

## C. Gradient Computation through ODEs

We want to find a solution to the optimization problem in (4) using gradient-based search, as any other optimization method would be excruciatingly slow. In this paper, we make use of ADAM [20] as the underlying gradient-based descent algorithm for updating the parameters.

However, any gradient-based optimization algorithm clearly requires that the gradient of the loss function $J(\theta)$ be available to carry out parameter update. Since in formulation (4), the loss depends on the parameters through the solution $\phi(t;x_0,\theta)$ of an ODE, it is not immediately clear how the gradients $\nabla_\theta J(\theta)$ will be computed.

Employing the chain rule of differentiation yields that the gradient we are looking to compute depends on the gradient $\frac{\partial\phi}{\partial\theta}$ of the solution with respect to the parameters and the gradient $\frac{\partial J}{\partial x}$ of the loss function with respect to the system state.

In the recent literature [12], 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 may be computed through the expression

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

This approach is based on Pontryagin's maximum principle [21] 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 on a solver implemented entirely in a language with pervasive AD. In [14], a number of existing AD schemes and adjoint methods are generalized to enable gradient computation of solutions to ODEs with neural-net architectures embedded in them. We use this very recent framework to compute the gradients of the loss function $J(\theta)$ and use it in conjunction with ADAM to solve our optimization problem (4).

## D. Sampling the State Space

The initial conditions from which the predictions $\gamma$ are generated must be sampled finely enough for sufficient training. Rather than sampling randomly across the state space, we follow the DAGGER approach proposed in [22]. In summary, DAGGER simulates the system forward using the learned policy, which is initially poorly-trained. We then select $M$ points along the generated trajectory to use as initial conditions for creating new prediction-data pairs. As learning progresses, it iteratively collects new training samples from the regions of state-space visited by the learned policy.

## E. Training the Neural Network

The parameters $\theta$ are trained using ADAM with the default settings. We summarize the training process in Algorithm 1. The parameters $N, K, M, P$ are hyperparameters to be chosen by the user. $N$ is the number of sampled states along $\gamma$ used to compute the loss in eq. (5). $K$ is the number of trainings repeated on a single prediction $\gamma$. $M$ is the number of training samples randomly selected from $\gamma_0$, which is the initial trajectory generated according to DAGGER. $P$ is the number of repetitions of the sample collection process. This corresponds to the total number of $\gamma_0$ to be generated and sampled from. In total, this algorithm creates $MP$ training samples.

**Algorithm 1:** Training Process

```
Input: Initial parameters $\theta_0$
1  Initialize $\mathcal{D} \leftarrow \emptyset$, a container for storing initial states $x(t_0)$
2  for $i = 1, \dots, P$ do
3      $\gamma_0 \leftarrow$ integrate eq. (3) w/ current $\theta$ and random $x(t_0)$
4      $\gamma_0^{(m)} \leftarrow$ pick $M$ states randomly sampled from $\gamma_0$
5      $\mathcal{D} \leftarrow \mathcal{D} \cup \gamma_0^{(m)}$
6      for each $d \in \mathcal{D}$ do
7          for $j = 1, \dots, K$ do
8              $\gamma \leftarrow$ integrate eq. (3) w/ current $\theta$ and
                 $x(t_0) = d$
9              Compute $J(\gamma; \theta)$ and $\partial J/\partial\theta$ (for $N$ sample
                 states along $\gamma$)
10             $\theta \leftarrow$ update $\theta$ according to ADAM
Output: Solution of optimization problem (4)
```

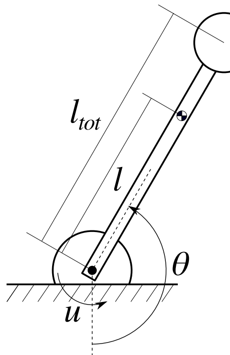

Fig. 1: Schematic of the pendulum. The actuated revolute joint is torque-limited and is modeled with viscous friction.

TABLE I: Physical parameters for the pendulum

| Parameter | Symbol | Value | Units |
|---|---|---|---|
| Combined mass | $m$ | 0.377 | kg |
| Length to CoM | $l$ | 0.232 | m |
| Total length | $l_{\text{tot}}$ | 0.420 | m |
| Moment of inertia | $I$ | 0.02583 | kg-m$^2$ |
| Friction coefficient | $b$ | 0.005 | Nm-s |
| Max torque | $u_{\max}$ | 0.1313 | Nm |
| Gear ratio | $\eta$ | 13/3 | - |

## IV. Results

We apply our approach to two robots commonly used as benchmarks: the physical pendulum and the inertia wheel pendulum. We train two neural networks to find swing-up controllers for each system. Results from simulation studies are provided for both systems. Additionally, results from physical experiments are provided for the physical pendulum.

## A. The Physical Pendulum

The pendulum consists of a bob and an arm of length $l_{\text{tot}}$. Their combined mass is denoted by $m$. The arm is connected to ground through a revolute joint, whose position is denoted by the angle $\theta$ between the centerline of the arm and the downward vertical line. The center of mass (CoM) is located at a distance $l$ from the center of the revolute joint. The revolute joint is actuated and is modeled with viscous friction with the coefficient $b$. Figure 1 shows a schematic of the pendulum.

The dynamics of the pendulum is given as

$$I\ddot{\theta} = -mgl\sin(\theta) - b\dot{\theta} + \eta u \tag{7}$$

where $I$ is the moment of inertia of the whole mechanism, $g$ is the gravitational constant, $u$ is the torque generated by the actuator, and $\eta$ is the gear ratio of the actuator. The torque $u$ is limited by $|u| \leq u_{\max}$. The physical parameters used in the experiments are summarized in Table I.

It is important to note that the maximum torque $u_{\max}$ available in both our simulation and experimental setup is such that the upward equilibrium point cannot be reached by just rotating in one direction as the gravitational force overcomes the motor torque eventually. As a result, the

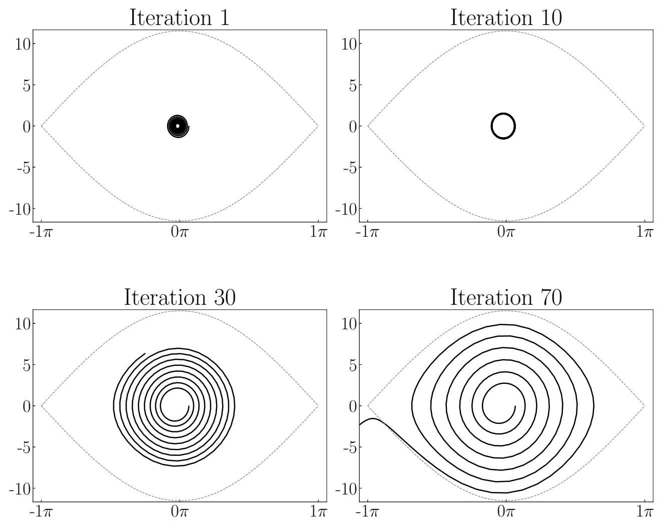

Fig. 2: Convergence to the homoclinic orbit (dashed line) of the learned controller for the physical pendulum. The horizontal and vertical axes are the pendulum angle and angular velocity, respectively. The overall training process only needs the homoclinic orbit as the training data, demonstrating the data-efficiency of our approach.

controller has to be clever enough to successfully overcome gravitational forcing with a combination of built-up momentum and available motor torque.

We use a simple fully-connected neural network with (ELU) activations [23] to learn the swing-up control. The neural net has two hidden layers with 48 nodes in the first and 36 nodes in the second. There are a total of 1,945 parameters to train with this architecture. We train this neural net to find $H_d$ in eq. (II-B). The training data is generated using the homoclinic orbit, which is characterized by

$$\dot{\theta}^{\star}(\theta) = \sqrt{\frac{2}{I}\left(E_0 - mgl(1-\cos\theta)\right)}, \tag{8}$$

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 $S$ used to compute the set distance loss $r_{\text{set}}$ described in Section III-B is defined as

$$S = \left\{ x \in \mathbb{R}^2 : \|x - x_{\star}\| \leq 0.1 \right\},$$

where $x_{\star} = \begin{bmatrix} k\pi & 0 \end{bmatrix}^{\top}$, with $k$ an odd integer, is the upward equilibrium point. We train the neural network with the hyperparameters $P, M, K, N$ in Algorithm 1 chosen as $25, 8, 12, 64$, respectively.

With this set of parameters and loss function, the training required approximately ten hours on a single CPU. The training progress for a single initial condition is demonstrated in Figure 2. We use the programming language Julia with the packages `Flux.jl` [24], `DiffEqFlux.jl`, and `DifferentialEquations.jl` to perform the training. Automatic differentiation is handled by the packages

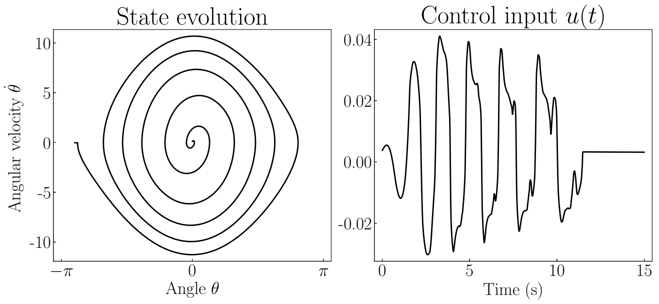

Fig. 3: Simulation results for the pendulum. The learned controller takes the pendulum to the homoclinic orbit. When the state comes close to the unstable equilibrium, the linear controller takes over.

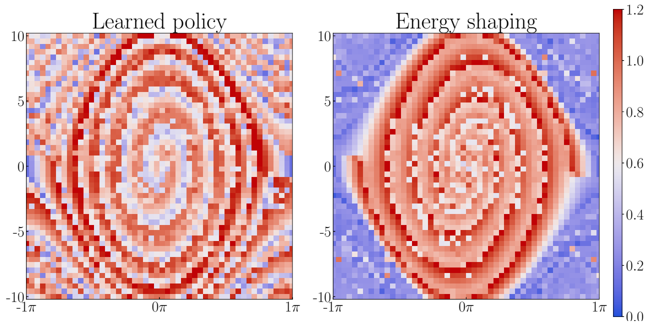

Fig. 4: Accumulated control input demanded by the learned policy (left) and the energy shaping controller (right) to swing up the pendulum. The horizontal and vertical axes are the pendulum angle and angular velocity, respectively. The demanded control effort by learned policy mimics that of the energy-shaping control.

`Tracker.jl` and `Zygote.jl`. The source code for our learning framework is available at `https://bitbucket.org/bsurobotics/ml_based_esc_iros_2020`.

The learned controller is evaluated by simulating the pendulum for 15 seconds from a range of initial states. Each initial angle is in the interval $[-0.9\pi, 0.9\pi]$ rad, and each initial velocity is in the interval $[-10, 10]$ rad/s. Fifty samples of each initial position and velocity are collected, for a total of 2,500 simulations. We employ a switching scheme where the learned controller is used for swinging up, and LQR is used near the upright position. If the linear controller catches the swing and stabilizes the upward equilibrium point $x_{\star}$, then the simulation is considered successful. Performance of the swing-up controller is evaluated by the number of successful simulations.

In only 2 out of 2,500 simulations, the learned controller failed bring the pendulum close enough to the unstable equilibrium for the linear controller to take over. This result gives us a success rate of 99.92% for the learned policy. A comparison of the control effort needed to swing up by the learned controller and traditional energy-shaping control is given in Figure 4. We see that the demanded effort by the learned policy resembles that of the traditional energy shaping controller.

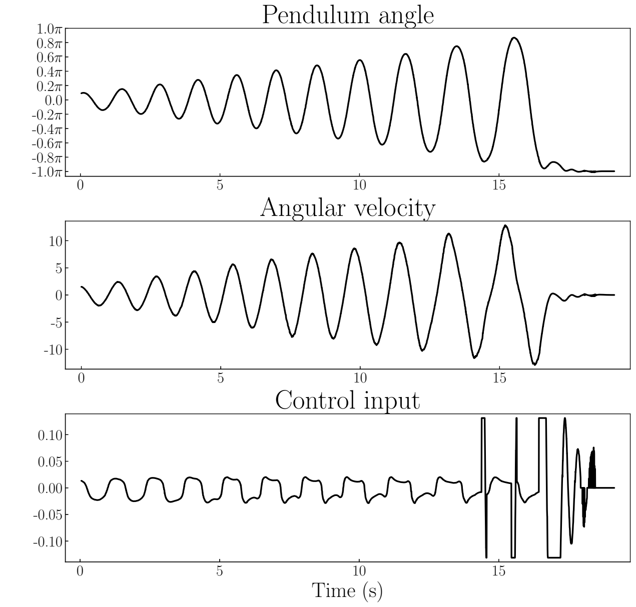

Fig. 5: Experimental results for the physical pendulum. The angle measurement contains a small amount of noise, and the velocity measurement is approximated. The learned controller is able to swing up despite these inaccuracies.

We further bolster the efficacy of our approach by implementing the learned controller on an experimental system. The parameters for our experimental setup are the same ones used in simulation and are listed in Table I. The viscous friction model is an approximation of the true friction in the bearings of our system. We start the pendulum at a random initial condition close the origin and record the angle and the commanded torque. Velocity measurements are not directly available in the experimental setup. They are estimated through the use of a digital low-pass filter. Figure 5 shows the results of the experiment. Note that the learned controller is robust against the inaccuracies in velocity approximation, system parameters, and friction model.

## B. The Inertia Wheel Pendulum

The inertia wheel pendulum (IWP) is a simple pendulum with an actuated wheel instead of a static bob. The wheel has mass $m$, which is connected to a massless rod of length $l$. The position $\theta_2$ of the wheel is measured with respect to the vertical line through the center of the wheel. The rod is connected to ground by an unactuated revolute joint, whose position is denoted by the angle $\theta_1$ measured with respect to the downward vertical position. A schematic of the inertia wheel pendulum is shown in Figure 7.

The dynamics of the inertia wheel pendulum is

$$\begin{aligned} I_1\ddot{\theta}_1 &= -mgl\sin(\theta_1) - u, \\ I_2\ddot{\theta}_2 &= u \end{aligned} \tag{9}$$

where $I_1$ is the moment of inertia of the pendulum, $I_2$ is the moment of inertia of the rotating wheel, $g$ is the gravitational constant, and $u$ is the torque generated by the actuator. The torque is limited by $|u| \leq u_{\text{max}}$. In the simulation studies,

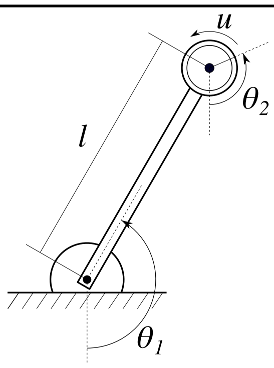

Fig. 6: Schematic of the inertia wheel pendulum. The rotating wheel $\theta_2$ is actuated.

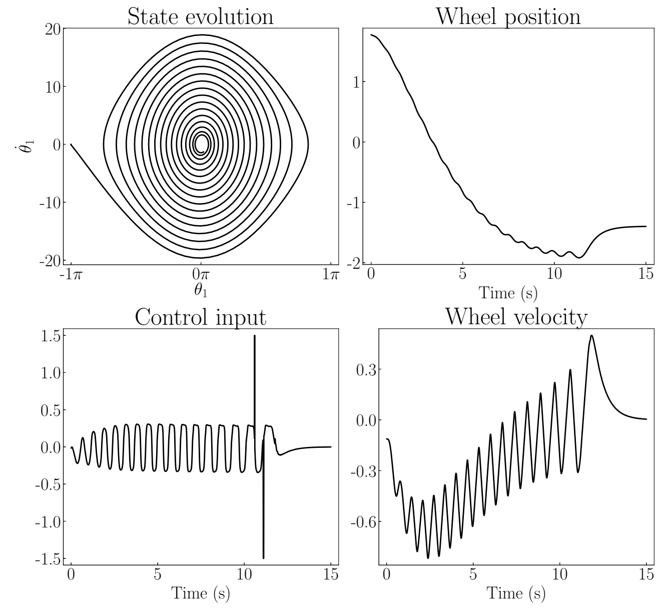

Fig. 7: Simulation results for the IWP starting from a random initial state. (Upper left): The state evolution projected onto the $\theta_1$-$\dot{\theta}_1$ plane. (Lower left): The control input using a switching scheme. The large spikes are commanded by the linear controller. (Right): The wheel position and velocity.

the following values are used for system parameters: $I_1 = 0.1$, $I_2 = 0.2$, $mgl = 10$, and $u_{\text{max}} = 1.5$.

For this experiment, we use a neural network with the following dimensions in each layer: $(4, 64, 64, 1)$. There are a total of 4,545 parameters to train with this architecture. We perform the training with the hyperparameters $P, M, K, N$ chosen as $30, 8, 4, 64$, respectively. As the pendulum subsystem of the IWP is passive, the homoclinic orbit of the IWP is the same as eq. (8), with $\theta$ replaced by $\theta_1$ and the inertia parameter is replaced with $I = I_1 + I_2$. We select the goal set $S$ in the computation of the set distance loss, $r_{\text{set}}$, to be

$$S = \left\{ \left(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2\right) : \sqrt{(\theta_1 - \pi)^2 + \dot{\theta}_1^2 + \dot{\theta}_2^2} \leq 0.1 \right\}.$$

The learned controller is evaluated by simulating the inertia wheel pendulum for 20 seconds from a range of initial states with $\theta_1 \in [-0.9\pi, 0.9\pi]$ rad, and $\dot{\theta}_1 \in [-10, 10]$ rad/s. The initial wheel position and velocity are zero. Fifty samples of each initial $(\theta_1, \dot{\theta}_1)$ are collected, for a total of 2,500 simulations. We evaluate the performance of the trained controller using the same method as the pendulum, i.e. catching the swing with LQR controller. There are 2,495 successful simulations, resulting in a success rate of $99.80\%$.

## V. Conclusion

We demonstrate a novel approach to the swing-up control of underactuated robots by embedding a neural network into the physical model through the control function and using techniques from dynamics and control theory, such as transverse dynamics and energy-shaping controllers. We encode a term in the loss function for the neural network, which learns and energy-like function for the closed-loop dynamics, using the transverse coordinates from a desired orbit. This term encourages the the controller, derived from the energy-like function, to pull the trajectories towards the orbit. Additional terms in the loss function that encourage convergence to a desired equilibrium help overcome getting stuck in local optimal solutions. We demonstrate the success of the proposed framework in two benchmark problems in control both in simulation and experiment.

## References

[1] S. Mori, H. Nishihara, and K. Furuta, “Control of unstable mechanical system control of pendulum,” *International Journal of Control*, vol. 23, no. 5, pp. 673–692, 1976.

[2] M.-S. Park and D. Chwa, “Swing-up and stabilization control of inverted-pendulum systems via coupled sliding-mode control method,” *IEEE transactions on industrial electronics*, vol. 56, no. 9, pp. 3541–3555, 2009.

[3] P. Mason, M. Broucke, and B. Piccoli, “Time optimal swing-up of the planar pendulum,” *IEEE Transactions on Automatic Control*, vol. 53, no. 8, pp. 1876–1886, 2008.

[4] K. J. Åström and K. Furuta, “Swinging up a pendulum by energy control,” *Automatica*, vol. 36, no. 2, pp. 287–295, 2000.

[5] R. Ortega, M. W. Spong, F. Gómez-Estern, and G. Blankenstein, “Stabilization of a class of underactuated mechanical systems via interconnection and damping assignment,” *IEEE transactions on automatic control*, vol. 47, no. 8, pp. 1218–1233, 2002.

[6] A. J. van der Schaft and A. Van Der Schaft, *L2-gain and passivity techniques in nonlinear control*. Springer, 2000, vol. 2.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[8] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, *et al.*, “Emergence of locomotion behaviours in rich environments,” *arXiv preprint arXiv:1707.02286*, 2017.

[9] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[11] M. Deisenroth and C. E. Rasmussen, “Pilco: A model-based and data-efficient approach to policy search,” in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[12] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, “Neural ordinary differential equations,” 2018.

[13] Y. D. Zhong, B. Dey, and A. Chakraborty, “Symplectic ode-net: Learning hamiltonian dynamics with control,” *arXiv preprint arXiv:1909.12077*, 2019.

[14] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, “Universal differential equations for scientific machine learning,” 2020.

[15] A. S. Shiriaev and L. B. Freidovich, “Transverse linearization for impulsive mechanical systems with one passive link,” *IEEE Transactions on Automatic Control*, vol. 54, no. 12, pp. 2882–2888, 2009.

[16] I. R. Manchester, “Transverse dynamics and regions of stability for nonlinear hybrid limit cycles,” *arXiv preprint arXiv:1010.2241*, 2010.

[17] F. Bullo and A. Lewis, *Geometric Control of Mechanical Systems: Modeling, Analysis, and Design for Simple Mechanical Control Systems*, ser. Texts in Applied Mathematics. Springer New York, 2019.

[18] B. Conway, “Practical methods for optimal control using nonlinear programming,” 2002.

[19] A. Shiriaev, J. W. Perram, and C. Canudas-de Wit, “Constructive tool for orbital stabilization of underactuated nonlinear systems: Virtual constraints approach,” *IEEE Transactions on Automatic Control*, vol. 50, no. 8, pp. 1164–1176, 2005.

[20] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[21] L. S. Pontryagin, E. Mishchenko, V. Boltyanskii, and R. Gamkrelidze, “The mathematical theory of optimal processes,” 1962.

[22] S. Ross, G. J. Gordon, and J. A. Bagnell, “No-regret reductions for imitation learning and structured prediction,” *CoRR*, vol. abs/1011.0686, 2010. [Online]. Available: http://arxiv.org/abs/1011.0686

[23] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.

[24] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, “Fashionable modelling with flux,” *CoRR*, vol. abs/1811.01457, 2018. [Online]. Available: http://arxiv.org/abs/1811.01457