DATA-DRIVEN PASSIVITY-BASED CONTROL OF UNDERACTUATED ROBOTIC SYSTEMS

by

Wankun Sirichotiyakul

A dissertation

submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Boise State University

August 2022

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the dissertation submitted by

Wankun Sirichotiyakul

Dissertation Title: Data-Driven Passivity-Based Control of Underactuated Robotic Systems

Date of Final Oral Examination: 17 May 2022

The following individuals read and discussed the dissertation submitted by student Wankun Sirichotiyakul, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Aykut Satici, Ph.D.	Chair, Supervisory Committee
Hao Chen, Ph.D.	Member, Supervisory Committee
John Chiasson, Ph.D.	Member, Supervisory Committee
Alireza Mohammadi, Ph.D.	Member, Supervisory Committee

The final reading approval of the dissertation was granted by Aykut Satici, Ph.D., Chair of the Supervisory Committee. The dissertation was approved by the Graduate College.

ABSTRACT

Classical control strategies for robotic systems are based on the idea that feedback control can be used to override the natural dynamics of the machines. Passivity-based control (PBC) is a branch of nonlinear control theory that follows a similar approach, where the natural dynamics is modified based on the overall energy of the system. This method involves transforming a nonlinear control system, through a suitable control input, into another fictitious system that has desirable stability characteristics. The majority of PBC techniques require the discovery of a reasonable *storage function*, which acts as a Lyapunov function candidate that can be used to certify stability.

There are several challenges in the design of a suitable storage function, including: 1) what a reasonable choice for the function is for a given control system, and 2) the control synthesis requires a closed-form solution to a set of nonlinear partial differential equations. The latter is in general difficult to overcome, especially for systems with high degrees of freedom, limiting the applicability of PBC techniques.

A machine learning framework that automatically determines the storage function for underactuated robotic systems is introduced in this dissertation. This framework combines the expressive power of neural networks with the systematic methods of the PBC paradigm, bridging the gap between controllers derived from learning algorithms and nonlinear control theory. A series of experiments demonstrates the efficacy and applicability of this framework for a family of underactuated robots.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Control problems for underactuated robotic systems have been addressed by many researchers, using various design strategies ranging from bang-bang control, energy-based approaches, and many others [1, 2, 3]. Passivity-based control [4, 5], which may be viewed as a generalization of energy-shaping, has also been proven effective for designing controllers for nonlinear systems. The Interconnection and Damping Assignment (IDAPBC) is one of the more prominent techniques in this family, thanks to its applicability to a large class of physical systems [6, 7, 8, 9].

The success of IDAPBC hinges on the ability to solve a set of nonlinear partial differential equations (PDE) from which the control function is derived. In general, control methodologies that involve solving nonlinear PDEs, such as optimal control (i.e. the Hamilton-Jacobi-Bellman equation) and feedback linearization, are notoriously computationally expensive. To overcome these challenges, many researchers have turned to learning-based control techniques that aim to solve the control problem using data-driven approaches. In this category, learning algorithms utilize 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 (RL) [10], which seeks a direct mapping from system states to control inputs through repeated

interactions with the environment.

Reinforcement learning algorithms have been used in multiple control tasks such as robot locomotion and manipulation [11, 12] and control of underactuated systems [13]. Despite its success, RL approaches typically suffer from poor sample complexity, slow convergence, and the lack of interpretability of the resulting control policy. These shortcomings stem from the disregard for the potential geometric or algebraic structures that exist in a typical robotic control system. While this increases the flexibility of the approach, it also increases tremendously the amount of training data required. In the analysis of physical control systems, the cost of data acquisition to use these algorithms is effectively prohibitive. Furthermore, despite being the central concern of any control problem, stability considerations in RL-based design methods are often limited to heuristic algorithms and remain an open research question [14, 15].

Recent efforts in machine learning research have unsurprisingly indicated that it is advantageous to inject prior knowledge about the system into the learning framework. The Hamiltonian Neural Network [16] is among the first methods that attempt to incorporate the structure of dynamical systems into a machine learning framework. Using system states as data, this approach finds a neural network that represent the Hamiltonian of the system. The equation of motion is then derived from the learned Hamiltonian. This approach leads to predictions of system trajectories that obey conservation of energy much more accurately than directly inferring the equations of motion of the system, i.e. learning $\dot{x} = f(x)$ directly. Neural ordinary differential equation (ODE) [17], is another 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 machine learning problems, e.g. using Neural ODE to discover the Hamiltonian associated with a mechanical

system from recorded trajectories [18]. The learned Hamiltonian is then used to devise an energy-shaping controller. Physics-Informed Neural Network (PINN) [19] incorporates prior scientific knowledge in the form of partial differential equations in neural network optimization problems. PINN has led to a series of notable results across a range of problems involving high-dimensional PDEs such as fluid mechanics and finite element analysis. In [20], the control function based on a PBC structure is parameterized by a set of user-defined basis functions. An actor-critic method that minimizes the error from the robot's current state to desired state is used to learn the coefficients of these basis functions.

These methods provide an opportunity to use the approximation capabilities of neural networks in existing control design methodologies that were derived from first principles, taking full advantage of the available knowledge about the control system at hand. This dissertation aims to use similar tools to merge many of the clever techniques researchers have proposed with the flexibility of machine learning approaches to design controllers for a family of underactuated robotic systems.

1.1 Summary of Contributions

We introduce two data-efficient learning frameworks that blend the techniques from passivity-based control with the well-known capability of neural networks as universal function approximators. The control problem is cast as an optimization problem that searches for a suitable *storage function*, an essential concept in PBC that facilitates the stability analysis of the closed-loop system and informs control synthesis.

The first framework proposed is referred to henceforth as NEURALPBC, wherein the storage function is represented by a neural network, and the control law is

parameterized by the gradients of the neural net. We develop a training algorithm that efficiently finds a suitable set of parameters such that the corresponding control law drives the system toward some desired equilibrium point. Unlike traditional PBC techniques, this framework is able to incorporate the optimization of any given performance objective defined in terms of the behavior of the closed-loop trajectories of the system. The contributions of NEURALPBC are summarized as follows:

1. Cast the PBC problem as an optimization problem that searches for a set of neural network parameters that best represent the storage (energy-like) function

2. Develop an algorithm that efficiently uses the available data, in the form of system trajectories, to train the parameters such that the controller's performance encoded in the notion of accumulated loss is optimized.

3. Demonstrate the efficacy and robustness of our framework through a series of experimental underactuated robotic systems: the inertia wheel pendulum, the cart-pole system, and the Acrobot.

The second proposed framework is referred to as NEURALIDAPBC. This approach aims to make a transparent connection between classical Lyapunov stability and controllers derived from learning algorithms. Instead of attempting to infer stability from controllers emerging from black-box approaches, we leverage the IDAPBC method to design a learning framework in which stability is an *intrinsic* property. The contributions of NEURALIDAPBC are summarized as follows:

1. Formulate an optimization problem that achieves the objective of IDAPBC,

2. Solve the optimization problem using a combination of deep neural networks and/or Sum-of-Squares (SoS) polynomials as surrogates for the solution,

3. Show that as the surrogates converge to the true solution, our method yields a faithful IdaPbc control law, preserving the intrinsic stability property,

4. Demonstrate the efficacy of the controllers through experiments on the inertia wheel pendulum and the ball-beam system.

1.2 Comparisons to Related Work

In this subsection, we compare our work to related methods that incorporate prior physical knowledge in machine learning frameworks, particularly in the subject of dynamics and control of underactuated systems.

The Symplectic-ODE method [18] uses Neural ODE [17] to learn the robot's Hamiltonian dynamics using system trajectories as data. Once the Hamiltonian dynamics is learned, the authors show that the learned model can be used to develop controllers using familiar Pbc techniques [21]. In our NeuralPbc approach, the aim is not to learn the dynamics of the system. Instead, we leverage the known dynamics and incorporate it into a data-driven control design framework.

In [19], the approach of using neural networks as surrogates for solving PDEs is referred to as Physics Informed Neural Network (PINN). The NeuralIdaPbc framework we present here resembles the PINN approach to a small degree, as both approaches offer a basis to inject physical laws in the form of PDEs into a machine learning framework. However, in our work, the solution surrogates for the PDEs are constrained differently. In PINN, the boundary conditions and other constraints are represented in the objective function, whereas we constrain by the construction of the relevant physical quantities, such as the positive-definiteness of the mass matrix, and

the boundedness of the potential energy. Our approach ensures that these quantities are physically valid even when the training has not yet converged.

1.3 Summary of Publications

The contents of Chapter 3 (NeuralPbc) appear in the following publications:

1. [22] W. Sirichotiyakul and A. C. Satici, "Data-driven design of energy-shaping controllers for swing-up control of underactuated robots," in *International Symposium on Experimental Robotics*. Springer, 2020, pp. 323-333.

2. [23] W. Sirichotiyakul and A. C. Satici, "Combining energy-shaping control of dynamical systems with data-driven approaches," in *2021 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2021, pp. 1121-1127.

The contents of Chapter 4 (NeuralIdaPbc) appear in the following publication:

1. [24] W. Sirichotiyakul and A. C. Satici, "Data-Driven Passivity-Based Control of Underactuated Mechanical Systems via Interconnection and Damping Assignment," in *International Journal of Control*. (Accepted for publication March 2022).

The contents of this dissertation lay the groundwork for an ongoing research that extends NeuralPbc and NeuralIdaPbc to account for uncertainty in the parameters of the nominal dynamical model used during training. Preliminary results from this research area are presented in the following articles:

1. [25] W. Sirichotiyakul, N. A. Ashenafi, and A. C. Satici, "Robust Data-Driven Passivity-Based Control of Underactuated Systems via Neural Approximators

and Bayesian Inference," in *2022 American Control Conference, ACC*. IEEE, Accepted for publication January 2022.

2. N. A. Ashenafi, W. Sirichotiyakul, and A. C. Satici, "Robust Passivity-Based Control of Underactuated Systems via Neural Approximators and Bayesian Inference," in *2022 Conference on Decision and Control, CDC*. IEEE. (Submitted for review March 2022).

3. W. Sirichotiyakul, N. A. Ashenafi, and A. C. Satici, "Robust Interconnection and Damping Assignment Passivity-Based Control via Neural Bayesian Inference," in *IEEE Transactions on Automatic Control*. (Submitted for review April 2022).

CHAPTER 2

PRELIMINARIES AND BACKGROUND

This chapter aims to provide the reader with the preliminary background to formulate the passivity-based control design as a neural network optimization problem. In Section 2.1 we provide a brief overview of the connection between Lagrangian mechanics to Hamiltonian mechanics, the modeling basis used in this dissertation. Section 2.2 introduces the concepts of passive systems and their inherent stability properties. Properties of passive systems are the fundamental concepts for the control synthesis in a family of techniques called passivity-based control (PBC), described in Section 2.2.2. The machine learning frameworks introduced in Chapters 3 and 4 require familiarity with transverse coordinates, adjoint sensitivity analysis, and sums-of-square (SoS) polynomials. A cursory exposition of these subjects are presented in Section 2.3 through Section 2.5, and references for further study are provided therein.

2.1 Hamiltonian Mechanics

Following [26], we begin by stating the equations of motion of a physical system from the Lagrangian point of view. Let $x \in \mathbb{R}^{2n}$ denote the state of a conservative dynamical system. The state x may be represented in terms of the generalized positions $q \in \mathbb{R}^n$ and their velocities, i.e. $x = (q, \dot{q})$. The evolution of the system in the configuration space from point to another is governed by the Euler-Lagrange

equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = 0, \tag{2.1}$$

where $\mathcal{L}$ is the Lagrangian, defined as the difference between kinetic energy $\mathcal{T}$ and potential energy $\mathcal{T}$:

$$\mathcal{L} = \mathcal{T} - \mathcal{V}.$$

It can be shown, using techniques of calculus of variations, that the dynamic path of a system described by the Lagrangian $\mathcal{L}$ from time $t = t_0$ to $t = t_1$ coincide with the extremal of the functional

$$\Phi = \int_{t_0}^{t_1} \mathcal{L}(q, \dot{q}, t)\ \mathrm{d}t.$$

This notion is called the Hamilton's principle of stationary action. This principle reduces the problem of deriving the equations of motion of a system to an optimization problem of a functional.

The Lagrangian formulation of mechanics is not the only possible way to describe the motion of a system. There is also the Hamiltonian formulation, where we describe the state of the system in terms of the generalized coordinates q and momenta p. For simplicity, we restrict ourselves to mechanical systems and view the Hamiltonian formulation as a consequence of the Euler-Lagrange equation and a simple change of variables. We begin by defining a conjugate variable p, called the generalized momenta, as follows

$$p \triangleq \frac{\partial \mathcal{L}(q, \dot{q}, t)}{\partial \dot{q}}. \tag{2.2}$$

The Hamiltonian is defined as the Legendre transform of the Lagrangian with respect to the conjugate variable p, i.e.

$$\mathcal{H}(q,p,t) = p^\top \dot{q} - \mathcal{L}(q,\dot{q},t) \tag{2.3}$$

For most mechanical systems where the kinetic energy is a homogeneous quadratic function in $\dot{q}$, the Hamiltonian is equivalent to the total energy, i.e.

$$\mathcal{H} = \mathcal{T} + \mathcal{V}.$$

Then, from the Euler-Lagrange equation (2.1), we can directly deduce the Hamilton's canonical equations of motion as

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \begin{bmatrix} \nabla_q \mathcal{H} \\ \nabla_p \mathcal{H} \end{bmatrix}. \tag{2.4}$$

These equations are $2n$-dimensional, first-order ordinary differential equations.

2.2 Passive System Theory

In this subsection, a brief overview of the theory of passive systems is treated following [5, 27], laying the foundation for the work developed in this dissertation. Consider the state space of the general form

$$\Sigma : \begin{aligned} \dot{x} &= f(x,u), \\ y &= h(x,u), \end{aligned} \tag{2.5}$$

where $x \in \mathcal{X} \subset \mathbb{R}^{2n}$ is the local coordinate for a $2n$-dimensional state space, $u \in \mathcal{U} \subset \mathbb{R}^m$ is the control input, and $y \in \mathcal{Y} \subset \mathbb{R}^m$ is the output. We assume that $f : \mathbb{R}^{2n} \times \mathbb{R}^m \to \mathbb{R}^{2n}$ is locally Lipschitz, $h : \mathbb{R}^{2n} \times \mathbb{R}^m \to \mathbb{R}^m$ is continuous, $f(0,0) = 0$

and $h(0,0) = 0$. The state of a robotic system may be represented in terms of the generalized positions $q \in \mathbb{R}^n$ and momenta $p \in \mathbb{R}^n$, i.e., $x = (q, p)$.

Definition 2.2.1 (Dissipative Property). The system Σ is *dissipative* with respect to a supply rate s if there exists a nonnegative function $\mathcal{H} : \mathcal{X} \to \mathbb{R}^+$, called the storage function, such that the following inequality holds for all initial conditions $x(t_0) = x_0$ at any time t_0, all input function u, and all $t_1 \geq t_0$:

$$\mathcal{H}(x(t_1)) \leq \mathcal{H}(x(t_0)) + \int_{t_0}^{t_1} s(u(t), y(t)) \, \mathrm{d}t. \tag{2.6}$$

The inequality (2.6) is called the dissipation inequality. It states that the energy stored in the system at a future time t_1 is at most equal to the energy stored at the present time t_0, plus the total externally supplied energy $s(t)$ accumulated during the time interval (t_0, t_1). That is, in dissipative systems, there is no generation of energy, and only the internal dissipation of energy is possible.

A dynamical system that is dissipative with respect to a particular choice of the supply rate function s is *passive*.

Definition 2.2.2 (Passive Property). The system Σ with $\mathcal{U}, \mathcal{Y} \subset \mathbb{R}^m$ is *passive* if it is dissipative with respect to the supply rate $s(u, y) = u^\top y$. That is, Σ is passive if there exists a continuously differentiable storage function $\mathcal{H}(x)$ such that

$$u^\top y \geq \dot{\mathcal{H}} = \frac{\partial \mathcal{H}}{\partial x} f(x, u), \quad \forall \, (x, u) \in \mathbb{R}^{2n} \times \mathbb{R}^m.$$

Moreover, Σ is said to be

- *input strictly passive* if there exists $a > 0$ such that $u^\top y \geq \dot{\mathcal{H}} + a \left\| u \right\|^2$
- *output strictly passive* if there exists $b > 0$ such that $u^\top y \geq \dot{\mathcal{H}} + b \left\| y \right\|^2$.

- *strictly passive* if there exists $c > 0$ such that $u^\top y \geq \dot{\mathcal{H}} + c\,\|x\|^2$.

Example 2.2.3. Consider the electrical circuit shown in Figure 2.1. By Kirchoff's law, the system is described by the differential equation

$$\Sigma_e : \quad v = Ri + \frac{1}{C}\int_0^t i(\tau)\,\mathrm{d}\tau + L\frac{\mathrm{d}i}{\mathrm{d}t}.$$

where v is the source voltage, i is the current, L is the inductance, R is the resistance, and C is the capacitance. Rearranging, we have

$$vi - Ri^2 = \frac{\mathrm{d}}{\mathrm{d}t}\left(\underbrace{\frac{1}{2C}\left(\int_0^t i(\tau)\,\mathrm{d}\tau\right)^2}_{\mathcal{V}} + \underbrace{\frac{1}{2}Li^2}_{\mathcal{T}}\right).$$

Let $\mathcal{H} = \mathcal{V} + \mathcal{T}$, and integrate to obtain

$$\mathcal{H}(t) - \mathcal{H}(0) = \int_0^t v(\tau)i(\tau)\,\mathrm{d}\tau - \int_0^t Ri^2(\tau)\,\mathrm{d}\tau \leq \int_0^t v(\tau)i(\tau)\,\mathrm{d}\tau.$$

Therefore, the system Σ_e is passive with respect to the supply rate $s(t) = v(t)i(t)$ and the storage function $\mathcal{H}$.

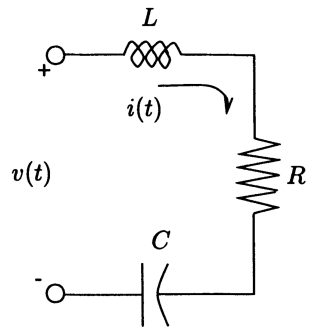


Figure 2.1: The RLC circuit is a passive system with the storage function $\mathcal{H} = \mathcal{V} + \mathcal{T}$.

2.2.1 Stability of Passive Systems

Passivity lays the groundwork as an important tool for the analysis of autonomous dynamical systems, as the property relates nicely to Lyapunov and $\mathcal{L}_2$ stability. We provide a cursory exposition of this important connection in the following subsection. The interested reader may refer to [5, 27] and references therein for a detailed explanation.

Lemma 2.2.4. *If the system Σ given by (2.5) is passive with a positive definite storage function $\mathcal{H}(x)$, then the origin of $\dot{x} = f(x, 0)$ is stable.*

Proof. Take $\mathcal{H}$ as a Lyapunov function candidate for $\dot{x} = f(x, 0)$. Then, $\dot{\mathcal{H}} \leq 0$. □

To prove asymptotic stability of the origin of f, we apply LaSalle's invariance principle by considering a case where $\dot{\mathcal{H}} = 0$ when $y = 0$, and then require the additional property that

$$y(t) \equiv 0 \Rightarrow x(t) \equiv 0 \tag{2.7}$$

for all solutions of (2.5). Equivalently, this requires that no solutions of $f(x, 0)$ can stay identically in the set $\{x \in \mathbb{R}^{2n} \,|\, h(x, 0) = 0\}$ except for the trivial solution $x(t) \equiv 0$. The property (2.7) may be interpreted as an observability condition.

Definition 2.2.5. The system Σ is *zero-state observable* if no solution of $\dot{x} = f(x, 0)$ can stay identically in $\{x \in \mathbb{R}^{2n} \,|\, h(x, 0) = 0\}$ other than the trivial solution $x(t) \equiv 0$.

Lemma 2.2.6. *The origin of (2.5) is asymptotically stable if the system is*

- *strictly passive, or*
- *output strictly passive and zero-state observable.*

Furthermore, if the storage function is radially unbounded, the origin will be globally asymptotically stable.

Proof. See [27] for the proof if this lemma. □

These results provide the fundamental concepts used in *passivity-based control*, a paradigm in which the control objective is to transform the system Σ such that the closed-loop system is rendered passive with respect to some desired storage function. This dissertation introduces a systematic way to automatically find a suitable storage function and synthesize controllers using PBC techniques. The following subsection provides the preliminary background on the subject of PBC.

2.2.2 Passivity-Based Control

Let $x \in \mathcal{X} \subset \mathbb{R}^{2n}$ denote the state of the robot. The state x is represented in terms of the generalized positions $q \in \mathbb{R}^n$ and momenta $p \in \mathbb{R}^n$, i.e., $x = (q, p)$. The Hamiltonian H of the robot is defined as

$$H(q,p) = \frac{1}{2} p^\top M^{-1}(q) p + V(q), \tag{2.8}$$

where $M : \mathbb{R}^n \to \mathbb{S}^{n\times n}_{++}$ is the positive-definite mass matrix, and $V : \mathbb{R}^n \to \mathbb{R}$ represents the potential energy. Henceforth, we denote the positive definiteness of a matrix with the notation $\succ 0$, and the positive semidefiniteness with the notation $\succeq 0$. The system's equations of motion can then be expressed as

$$\begin{aligned} \begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} &= \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix} \begin{bmatrix} \nabla_q H \\ \nabla_p H \end{bmatrix} + \begin{bmatrix} 0 \\ G(q) \end{bmatrix} u, \\ y &= G^\top \dot{q}, \end{aligned} \tag{2.9}$$

where ∇_x denotes vector of partial derivatives of a function with respect to x, $G(q) \in \mathbb{R}^{n\times m}$ is the input matrix, I_n denotes the $n \times n$ identity matrix, and $u \in \mathcal{U} \subset \mathbb{R}^m$ is the control input.

Definition 2.2.7. The system (2.9) is *underactuated* if rank $G = m < n$.

The quintessential idea of PBC [5] is to design the control input u with the objective of imposing a desired storage function $H_d : \mathcal{X} \to \mathbb{R}$ on the closed-loop system, rendering it passive. In the standard PBC formulation, the control law u comprises two fundamental steps: 1) *energy shaping* u_{es} where the total energy function of the system is modified in order to assign the new desired equilibrium $q^\star$, and 2) *damping injection* u_{di} to achieve asymptotic stability

$$u = u_{es}(x) + u_{di}(x), \tag{2.10}$$

where u_{es} and u_{di} are selected such that the closed loop dynamics satisfies

$$H_d(x(t)) - H_d(x(0)) = \int_0^t u_{di}^\top(\tau) y(\tau)\ \mathrm{d}\tau - d^\star(x),$$

where $d^\star \geq 0$ is the desired damping dissipation. For mechanical systems, one solution to the PBC problem is of the form

$$u_{es}(x) = -G^\dagger \left(\nabla_q H_d - \nabla_q H\right), \tag{2.11}$$

$$u_{di}(x) = -K_v\, y, \tag{2.12}$$

where $G^\dagger = \left(G^\top G\right)^{-1} G^\top$, and $K_v \succ 0$ is the damping gain matrix. The most straightforward solution is to assign a quadratic potential energy such that the minimum is

at $q^\star$, i.e., with $K_p \succ 0$.

$$H_d(q,p) = \frac{1}{2}p^\top M^{-1}(q)p + \frac{1}{2}(q-q^\star)^\top K_p(q-q^\star). \tag{2.13}$$

Note that this choice is untenable for most underactuated systems and is only suitable for fully actuated or redundant systems. There is, in general, an infinite number of choices for H_d that achieves the objective of PBC. In Chapter 3, we embed PBC techniques into a machine learning framework with an objective of automatically discovering a suitable choice for H_d such that a certain objective function, defined based on the general behavior of the closed-loop trajectories, is optimized.

Since the first introduction of PBC in the 1980s, control researchers have proposed several systematic methods to design a suitable H_d, leading to a variety of PBC techniques. Extensions of the PBC approach introduce additional algebraic structure to the system in order to improve performance and ease stability analysis for a family of dynamical systems. In the following subsection, we present one such extension called the interconnection and damping assignment passivity-based control (IDAPBC) [8], which is a prominent approach in the domain of underactuated mechanical systems.

2.2.3 Interconnection and Damping Assignment

The materials described in this subsection are used in Chapter 4 of this dissertation. In IDAPBC, the closed-loop dynamics is chosen to take the port-controlled Hamiltonian (PCH) form given by

$$\begin{bmatrix}\dot{q}\\ \dot{p}\end{bmatrix} = \Big[J_d(q,p) - R_d(q,p)\Big]\begin{bmatrix}\nabla_q H_d\\ \nabla_p H_d\end{bmatrix} \tag{2.14}$$

where J_d and R_d are, respectively, the desired interconnection and damping matrices:

$$J_d = -J_d^\top = \begin{bmatrix} 0 & M^{-1}M_d \\ -M_dM^{-1} & J_2(q,p) \end{bmatrix}, \quad R_d = R_d^\top = \begin{bmatrix} 0 & 0 \\ 0 & GK_vG^\top \end{bmatrix} \succeq 0,$$

with $J_2 = -J_2^\top$ and $K_v \succ 0$ is a tunable gain matrix for damping injection. The storage function H_d is chosen as a Hamiltonian of a fictitious mechanical system, i.e., H_d is quadratic in the system momenta p:

$$H_d(q,p) = \frac{1}{2}p^\top M_d^{-1}(q)p + V_d(q), \tag{2.15}$$

where $M_d(q) \succ 0$ is the closed-loop, positive definite mass matrix and $V_d : \mathbb{R}^n \to \mathbb{R}$ is the closed-loop potential energy function that satisfies

$$q^\star = \underset{q}{\operatorname{argmin}} \;\; V_d(q). \tag{2.16}$$

With the control law of the form given by (2.10), the energy shaping term u_{es} may be obtained by equating the system given by (2.9) to the one given by (2.14)

$$\begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \begin{bmatrix} \nabla_q H \\ \nabla_p H \end{bmatrix} + \begin{bmatrix} 0 \\ G \end{bmatrix} (u_{es} + u_{di}) = \begin{bmatrix} 0 & M^{-1}M_d \\ -M_dM^{-1} & J_2(q,p) - GK_vG^\top \end{bmatrix} \begin{bmatrix} \nabla_q H_d \\ \nabla_p H_d \end{bmatrix}.$$

With the damping injection term chosen as

$$u_{di} = -K_vG^\top \nabla_p H_d, \tag{2.17}$$

the equation of motion is reduced to

$$\begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \begin{bmatrix} \nabla_q H \\ \nabla_p H \end{bmatrix} + \begin{bmatrix} 0 \\ G \end{bmatrix} u_{es} = \begin{bmatrix} 0 & M^{-1}M_d \\ -M_d M^{-1} & J_2(q,p) \end{bmatrix} \begin{bmatrix} \nabla_q H_d \\ \nabla_p H_d \end{bmatrix}.$$

The first row of these equations is trivially satisfied, revealing the matching condition

$$G u_{es} = \nabla_q H - M_d M^{-1} \nabla_q H_d + J_2 M_d^{-1} p. \tag{2.18}$$

If the system is underactuated, G is not invertible, and Equation (2.18) can only be solved if the following constraint is satisfied for any choice of u_{es}:

$$G^{\perp} \left\{ \nabla_q H - M_d M^{-1} \nabla_q H_d + J_2 M_d^{-1} p \right\} = 0. \tag{2.19}$$

Equation (2.19) is a set of nonlinear PDEs, which is often referred to as the *matching equations* in the literature, parametrized by M_d, V_d, and J_2. The skew-symmetric matrix J_2 serves as free parameters to ease the burden of solving the PDEs. Once a suitable H_d is determined, the energy shaping term is given by

$$u_{es} = G^{\dagger} \left(\nabla_q H - M_d M^{-1} \nabla_q H_d + J_2 M_d^{-1} p \right), \tag{2.20}$$

where $G^{\dagger} = \left(G^{\top} G \right)^{-1} G^{\top}$ is the left pseudo-inverse of G.

The success of the IDAPBC approach hinges on the ability to obtain the closed-form solutions to the nonlinear PDEs given by (2.19). That is, as long as the choice of H_d satisfies (2.19) and (2.16), the closed-loop dynamics takes on the form given by (2.14), and the following proposition reveals the stabilization properties of the passivity-based control law $u = u_{es} + u_{di}$:

Proposition 2.2.8. *The closed-loop Hamiltonian H_d in IDAPBC is, by construction,*

a Lyapunov function for the closed-loop system. The time-derivative of H_d *is*

$$\begin{aligned}\dot{H}_d = (\nabla_q H_d)^\top \dot{q} + (\nabla_p H_d)^\top \dot{p} &= (\nabla_p H_d)^\top \left(J_2 - G K_v G^\top\right) \nabla_p H_d \\ &\leq -\lambda_{min}\{K_v\} \left|(\nabla_p H_d)^\top G\right|^2 \leq 0,\end{aligned}$$

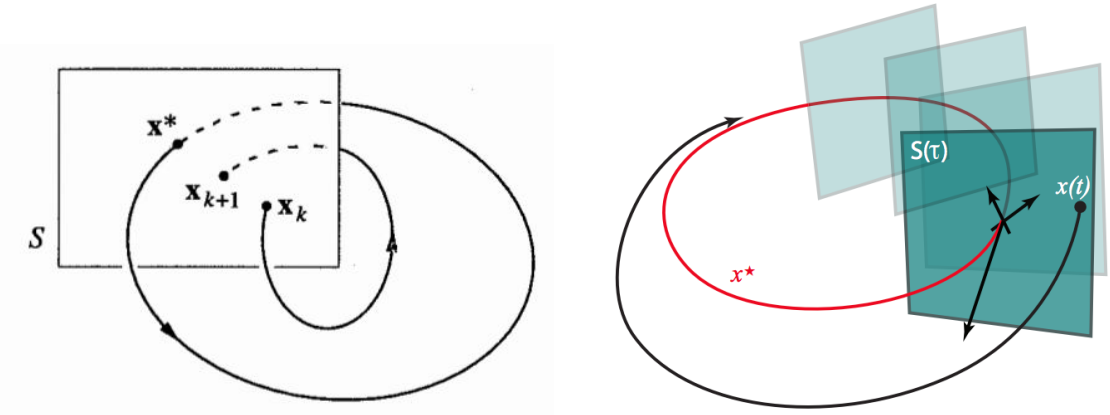
where the last inequality follows from $J_2 = -J_2^\top$ *and* $K_v \succ 0$*. Therefore, as long as* V_d *is bounded from below and the conditions* (2.16) *and* (2.19) *are satisfied,* $(q^\star, 0)$ *is a stable equilibrium of the system* (2.14)*.*

As with other nonlinear PDEs, solving the matching equations is computationally expensive. Since the original introduction of the IDAPBC approach in the early 2000s, researchers have worked to classify a family of dynamical systems to which the method is applicable. In [8], the authors identified a class of underactuated mechanical systems where the matching equations are solvable. Chapter 4 of this dissertation presents a way to automatically and efficiently find an approximate solution to the matching equations.

2.3 Transverse Coordinates

One of the performance objectives considered for the control design framework introduced in this dissertation is the ability to track a given orbit or trajectory $\gamma^\star$. An orbit is the solution (or flow) to the (ODE) (2.5) starting at a given $x(t_0) = x_0$. This subsection describes an efficient method to quantify how closely the system is tracking this orbit, using the concept of *transverse coordinates* [28, 29, 30].

The transversal coordinate system is adapted to the desired orbit $\gamma^\star$. In these coordinates, the error between the current state x and $\gamma^\star$ can be efficiently computed.



(a) Single Poincaré section [31] (b) Moving Poincaré sections [28]

Figure 2.2: Visualization of Poincaré section

In order to define this adapted coordinate system, we start by explaining the idea of a moving Poincaré section.

Given a state $x \in \mathbb{R}^{2n}$ in the original coordinate system, define τ, a scalar coordinate aligned with the flow of the system along the desired orbit $\gamma^\star$. At each τ, we define a $2n-1$ dimensional hypersurface transversal to this orbit, resulting in a family of hypersurfaces $\mathscr{S}(\tau)$, mimicking the behavior of a Poincaré section that is *moving* along the orbit. This is visualized in Figure 2.2.

The remaining coordinates, denoted $x_\perp \in \mathbb{R}^{2n-1}$ and referred to as the *transverse coordinates*, are defined such that they represent the location of the state x on each $\mathscr{S}(\tau)$ relative to the desired orbit, with $x_\perp = 0$ implying that $x = \gamma^\star(\tau)$. Once the new coordinate system is constructed, we then define a smooth mapping $\Pi : x \mapsto (x_\perp, \tau)$ that relates original coordinate system to the transversal coordinates

$$x_\perp(\tau) = \Pi(\tau)\, x, \quad x \in \mathscr{S}(\tau). \tag{2.21}$$

This construction gives us a continuous representation of the system's evolution

along the orbit, as opposed to at a single section when using the Poincaré map. Moreover, it allows for an efficient definition of a penalty or loss incurred whenever the state x spends time away from the desired orbit, e.g. by requiring that penalty vanishes everywhere along the orbit, and is strictly positive everywhere away from the orbit. The concept of transverse coordinates is used in the NeuralPbc framework presented in Chapter 3.

2.4 Neural Ordinary Differential Equations

Neural ordinary differential equation (ODE) is a deep learning model recently developed in [17]. Starting with the initial value problem (IVP)

$$\dot{x} = f(x), \quad x(t_0) = x_0,$$

this method parametrizes the dynamics with a neural network $f_\theta(x)$ and uses time series data $\hat{x} = (\hat{x}_{t_0}, \ldots, \hat{x}_{t_f})$ to learn to approximate any sufficiently smooth f. Collecting training data for f_θ consists of numerically integrating the IVP to obtain a solution $x_\theta(t)$ with respect to the current choice of the neural network parameters θ. Training a neural ODE then amounts to minimizing a cost function, for example,

$$J(\theta) = \sum_{\tau=t_0}^{t_f} \left\| x_\theta(\tau) - \hat{x}_\tau \right\|.$$

This optimization problem may be solved with any gradient-based method, as long as the computation of the gradient $\nabla_\theta J$ is feasible. It is not immediately clear how this quantity may be computed. Employing the chain rule yields that this amounts

to calculating $\partial x/\partial\theta$, the gradients of the solution $x_\theta(t)$ with respect to the neural network parameters.

From a software implementation perspective, the quantity $\partial x/\partial\theta$ may be obtained by leveraging the recent development of automatic differentiation (AD) suites available in most modern machine learning libraries, e.g. `PyTorch`, `Flux.jl`, `TensorFlow`. However, combining AD with numerical ODE solvers introduces numerical errors in the gradients. Furthermore, the required memory footprint is significant, limiting their applicability in practice.

To combat memory consumption issues, the application of adjoint sensitivity methods in a machine learning framework is first introduced by [17]. This method computes the desired derivative $\partial J/\partial\theta$ by first defining another ODE, known as the adjoint problem:

$$\frac{\mathrm{d}\lambda}{\mathrm{d}t} = -\lambda\frac{\partial f}{\partial x}.$$

The desired gradient of the loss function may be computed through the following expression that depends on the solution to the adjoint problem:

$$\frac{\partial J}{\partial\theta} = \lambda(t_0)\frac{\partial f}{\partial x}.$$

This approach is based on Pontryagin's maximum principle [32] and is quite mathematically elegant. The adjoint problem needs to be solved backwards in time, which may be done numerically [33, 34]. However, the numerical methods for solving the adjoint problem require specialized implementations of the solver. Furthermore, they may require storing multiple forward solutions of the ODE to implement; hence, their execution can quickly become quite computationally costly. These obstacles make it difficult for researchers to apply the adjoint method, as most of the available numerical

differential equation solvers are not designed to accommodate this feature.

Recent advances in differentiable programming has enabled more efficient computations of the desired gradients through a solver implemented entirely in a language with pervasive AD. In [35], several automatic differentiation systems and the adjoint method are generalized to enable efficient gradient computation through solutions of ODEs. In Chapter 3, we leverage this recent development for the computation of the relevant gradients in our control design framework.

2.5 Sum of Squares Polynomials

Control synthesis using passivity-based control techniques relies on the discovery of a storage function, one manifestation of which may be the total energy in mechanical systems. One of the qualifications of a storage function is a well-defined, isolated minimum at some desired equilibrium point. This allows the storage function to be conveniently used as a Lyapunov function candidate to prove stability.

Without loss of generality, the isolated minimum condition can be imposed by choosing a storage function that is nonnegative and attains a value of zero at the desired equilibrium. A sum of square (SoS) polynomial offers a computationally efficient way to achieve this property. In this section, we provide the necessary background relating positive semidefinite matrices to SoS polynomials. We emphasize the aspect of computing the SoS decomposition. The interested reader may refer to [36, 37] for more information.

Definition 2.5.1 (SoS Polynomial). A polynomial $P \in \mathbb{R}[x]$ of degree $d = \eta_1 + \cdots + \eta_n$,

$$P(x) = \sum_{\eta_1+\cdots+\eta_n \leq d} c_\eta x_1^{\eta_1} \cdots x_n^{\eta_n},\ \eta_i \in \mathbb{N},$$

is a sum-of-squares if there exist a finite number of polynomials $P_i \in \mathbb{R}[x]$ such that P can be written as $P(x) = \sum_i P_i^2(x)$.

Note that if $P(x)$ is an SoS, then $P(x) \geq 0\ \forall x \in \mathbb{R}^n$. David Hilbert proved that not every positive semidefinite polynomial can be written as a sum-of-squares. However, in 1927, Artin's Theorem answered Hilbert's seventeenth problem by stating that every semidefinite polynomial is a sum-of-squares of rational functions [38]. This leads us to the question: When can a polynomial be written as a sum-of-squares?

Theorem 2.5.2. *[39] A polynomial $P \in \mathbb{R}[x]$ of degree $2d$ has a sum-of-squares decomposition if and only if there exists a positive semidefinite matrix Q such that*

$$P(x) = \mu^\top(x) Q \mu(x),$$

where μ is the vector of all monomials in $x_1, \ldots, x_n$ of degree less than or equal to d, i.e. $\mu(x) = \begin{bmatrix} 1 & x_1 & x_2 & \ldots & x_n & x_1^2 & x_1x_2 & \ldots & x_n^d \end{bmatrix}$. There exist $\binom{n+d}{n}$ such monomials.

This representation theorem, based on the Gram matrix method, implies that the set of sum-of-squares polynomials are parametrized by the (convex) set of positive semidefinite matrices. Searching for a SoS polynomial can be formulated as an optimization problem that searches for a positive semidefinite matrix Q that will ensure the nonnegativity of $P(x)$ for any $x \in \mathbb{R}^n$ while satisfying a set of affine constraints. Note that this is a *finite-dimensional, convex optimization problem*, which can be solved efficiently via semidefinite programming [40].

Example 2.5.3. An example taken from [37] illustrates the theorem. Consider the polynomial in two indeterminates of degree 4

$$P(x) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4.$$

We want to check whether P can be written as a sum-of-squares polynomial. By Theorem 2.5.2, we search for a matrix Q such that with the vector of monomials defined as $\mu(x) = \begin{bmatrix} x_1^2 & x_2^2 & x_1x_2 \end{bmatrix}^\top$, we have

$$\begin{aligned} p(x) &= \mu^\top(x) Q \mu(x) \\ &= \mu^\top(x) \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \mu(x) \\ &= q_{11}x_1^4 + q_{22}x_2^4 + q_{33}x_1^2x_2^2 + 2q_{12}x_1^2x_2^2 + 2q_{13}x_1^3x_2 + 2q_{23}x_1x_2^3. \end{aligned}$$

Equating the coefficients leads to

$$q_{11} = 2,\ q_{22} = 5,\ q_{33} + 2q_{12} = -1,\ q_{13} = 1,\ q_{23} = 0.$$

We are left with one unknown parameter, q_{12} (or q_{33}). Whether $P \in \mathbb{R}[x]$ is a SoS polynomial or not is equivalent to questioning a q_{12} exists such that $Q \succeq 0$. The feasibility of the following semidefinite program then would verify that p is a sum-of-squares polynomial

$$Q = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 5 & 0 \\ 1 & 0 & -1 \end{bmatrix} + q_{12} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -2 \end{bmatrix} \succeq 0.$$

One solution to this feasibility problem is $q_{12} = -3$, which implies that p is indeed an SoS polynomial.

Remark 2.5.4. By the Cholesky decomposition [41], a matrix $Q \in \mathbb{R}^{m \times m}$ is non-negative definite if Q can be factored as $Q = LL^\top$, where L is a lower-triangular matrix. This allows an efficient way to construct SoS polynomials by constructing a lower-triangular matrix L with real entries.

In Chapter 4, we use Theorem 2.5.2 along with the Cholesky decomposition in an optimization framework to efficiently find a suitable storage function that achieves the objective of IDAPBC.

CHAPTER 3

NEURAL PASSIVITY-BASED CONTROL

This chapter describes NeuralPbc, a control design approach that integrates passivity based control methods with a machine learning framework. This approach directly embeds a learning-based control law, which is parameterized by the gradient of a neural network, into the dynamical model of a robotic system. The derivation of the control policy is inspired by the techniques of Pbc, where the control action depends on the gradient of an energy-like (storage) function, akin to a Lyapunov function for stability analysis of autonomous dynamical systems. We invoke techniques from machine learning to automatically determine this energy-like function through an optimization of a cost function, derived from the general behavior of the closed-loop system trajectories.

3.1 Problem Statement

Concretely, this chapter considers mechanical systems governed by equations of motions of the form given by Eq. (2.9), which is repeated here for convenience:

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

where the state $x = (q, p)$, the Hamiltonian is $H = \frac{1}{2}p^\top M(q)p + V(q)$, $M \succ 0$ is the positive-definite mass matrix, and $V(q)$ is the potential energy. We consider the task of asymptotically stabilizing a desired equilibrium $x^\star = (q^\star, 0)$ of (2.9), using passivity-based control techniques described in Section 2.2.

We choose the control law u of the form given by Eq. (2.10) with the energy shaping term u_{es} and the damping injection term u_{di} defined according to Eq. (2.11) and (2.12), respectively. However, rather than limiting the closed-loop energy-like function H_d to be of a certain form, for instance the one given by Eq. (2.13), we leverage the expressive power of neural networks and formulate an optimization problem to automatically come up with a suitable function. We aim to find an energy-like function H_d such that the behavior of the closed-loop trajectories optimizes a certain objective encoded in the notion of accumulated loss. In this setting, the control task is viewed as ensuring the trajectories of (2.9) pass through a small neighborhood of the desired equilibrium $x^\star$, at which point a standard linear controller may be employed.

Let $H_d^\theta : \mathcal{X} \to \mathbb{R}$ be a neural network, and $K_v^\theta \in \mathbb{R}^{m\times m}$ be a positive-definite matrix. Let $\phi = \phi\,(t, x_0, u)$ denote the flow of the equations of motion (2.9) with the initial condition $x_0 \in \mathcal{X}$. With the decision variables $\theta \in \mathbb{R}^w$ comprising the parameters of H_d^θ and the entries of K_v^θ, we introduce the minimization problem:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & J(\theta) = \int_0^T \ell\left(\phi, u^\theta\right)\,\mathrm{d}t, \\ \text{subject to} \quad & \dot{x} = \begin{bmatrix} \nabla_p H \\ -\nabla_q H \end{bmatrix} + \begin{bmatrix} 0 \\ G(q) \end{bmatrix} u^\theta, \\ & u^\theta = -G^\dagger \nabla_q H_d^\theta - K_v^\theta G^\top \nabla_p H_d^\theta, \end{aligned} \tag{3.1}$$

where $T > 0$ is the time horizon, and $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^+$ is a running cost function through which the control task is encoded. For any given x_0, we denote by $\gamma : t \to \phi(t; x_0, u)$ the closed-loop trajectory of (2.9) under the application of the control law $u = u^\theta$ with the parameters θ fixed.

The running cost function ℓ that represents the control objective is defined as

$$\ell := \ell_{\text{set}}(\gamma) + \ell_\perp \left(\gamma, u^\theta\right), \tag{3.2}$$

where ℓ_{set} is the *set distance* from the closed-loop trajectory γ to a goal set $\mathcal{S}$ containing a neighborhood of $x^\star$, and $\ell_\perp$ is the *transversal distance* between γ and a preferred orbit $\gamma^\star$. This formulation allows the user to additionally include the performance objective of a task execution in the optimization, a feat that is not easily achievable in the original PBC framework. Each term of ℓ is now described in detail as follows.

1. The set distance ℓ_{set} is constructed by defining a convex open neighborhood $\mathcal{S}$ containing $x^\star$ and computing the set distance of γ to $\mathcal{S}$:

$$\ell_{\text{set}}(\gamma) = \inf_t \{\|a - b\| : a \in \gamma(t),\, b \in \mathcal{S}\}. \tag{3.3}$$

   For instance, the set $\mathcal{S}$ may be chosen as a ball around of radius r around $x^\star$ in the standard norm topology. Here, r becomes a hyperparameter of the training algorithm. With this particular $\mathcal{S}$, if at any point along the prediction γ the state x is closer than r to $x^\star$, no penalty is incurred by ℓ_{set}.

2. The transversal distance $\ell_\perp$ to a preferred orbit $\gamma^\star$ is defined in terms of the transverse coordinates [42], denoted by $x_\perp$, along $\gamma^\star$. By construction, the

prediction γ converges to $\gamma^\star$ if and only if $x_\perp \to 0$ as $t \to \infty$. This provides an efficient way to compute the penalty $\ell_\perp$ as

$$\ell_\perp(x, u) = x_\perp^\top Q x_\perp + u^\top R u,\ Q \succeq 0,\ R \succeq 0. \tag{3.4}$$

The preferred orbit may be generated using any motion planning algorithm that runs fast enough, such as RRT [43], A* [44], trajectory optimization [45], virtual holonomic constraints [46], etc. In some cases, there are natural choices for these preferred orbits, such as the homoclinic orbit of the pendulum [47]. If there is no preferable $\gamma^\star$ available, we simply let $Q = 0$ and only incur penalty due to the expenditure of the control effort u.

Symmetry of the Control Policy

For robotic systems that naturally possess symmetric properties, e.g. $f(x) = -f(-x)$, it is desirable to maintain the symmetry in the closed-loop system. Imposing this symmetry constraint in the control function halves 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.

This property can be achieved by designing a symmetric control policy $u^\theta(x) = -u^\theta(-x)$ through the decomposition of the control function into even and odd parts. Another way to achieve symmetry in our control law is by designing a symmetric function $h(x)$ as the input to the first layer of the neural network H_d^θ. To demonstrate this, consider a simple two-layer neural network whose weights, biases, and activation function of the i^{th} layer denoted by W_i, b_i, and σ_i, respectively:

$$H_d^\theta(q) = W_2\,\sigma_1(W_1 h(q) + b_2) + b_2.$$

The gradient of H_d^θ with respect to the input x is

$$\nabla_x H_d^\theta(x) = W_2\,\sigma_1'(W_1 h(x) + b_2) \odot W_1 \odot h'(x),$$

where $\odot$ denotes the Hadamard (element-wise) product operator. 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 policy $u^\theta(x)$.

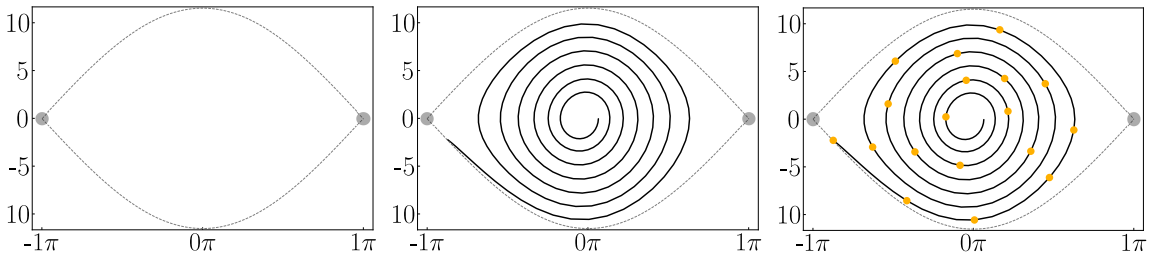
3.2 Solving the Optimization Problem

The optimization problem (3.1) reduces the infinite-dimensional search for a suitable energy-like function H_d to a finite-dimensional search for a set of parameters θ for the neural network surrogates H_d^θ and the damping gain matrix K_v^θ. In this subsection, we develop an algorithm leveraging a combination of methods from machine learning and optimization in order to efficiently reach a satisfactory solution.

The objective function of (3.1) depends on the initial state $x_0 \in \mathcal{X}$ through the trajectory γ. We are interested in obtaining a controller that performs well for a known distribution of initial states rather than for a single point. Viewing x_0 as a random variable over the state space $\mathcal{X}$, the objective function of (3.1) may be expressed as the expectation of the loss over the known distribution of x_0:

$$J(\theta) = \mathbb{E}_{x_0 \sim p(\boldsymbol{x}_0)}\Big[\ell\left(\phi(t, x_0), u^\theta\right)\Big]. \tag{3.5}$$

In this subsection, we describe a strategy that samples $\mathcal{X}$ efficiently.



(a) Desired set $\mathcal{S}$ and orbit $\gamma^\star$ (b) Execute current u^θ (c) Sample from trajectory

Figure 3.1: The sampling strategy used to train NeuralPbc, demonstrated on the phase space of a simple pendulum. (a) The desired set $\mathcal{S}$ containing $x^\star$ is defined, along with any potential preferred orbit $\gamma^\star$. (b) The system is simulated forward under the control law u^θ with current knowledge of the parameters θ. (c) Sample points along the resultant trajectory from previous step, and use them as initial conditions to generate trajectories $\{\gamma\}$ to train from.

Instead of randomly sampling the entire space, we adopt the key ideas from DAgger [48]. Let $\mathcal{D}$ denote the training dataset of size $N_\mathcal{D}$. This is a collection of initial states $\{x_0\}$ from which the collection of trajectories $\{\gamma\}$ will be generated and used to compute the loss. DAgger populates $\mathcal{D}$ with points along the trajectories generated by executing the control law u^θ with the current set of parameters θ. This captures the evolution of the system under the application of u^θ and iteratively collects new samples from the regions of the state-space visited by the learned control policy. This process is demonstrated on the phase space of the simple pendulum, shown in Figure 3.1.

Initially, u^θ is poorly trained, and $\mathcal{D}$ may be dominated by points that are far away from the goal set $\mathcal{S}$. In high-dimensional state spaces with complicated dynamics, it may be difficult to recover from locally optimal solutions if $\{\gamma\}$ predominantly misses the goal set $\mathcal{S}$ by a large amount. To circumvent this problem, the training algorithm randomly alternates between populating $\mathcal{D}$ using (a) the DAgger strategy, and (b) sampling from a normal distribution centered at $x^\star$, i.e. $x_0 \sim \mathcal{N}(x^\star, \Sigma)$, where the

Algorithm 1 Solution to Nonlinear Program (3.1)

procedure NEURALPBC(H, G, θ)
$f \leftarrow f(H, G)$ dynamics given by ODE (2.9) with $u = u^\theta$
$\mathcal{D} \leftarrow \{x_0\}_{(N_\mathcal{D})}$ ▷ $N_\mathcal{D}$ samples of x_0 (DAGGER)
while $i <$ `maxiters` **do**
  for each `batch` $\subset \mathcal{D}$ **do**
    $J \leftarrow 0$ ▷ Batch loss
    for each $x_0 \in$ `batch` **do**
      $\gamma \leftarrow$ integrate f from x_0 with current θ
      $J \leftarrow J + \ell(\gamma; \theta)$ ▷ Eq. (3.2)
    $\theta \leftarrow \theta + \alpha_i \partial J / \partial \theta$ ▷ SGD step
  $\mathcal{D} \leftarrow \{\mathcal{D}\}_{(1,\ldots,N_\mathcal{D}-N_\mathrm{R})} \cup \{x_0\}_{(N_\mathrm{R})}$ ▷ Replay buffer
  $i \leftarrow i + 1$
return θ

variance Σ is kept small. The likelihood of populating $\mathcal{D}$ by DAGGER is gradually increased as training progresses. This kickstarts the algorithm with a dataset that has several initial states already close to the goal set $\mathcal{S}$. Finally, after each iteration through the entire dataset, we substitute N_R points in $\mathcal{D}$ with new samples, keeping an $N_\mathcal{D} - N_\mathrm{R}$ portion of initial states as the *replay buffer*, a common technique used in reinforcement learning algorithms [49]. Algorithm 1 provides a summary of the NEURALPBC training process.

3.3 Conclusion

This chapter introduces NEURALPBC, wherein a passivity-based control law is synthesized by training a neural network that serves as a surrogate for the storage function H_d of a passive system. We formulate a machine learning framework that automatically finds a suitable set of parameters of the controller, which is governed by the neural network gradients. The training algorithm uses the closed-loop trajectories

as data, and performs backpropagation through the ODE solutions using the help of adjoint sensitivity analysis and automatic differentiation. The training process is paired with a carefully designed sampling strategy that efficiently covers the relevant portion of the state space without incurring too much sample complexity.

In Chapter 5, we study the efficacy of NeuralPbc through a series of simulated and real-world experiments commonly used as benchmark control problems. The experiments demonstrate that the proposed framework enables a feasible way to simultaneously employ Pbc techniques and incorporate performance considerations into the control design. Further, the performance of our controllers remains satisfactory in hardware implementation, empirically demonstrating robustness properties against model uncertainties and measurement noise.

CHAPTER 4

NEURAL INTERCONNECTION AND DAMPING ASSIGNMENT PASSIVITY-BASED CONTROL

One of the primary goals of this dissertation is to bridge the gap between controllers derived from learning algorithms and rigorous stability analysis from nonlinear control theory. An astute reader may recognize that while the NeuralPbc framework enables a flexible way to design controllers, it is not immediately clear how rigorous stability analysis may be performed when the energy-like (storage) function H_d is chosen as a dense neural network. In this chapter, we make further progress toward this goal by leveraging the IdaPbc method and formulate an optimization problem that restricts H_d to be of a more parsimonious form that facilitates stability analysis.

We introduce NeuralIdaPbc, a systematic method that alleviates the burden of solving the matching PDEs (2.19) in closed-form by finding an approximate solution that respects the structure and constraints imposed by the original IdaPbc framework. In particular, we leverage Proposition 2.2.8 and use the fact that when the choice of H_d satisfies equation (2.19), the controller given by (2.10) transforms the dynamics into the port-controlled Hamiltonian form (2.14), whose stability results are well established [5]. Our main interest lies in obtaining a solution to the matching PDEs (2.19) through an optimization problem without breaking the intended structure of the closed-loop system. In the subsequent sections, we describe the formulation

of the optimization problem and how the relevant constraints are imposed.

4.1 Problem Statement

Control synthesis in the IDAPBC paradigm can be formulated as the following feasibility problem that searches for a suitable closed-loop Hamiltonian H_d^θ:

$$\begin{aligned} \underset{M_d,\, J_2,\, V_d}{\text{minimize}} \quad & 0, \\ \text{subject to} \quad & 0 = G^\perp \left(\nabla_q H - M_d M^{-1} \nabla_q H_d + J_2 M_d^{-1} p \right), \\ & H_d = \frac{1}{2} p^\top M_d^{-1}(q) p + V_d(q), \\ & M_d = M_d^\top \succ 0, \\ & J_2 = -J_2^\top, \\ & q^\star = \underset{q}{\operatorname{argmin}}\ V_d. \end{aligned} \tag{4.1}$$

This is an infinite-dimensional, nonlinear optimization problem that is intractable to solve in general. We therefore seek to reduce the problem to a finite-dimensional one by the means of approximation by neural networks.

To this end, we proceed by representing the candidate solutions (surrogates) $M_d(q)$, $J_2(q,p)$, and $V_d(q)$ by function approximators. We define $M_d^{\theta_m} : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ and $J_2^{\theta_j} : \mathbb{R}^{2n} \to \mathbb{R}^{2n\times 2n}$ using fully-connected neural networks, where $\theta_m \in \mathbb{R}^{n_m}$ and $\theta_j \in \mathbb{R}^{n_j}$ each denote the corresponding neural network parameters. The surrogate $V_d^{\theta_v} : \mathbb{R}^n \to \mathbb{R}$ to the closed-loop potential energy needs to satisfy the condition given by (2.16). We elaborate on a suitable choice for this function approximator in Section 4.1.2.

Let $\theta := (\theta_m, \theta_j, \theta_v) \in \mathbb{R}^{n_\theta}$ with $n_\theta = n_m + n_j + n_v$. For compactness, we drop the

notation m, j, v from the function approximators and shall refer to them as $M_d^\theta, J_2^\theta, V_d^\theta$ henceforth. The loss function $l_\theta(x) = l_\theta(q,p)$ is defined as

$$l_\theta(x) = \left\| G^\perp \left(\nabla_q H - M_d^\theta M^{-1} \nabla_q H_d^\theta + J_2^\theta M_d^{\theta^{-1}} p \right) \right\|^2, \tag{4.2}$$

where $H_d^\theta(q,p) = \frac{1}{2} p^\top \left(M_d^\theta\right)^{-1} p + V_d^\theta(q)$. We denote by u^θ the resulting control law obtained by substituting H_d^θ into equation (2.10). The proposed framework aims to approximate the solution to (4.1) by finding the parameters θ such that l_θ is minimized over an appropriate region $\Omega \subset \mathcal{X}$ of the state space. We arrive at the following finite-dimensional optimization problem:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \qquad & J = \sum_{x \in \Omega} l_\theta(x), \\ \text{subject to} \qquad & M_d^\theta = \left(M_d^\theta\right)^\top \succ 0, \\ & J_2^\theta = -\left(J_2^\theta\right)^\top, \\ & q^\star = \underset{q}{\operatorname{argmin}}\ V_d^\theta. \end{aligned} \tag{4.3}$$

The first two constraints preserve the port-controlled Hamiltonian structure given by equation (2.14) in the closed-loop system, as originally intended in IdaPbc. The last two constraints enforce the arg min condition given by (2.16).

4.1.1 Stability Analysis

By construction of (4.3), as $J \to 0$, the solution H_d^θ of the optimization problem (4.3) converges to a Lyapunov function of the closed-loop system. This allows us to make a connection between stability and controllers derived from learning algorithms, using the well-known fact that solutions of ordinary differential equations depend

continuously on their parameters.

Proposition 4.1.1. *Suppose the optimization problem* (4.3) *has an optimal solution at* $\theta^\star$ *with the optimal value* $J^\star = 0$. *The control law* u^θ *is a continuous function of the objective value* J *and the solution* θ *of the optimization problem.*

Proof. Let $\mathbb{U} = \mathbb{U}(\theta)$ denote the right side of equation (2.18). In this notation, the objective function of (4.3) is the squared norm of $G^\perp\mathbb{U}$. Let $(\cdot)^{(k)}$ denote the k^{th} iteration of the optimization algorithm. Since $J^{(k)} \to 0$, $\mathbb{U}^{(k)} \to \mathbb{U}^\star$, and $\theta^{(k)} \to \theta^\star$, there exists an integer $K > 0$ such that when $k > K$, the following are true:

1. $0 \leq J^{(k)} < \delta_1$,
2. $0 \leq \left\|\mathbb{U}^{(k)} - \mathbb{U}^\star\right\| < \delta_2$,
3. $0 \leq \left\|\theta^{(k)} - \theta^\star\right\| < \delta_3$.

Since $G^\dagger(q) = \left(G^\top(q)G(q)\right)^{-1} G^\top(q)$ is a continuous function of q, and $u^\theta_{es} = G^\dagger\mathbb{U}$ is linear in $\mathbb{U}$, it follows that for all $\epsilon > 0$, $\exists\delta_2 > 0$ such that $\left\|G^\dagger\mathbb{U}^{(k)} - G^\dagger\mathbb{U}^\star\right\| < \epsilon$ whenever $\left\|\mathbb{U}^{(k)} - \mathbb{U}^\star\right\| < \delta_2$. The claim of the proposition is demonstrated by noting that $u^\theta_{es} = G^\dagger\mathbb{U}$ and $u^{\theta^\star}_{es} = G^\dagger\mathbb{U}^\star$. □

By Proposition 2.2.8, the control law u^θ with the optimal parameters $\theta = \theta^\star$ asymptotically stabilizes the desired equilibrium $x^\star = (q^\star, 0)$. Since u^θ is continuous on J, θ, and the solution of (2.9) is continuous with respect to the control input, we deduce that for $\|\theta - \theta^\star\|$ sufficiently small, the control law u^θ will still steer the trajectories to pass through a small neighborhood of $x^\star$, at which point standard linear control techniques may be employed to achieve asymptotic stability.

Proposition 4.1.2. *The Hamiltonian system* (2.9) *enters a neighborhood of* $(q^\star, 0)$ *upon the application of* u^θ *as long as the optimal value of the optimization problem* (4.3) *is sufficiently small.*

Proof. Let $\theta^\star$ denote an optimal solution of the problem (4.3) so that $G^\perp \mathbb{U}^{\theta^\star} = 0$, and let the corresponding control law be denoted by $u^{\theta^\star}$. By Proposition 2.2.8, the control law $u^{\theta^\star}$ asymptotically stabilizes $x^\star = (q^\star, 0)$. By Proposition 4.1.1, we know that u^θ is a continuous function of the optimal value J. It is well-known that the solution of the dynamical system (2.9) is a continuous function of u, hence it is also a continuous function of the parameters θ [50].

Combining these continuity results, we conclude that there exists a time horizon $T > 0$ such that the flow $\phi\left(t; u^\theta(x)\right)$ of the ODE (2.9) under the application of u^θ satisfies $\phi(T) \in B_r(x^\star)$, where $B_r(x)$ denotes a ball of radius r around x. In this context, the radius r is a function of the tolerance of the optimization algorithm. □

These results imply that the control law derived from the optimal solution of (4.3) is guaranteed to stabilize the desired equilibrium $x^\star$, whenever it is combined with a linear controller designed to stabilize (2.9) in a neighborhood of $x^\star$.

Proposition 4.1.3. *The optimal control law* u^θ *can be combined with a linear stabilizing controller* $\hat{u}$*, such as the Linear Quadratic Regulator (LQR), in order to asymptotically stabilize system* (2.9) *at* $x^\star$*.*

Proof. In Proposition 4.1.2 we have shown that the closed-loop trajectories of (2.9) passes through a neighborhood $B_r(x^\star)$, and r is a continuous function of the optimization precision. Suppose the region of attraction of the linear control law $\hat{u}(x)$ contains $B_{\hat{r}}(x^\star)$. Choose δ_2 sufficiently small such that $\exists T > 0$ with $\phi(T) \in B_{\hat{r}}(x^\star)$.

This guarantees that the trajectories of (2.9) asymptotically converge to $x^\star$ as $t \to \infty$ under the application of u^θ whenever $x \notin B_{\hat{r}}(x^\star)$ and $\hat{u}$ whenever $x \in B_{\hat{r}}(x^\star)$. □

4.1.2 Constraints

In the subsequent sections, we elaborate on how the constraints in the optimization problem (4.3) are imposed. In the remainder of this chapter, we let $\mathbb{S}_n$ denote the set of symmetric $n \times n$ matrices, $\mathbb{S}_n^+$ the set of positive semidefinite $n \times n$ matrices, and $\mathbb{S}_n^{++}$ the set of positive definite $n \times n$ matrices.

Positive-Definiteness of M_d^θ and Skew-Symmetry of J_2^θ

We leverage the Cholesky decomposition to express M_d^θ in the form

$$M_d^\theta(q) = L_\theta(q) L_\theta^\top(q) + \epsilon_M I_n, \tag{4.4}$$

where $\epsilon_M > 0$ is a small constant, I_n is an $n \times n$ identity matrix, and $L_\theta \in \mathbb{R}^{n \times n}$ is a lower-triangular matrix whose $n(n+1)/2$ entries are outputs of a neural network.

The skew symmetric J_2^θ is constructed by taking a square matrix A_θ, whose entries are outputs of a dense neural network, and computing

$$J_2^\theta(q,p) = A_\theta(q,p) - A_\theta^\top(q,p). \tag{4.5}$$

Positivity of V_d^θ with an Isolated Minimum at $q^\star$

This section describes the construction of the surrogate for the closed-loop potential energy V_d^θ such that the condition (2.16) is satisfied. If V_d^θ is convex, only first-order

derivative information is sufficient to guarantee the condition (2.16). To this end, we propose to approximate $V_d(q)$ with a sum-of-squares (SoS) polynomial.

Let $V_d^{\theta_v} : \mathbb{R}^n \to \mathbb{R}$ denote a SoS polynomial of degree $2d$, with $\theta_{n_v} \in \mathbb{R}^{n_v}$ representing the polynomial coefficients. The advantages of parametrizing V_d^θ with a SoS polynomial are as follows. First, it is easy to ensure the existence of a lower bound. If $V_d^\theta(q)$ is SoS, then $V_d^\theta(q) \geq 0,\ \forall q \in \mathbb{R}^n$. If the constant term is zero, then $V_d^\theta(q) = 0 \iff x = 0$. This allows the constraint (2.16) to be imposed without loss of generality by shifting the coordinate of q and aligning $q^\star$ with the origin. Moreover, any SoS polynomial can be parametrized in terms of a positive semidefinite matrix; hence, the search for a SoS polynomial is computationally efficient.

Following Theorem 2.5.2, we construct V_d^θ in terms of a positive definite matrix using the same Cholesky decomposition as (4.4). That is, with R_θ an $n \times n$ lower triangular matrix with constant entries, we define

$$V_d^\theta(q) = \mu^\top(q) R_\theta R_\theta^\top \mu(q), \tag{4.6}$$

where μ is the vector of monomials $\mu(q) = \begin{bmatrix} q_1 & \dots & q_n & q_1 q_2 & \dots & q_n^d \end{bmatrix}^\top$. The constant monomial is excluded from $\mu(q)$ to place the minimum of V_d^θ is at the origin.

For robotic systems with high-dimensional state space and complicated dynamics, the expressive power of V_d^θ may not be sufficient if the degree d of the polynomial is kept low. We combat this concern by introducing an option of choosing the desired potential function V_d^θ as a neural network whose output is a scalar. In this setting, the condition (2.16) can be achieved by using a neural network architecture with no bias. With W_i denoting the weights of the i^{th} layer, σ denoting the activation function in each layer, and ϕ a differentiable function, we have

$$V_d^\theta(q) = \phi\left(W_i \sigma\left(W_{i-1}\sigma\left(\dots W_2 \sigma(W_1 x)\right)\right)\right). \tag{4.7}$$

V_d^θ satisfies (2.16) as long as $\sigma(0) \equiv 0$; $\phi(0) \equiv 0$; and $\phi(z) > 0$, $z \neq 0$. Common choices for the activation function σ satisfying these requirements include, e.g. ReLU and Elu [51].

4.1.3 Reducing the Sample Space

In this subsection, we show how the loss function (4.2) can be expressed in a more parsimonious form, which only depends on the configuration variable q instead of the state $x = (q, p)$. This halves the sample complexity of the training algorithm. The main PDE (2.19) can be separated in terms that depend on p and terms are independent of p:

$$G^\perp\left\{\nabla_q\left(p^\top M^{-1} p\right) - M_d M^{-1}\nabla_q\left(p^\top M_d^{-1} p\right) + 2J_2 M_d^{-1} p\right\} = 0, \tag{4.8}$$

$$G^\perp\left\{\nabla_q V - M_d M^{-1}\nabla_q V_d\right\} = 0. \tag{4.9}$$

Following [8], we use the fact that

$$\nabla_q\left(z^\top A(q) z\right) = \left[\nabla_q\left(A(q)z\right)\right]^\top z,\ \forall z \in \mathbb{R}^n,\ \forall A \in \mathbb{S}_n,$$

to write the PDE constraint (4.8) as

$$G^\perp\left\{\left[\left[\nabla_q\left(M^{-1}p\right)\right]^\top - M_d M^{-1}\left[\nabla_q\left(M_d^{-1}p\right)\right]^\top + 2J_2 M_d^{-1}\right] p\right\} = 0.$$

The identity $\nabla_q\left(A(q)z\right) = \sum_{k=1}^n \nabla_q\left(A_{(\cdot,k)}\right) z_k$, where $A_{(\cdot,k)}$ denotes the k^{th} column of the matrix A, holds for all $z \in \mathbb{R}^n$ and all $A \in \mathbb{R}^{n\times n}$. We use this identity to

reparameterize J_2^θ in terms of the matrices $U_k^\theta(q) = \left(-U_k^\theta(q)\right)^\top \in \mathbb{R}^{n\times n}$ as

$$2J_2^\theta = \sum_{k=1}^{n} U_k^\theta p_k.$$

We can now express (4.2) only in terms of q as $l_\theta = (\sum_{k=1}^n l_{1k,\theta}) + l_{2,\theta}$, where

$$l_{1k,\theta}(q) = \left\| G^\perp \left\{ \left[\nabla_q \left(M_{(\cdot,k)}^{-1} \right) \right]^\top - M_d^\theta M^{-1} \left[\nabla_q \left(M_d^\theta \right)_{(\cdot,k)}^{-1} \right]^\top + U_k^\theta \left(M_d^\theta \right)^{-1} \right\} \right\|^2, \quad (4.10)$$

$$l_{2,\theta}(q) = \left\| G^\perp \left\{ \nabla_q V - M_d^\theta M^{-1} \nabla_q V_d^\theta \right\} \right\|^2. \quad (4.11)$$

Note that (4.10) is only dependent of M_d^θ and U_k^θ. This implies that the problem (4.3) can be solved in two stages, as elaborated in the following subsection.

4.1.4 Solving the Optimization Problem

We split the problem (4.3) into two subproblems, the first of which is

$$\begin{aligned} \underset{\theta}{\text{minimize}} \qquad & J_1 = \sum_{q\in\mathcal{Q}} \left(\sum_{k=1}^{n} l_{1k,\theta}(q) \right), \\ \text{subject to} \qquad & M_d^\theta = \left(M_d^\theta \right)^\top \succ 0, \\ & U_k^\theta = -\left(U_k^\theta \right)^\top, \quad k = 1, \ldots, n. \end{aligned} \quad (4.12)$$

To solve this optimization problem, we employ the reverse-mode automatic differentiation (AD) to obtain the appropriate gradients for use with gradient-based search algorithms, such as ADAM [52]. For differentiation of scalar functions, reverse-mode AD performs fewer computations than forward-mode AD, at the expense of memory

consumption [53]. In our application where neural net architectures are small, the reduced computation significantly outweighs the memory usage trade off.

Given the solutions M_d^θ, U_k^θ to (4.12), the following optimization problem can be solved to obtain the coefficients of V_d^θ:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & J_2 = \sum_{q \in \mathcal{Q}} l_{2,\theta}(q), \\ \text{subject to} \quad & V_d^\theta(q) \text{ is SoS}, \\ & V_d^\theta(q^\star) = 0. \end{aligned} \tag{4.13}$$

When V_d^θ is an SoS polynomial, the gradient of (4.6) can be analytically obtained, and the problem (4.13) can be re-formulated as the least squares problem: $\min \|Ax - b\|^2$. This problem is then solved in a single step, significantly increasing the computation speed of our algorithm.

The computational complexity of our optimization problem increases linearly as the degree of freedom of the dynamics increases. Similarly, however, as the model complexity increases, so does the difficulty in obtaining the closed-form solutions of the PDEs in the original IDAPBC framework. Our framework provides an alternative, and arguably a more practically applicable methodology, to apply IDAPBC to more realistic, complicated systems.

4.2 Conclusion

In this chapter, we have introduced a learning framework that automatically finds a stabilizing controller for a class of *underactuated* mechanical systems. First, we construct an optimization problem that leverages the IDAPBC methodology without destroying the passive structure and other physical constraints. By the approxima-

tion capability of neural networks, we solve the optimization problem that yields approximate solutions of the nonlinear PDEs relevant to the IDAPBC problem. We show that as the optimization converges to a faithful solution of the matching PDEs, the corresponding control law is guaranteed to steer the trajectories to pass through a neighborhood of the desired equilibrium, at which point standard linear control techniques may be employed to achieve asymptotic stability.

In Chapter 5, we apply this framework to the two benchmark problems presented in the original IDAPBC paper [8]. With relatively minimal computation, the controllers found by our learning framework successfully stabilize the desired equilibria of these systems. Current research is under way to rigorously improve the robustness of these controllers by accounting for model uncertainties during training.

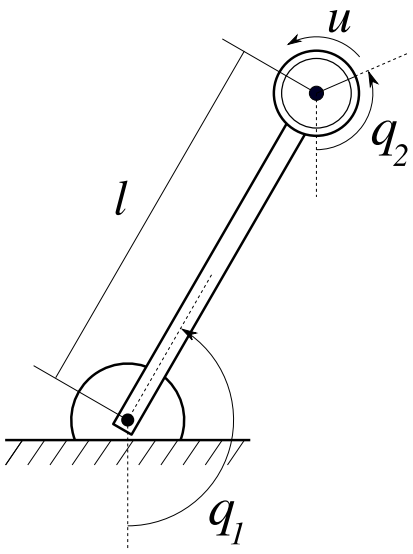
CHAPTER 5

EXPERIMENTS

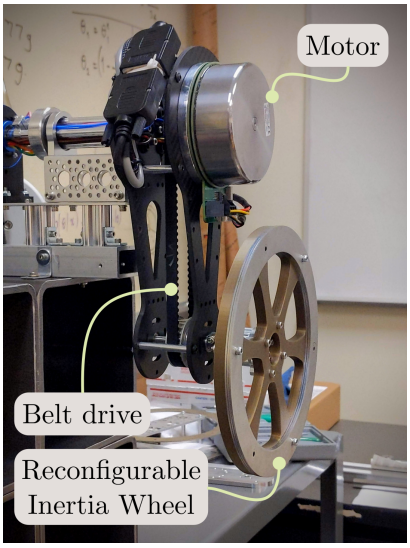
This chapter presents the results from a series of experiments that has been performed to study of the efficacy of NeuralPbc and NeuralIdaPbc, as well as to demonstrate their applicability on a wide range of underactuated robotic systems. Section 5.1 documents the NeuralPbc and NeuralIdaPbc experiments performed on the inertia wheel pendulum. Our custom hardware implementation for the IWP is described in detail therein. Experimental NeuralPbc results on the cart-pole system are presented in Section 5.2, and on the Acrobot system in Section 5.3. Finally, simulation experiments for NeuralIdaPbc are performed on the ball-beam system in Section 5.3.

5.1 Inertia Wheel Pendulum

The inertia wheel pendulum (IWP) is a simple planar pendulum that consists of an actuated wheel at the end of the arm instead of a static mass. The joint connecting the pendulum to the ground has no actuation. The wheel has mass m, and is connected to a massless rod of length l. The position of the rod is denoted by the angle q_1, and the position of the wheel is denoted by q_2. Figure 5.1 depicts the IWP system and the convention of the configuration variables q_1 and q_2.



(a) Schematic



(b) Hardware implementation

Figure 5.1: (a) Schematic of the IWP system. The joint q_1 is not actuated. (b) Hardware implementation. The joint q_2 is actuated by a belt-drive system with a motor mounted along the axis of rotation of q_1 to minimize the moment arm length.

The control task studied for the IWP is the swing-up task, in which the system must use the torque exerted on the inertia wheel to move the system into a vertical configuration then balance. This system is representative of the primary challenge in the control of underactuated robots. In order to perform the swing up, the control law must reason about the energetic coupling between the actuated (q_2) and unactuated (q_1) degrees of freedom.

5.1.1 System Model

The Hamiltonian of the IWP is given by Eq. (2.8) with $n = 2$ and

$$H(q,p) = \frac{1}{2}p^\top M^{-1}p + V(q) = \frac{1}{2}p^\top \begin{bmatrix} I_1 & 0 \\ 0 & I_2 \end{bmatrix}^{-1} p + mgl\left(\cos(q_1) - 1\right),$$

where $p = (I_1\dot{q}_1, I_2\dot{q}_2)$. The input matrix for this system is $G = \begin{bmatrix} -1 & 1 \end{bmatrix}^\top$. The state of the system is $x = (q_1, q_2, \dot{q}_1, \dot{q}_2)$. The parameter I_1 denotes the moment of inertia of the pendulum, I_2 is the moment of inertia of the rotating wheel, g is the gravitational constant, and l is the length between q_1 and q_2.

The equations of motion of the IWP can be written in standard form as

$$\begin{bmatrix} I_1 & 0 \\ 0 & I_2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -mgl \sin q_1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} u, \tag{5.1}$$

where the control input u is the torque applied to the inertia wheel. The maximum torque produced by the motor is limited to $|u| \leq u_{\max} = 0.4$ N-m, which is insufficient to overcome the gravity term $mgl = 1.795$ N-m. The desired equilibrium of this system is $x^\star = (q^\star, 0) = 0$, which corresponds to the upright position. A summary of the system parameters is provided in Table 5.1

5.1.2 Hardware Implementation

This subsection describes the custom design of the inertia wheel pendulum system. The first link l is a pair of laser cut carbon fiber sheets separated by standoffs, providing the necessary rigidity for mounting heavy components. The link is supported by the ground through a series of low-friction ball bearings. The position q_1 of the first link is measured using an `Autonics E40` encoder with a resolution of 4096 pulses per revolution in quadrature mode. The velocity $\dot{q}_1$ is approximated by computing the backward difference of the position divided by the sample period.

The joint q_2 connects the second link to the first link through radial ball bearings. This joint is actuated by a `Nanotec DFA90` brushless DC motor through a belt drive

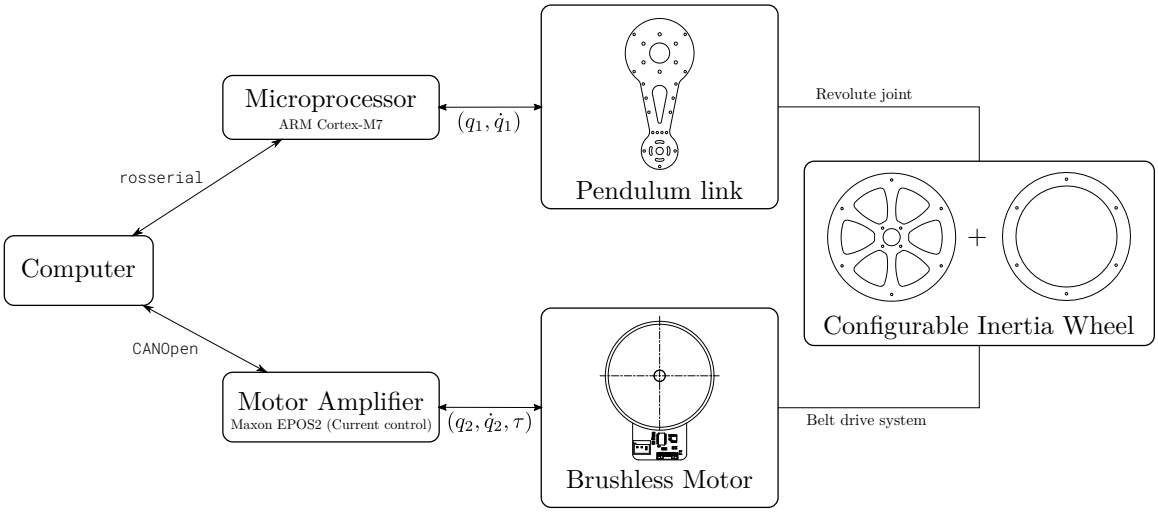


Figure 5.2: Custom hardware implementation of the IWP. The design uses a customizable inertia wheel for the purpose of testing robustness of control policies.

system. The motor is mounted concentrically with the joint q_1 to minimize mgl. A `Maxon EPOS2` motor amplifier powers the motor through a 40V battery, and is responsible for the control of the motor torque (current) and for the data acquisition of the q_2 joint. The link attached to q_2 is reconfigurable so that multiple combinations of system parameters are possible, enabling researchers to efficiently examine robustness properties of their controllers. Moreover, a second pendulum may be mounted in place of the inertia wheel, quickly turning the system into an Acrobot [54]. The system parameters for each possible IWP configuration are summarized in Table 5.1.

Table 5.1: System parameters of the IWP. The errors in the last column are normalized with respect to the nominal

Parameter set	I_1	I_2	mgl	Error
Nominal	0.0455	0.00425	1.795	0
Type A	0.0417	0.00330	1.577	0.122
Type B	0.0378	0.00235	1.358	0.243
Type C	0.0340	0.00141	1.140	0.365

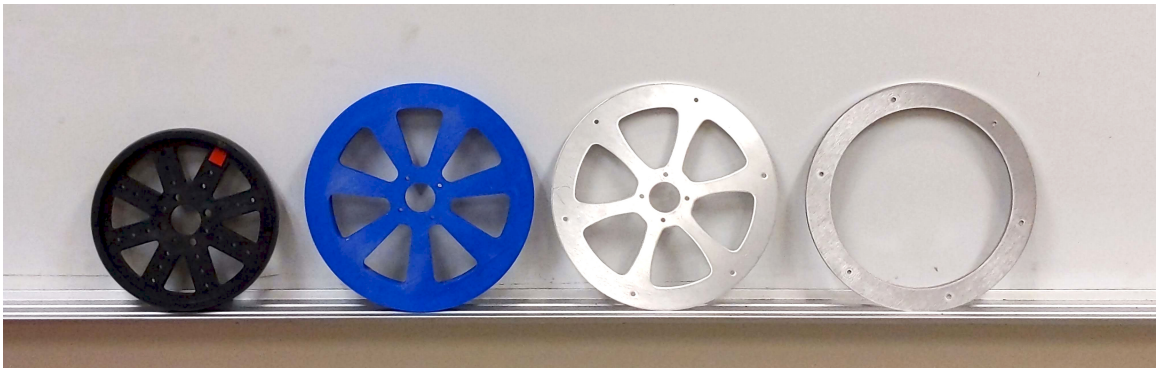


Figure 5.3: Design iterations of the inertia wheel. The final design maximizes inertia while minimizing mass, allowing large torque to be exerted on the wheel without spinning up too fast and thus minimizing the effects of the back electromotive voltage of the motor.

The inertia wheel is a multi-piece construction designed to concentrate the mass as far away from the axis of rotation as possible. The first piece is the core, which is 3D printed using polylactic acid (PLA) plastic filament with a light infill, minimizing mass. The remaining pieces are stainless steel rings, bolted to the core around the perimeter. Up to four stainless steel rings may be mounted to the core. This design minimizes mass while maximizing I_2, which allows the motor to exert high torque on the wheel without speeding it up too quickly. This avoids the saturation of the supply voltage due to the back electromotive voltage of the motor at high speed. The design iterations for the inertia wheel are shown in Figure 5.3.

The data acquisition and the digital signal processing of this system is performed on a Teensy 4.1 development board with the IMXRT1060 processor based on the ARM Cortex-M7 architecture. This development board communicates with the control computer through a serial protocol implemented in the open source library `rosserial` The communication between the computer executing the control law and the motor amplifier is established through the `CANOpen` protocol [55].

5.1.3 NeuralPbc Experiments

We apply the NeuralPbc framework to the IWP system with the energy-like function H_d^θ chosen as a fully-connected neural network with two hidden layers, each with the Elu activation function [51]. The parameters θ are initialized according to the Glorot (Xavier) [56] scheme. We apply Algorithm 1 to learn the parameters of H_d^θ and K_v^θ. In each gradient descent step, a batch of 4 initial conditions $\{x_0\}$ is integrated forward with using the Tsitouras 5(4) Runge-Kutta solver with a time horizon of $T = 3$ seconds. The cost function is then computed and back-propagated using the AD-assisted adjoint method implemented in the open source library `DiffEqFlux.jl` [35], written in the `Julia` computing language . The ADAM [52] optimizer is used to update the parameters. A replay buffer size of 400 initial conditions is used.

The control objective is to ensure that closed-loop trajectories of (5.1) enters the goal set $\mathcal{S}$ defined as

$$\mathcal{S} = \left\{ x \in \mathbb{R}^4 \, : \, \left\| x - \begin{bmatrix} 2\pi n & q_{2f} & 0 & 0 \end{bmatrix}^\top \right\| \leq 0.01, \, n \in \mathbb{Z}, \, q_{2f} \in \mathbb{R} \right\}.$$

Once the system reaches the set $\mathcal{S}$, a linear stabilizing controller can be employed for asymptotic stabilization. In particular, we apply the NeuralPbc controller in conjunction with the Linear Quadratic Regulator (LQR) for the linearization of (5.1) about $x^\star$. The LQR gains are obtained by solving the Riccati equation with the cost matrix $Q = R$ chosen as the identity matrix. The LQR controller is activated whenever the state x comes sufficiently close to the LQR's region of attraction. The learned controller is evaluated by simulating the system for 20 seconds from a range

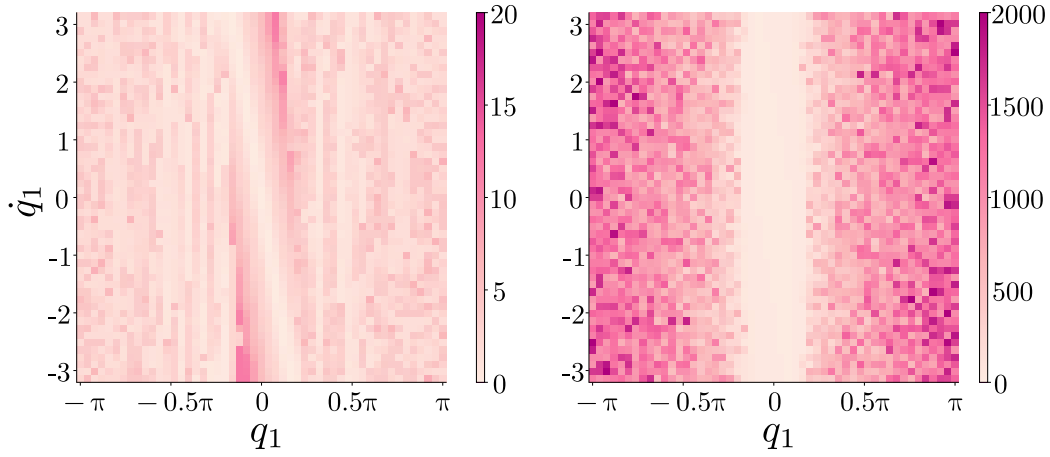


Figure 5.4: Comparison of accumulated control effort $\|u^\theta\|$ required to swing up the IWP, computed from simulated trajectories starting from each $(q_1, \dot{q}_1)$ grid. From left to right: NeuralPBC and classical IdaPbc.

of initial states with $q, \dot{q} \in [-\pi, \pi]$. The initial wheel position and velocity are zero.

The performance metric is chosen as the accumulated expenditure of control action $\|u^\theta\|$ along the trajectories of the closed-loop system. If the control law u^θ fails to bring the system inside the region of attraction of LQR, i.e., failing to stabilize $x^\star$, the performance metric is defined as ∞. Figure 5.4 shows that the expenditure of control action required to swing-up by the NeuralPbc controller is much lower (2 orders of magnitude) than that of the classical IdaPbc control law.

The robustness of our control law is examined by executing the learned controller on physical hardware, which differ from the nominal dynamical model (5.1) due to the effects of errors in the parameters, friction in the bearings, and any contribution to the dynamics from the belt-drive system. Further, we deliberately modify the hardware to introduce large errors in the model parameters and test the controllers without any additional training. In particular, the inertia wheel attached to q_2 is replaced with parts whose mass and inertia values differ from the nominal values (see Table 5.1).

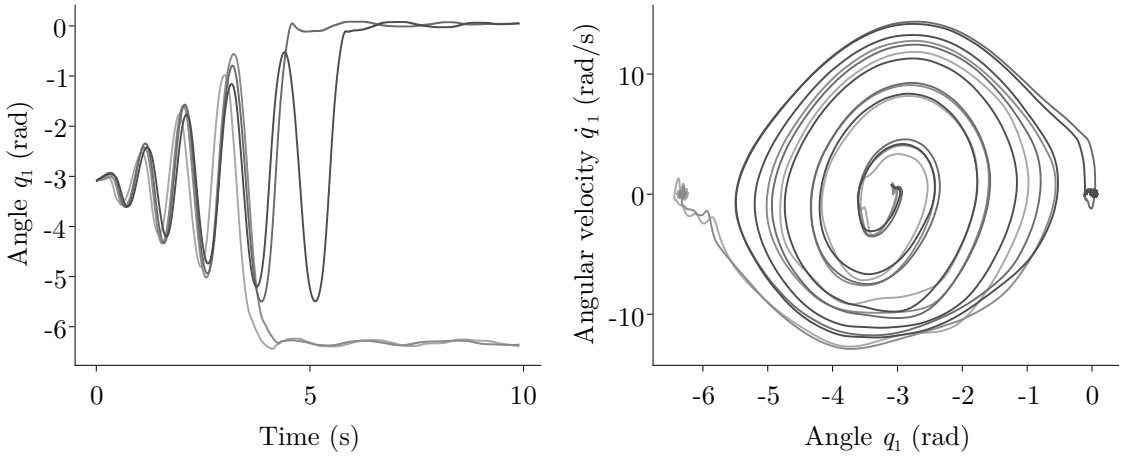


Figure 5.5: Recorded trajectories of the IWP system with modified system parameters in Table 5.1. Lines with lighter shade correspond to larger errors in the parameters. The NeuralPbc control law u^θ is trained once with the nominal parameters, but maintains the ability to bring the system to the desired equilibrium $x^\star$ even with large amount of discrepancies in the system parameters.

The experiment starts with the system at rest in the downward pose, i.e. $x = (\pi, 0, 0, 0)$. A small disturbance in the q_1 direction is introduced to start the swing-up. The state x is recorded, and a set of sample trajectories from the experiment trials are shown in Figure 5.5. In all scenarios, the controller is still able to perform the swing-up task despite the large errors introduced in the system parameters.

5.1.4 NeuralIdaPbc Experiments

In this subsection, we describe the NeuralIdaPbc experiments performed on the IWP. For this particular system with a constant mass matrix, the PDE constraint (4.8) would have been trivially satisfied if M_d^θ was a constant matrix, and $U_1^\theta = U_2^\theta = 0$. However, we demonstrate the flexibility of our approach by using the framework to automatically discover an appropriate solution, not necessarily the trivial one.

The entries of each of the matrices $M_d^\theta, U_1^\theta, U_2^\theta$ are outputs of neural networks. The

architecture of each network is summarized in Table 5.2. The closed-loop potential energy V_d^θ is a SoS polynomial of degree 4 ($d = 2$). In total, there are 1,261 parameters in θ to train.

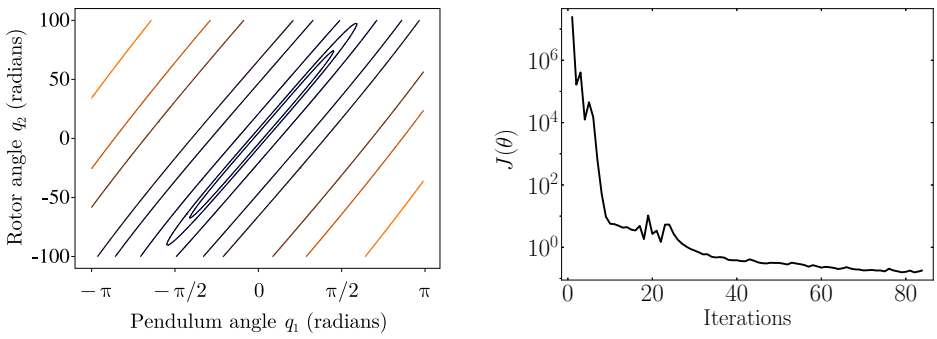
To gather the data for solving the optimization problems (4.12) and (4.13), the configuration space is sampled uniformly from $q_1, q_2 \in [-\pi, \pi]$, with a step size of 0.1. There are a total of 3,969 samples. The collection of these samples are then shuffled and organized into batches. The number of batches is a hyperparameter to be selected by the user.

For each batch, the objective function J is evaluated, and the gradient descent algorithm updates the parameters θ according to the gradient $\partial J/\partial\theta$. An epoch is completed when all samples have been processed. This process is repeated until the objective value is smaller than a user-defined threshold, or until a maximum number of epochs is reached.

Figure 5.6 shows the progress of the objective value from the training session. The loss rapidly decreases after only a few iterations. Each epoch typically takes about 10-15 seconds to perform on a machine with the Intel Core i7-10750H processor, without any parallel computation. This amount of computation is relatively small compared to a typical training session in reinforcement learning algorithms. It took 84 iterations to obtain the results shown in this subsection. These results demonstrate the computational efficiency of our approach.

Table 5.2: Neural network architectures for solving (4.12) in the IWP case study. The ordering of the layer dimensions and activations are arranged from input to output.

	Layer Dimensions	Activation Functions
M_d^θ Equation (4.4)	(2, 16, 16, 3)	(ELU, ELU, ELU, Identity)
U_1^θ Equation (4.5)	(2, 8, 8, 1)	(ELU, ELU, ELU, Identity)
U_2^θ Equation (4.5)	(2, 8, 8, 1)	(ELU, ELU, ELU, Identity)



(a) The learned energy-like function $V_d^\theta(q)$ (b) The objective value of equation (4.12) during training

Figure 5.6: (a) The energy-like function V_d^θ for the IWP after training, which has an isolated minimum at the desired equilibrium $q^\star = (0, 0)$, and (b) the loss of the PDE constraints (4.8) rapidly decreasing during training.

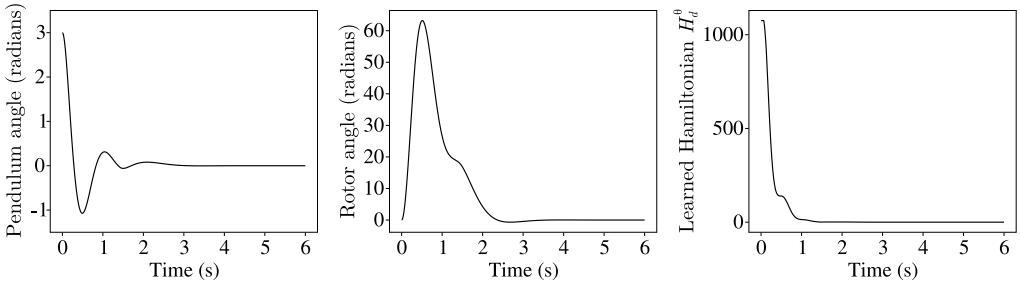


Figure 5.7: Time evolution of inertia wheel pendulum with the data-driven IDAPBC. The last plot to the right shows the learned Hamiltonian H_d^θ decreasing along the trajectory of the system.

We demonstrate the efficacy of our approach through simulated experiments. After training, the swing-up controller is derived according to equations (2.10), (2.20) and (2.17). The damping gain K_v in (2.17) is chosen as the identity matrix. A simulated trajectory executed using the learned controller is shown in Figure 5.7. For this particular simulation, the system starts at rest with the initial configuration $q(0) = (3, 0)$, i.e. the pendulum is near the downward equilibrium. The controller successfully brings the mechanism to the desired upright equilibrium. The results here suggest that approximations to the solutions of the matching PDEs in the IDAPBC

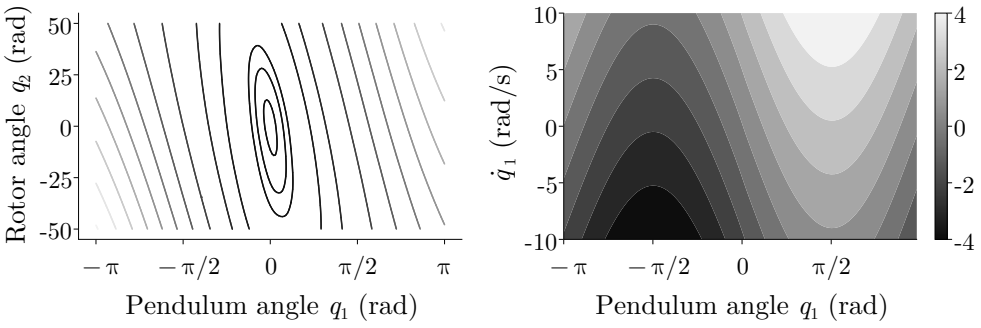


Figure 5.8: The left plot shows the level sets of the closed-loop potential energy $V_d^\theta(q)$, which has an isolated minimum at the desired equilibrium $q^\star = 0$. The right plot shows the control effort u^θ derived from H_d^θ, projected on q_1-$\dot{q}_1$ plane with $q_2 = \dot{q}_2 = 0$. The controller correctly commands torque in the appropriate direction, pushing the trajectories to $x^\star$.

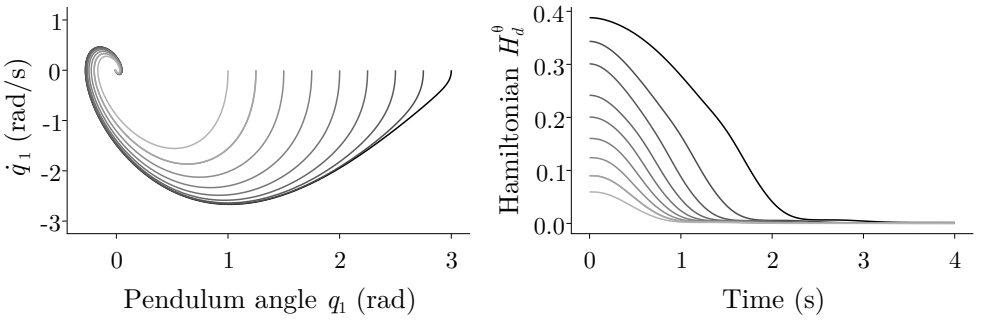


Figure 5.9: The time evolution of the system under the application of the controller derived from the learned Hamiltonian H_d^θ. The right plot shows H_d^θ decreasing along the trajectories of the closed-loop system, satisfying Lyapunov's criteria.

design methodology is an effective way to algorithmically find stabilizing controllers for underactuated mechanical systems.

We further investigate the flexibility of our framework by replacing the potential energy function V_d^θ with a fully-connected neural network given by Eq. (4.7) with $\Phi(z) = \|z\|_2^2$ and two hidden layers, each of which has the activation function ELU. The contours of the neural net V_d^θ and the corresponding control law are shown in Figure 5.8. We observe that the condition (2.16) is still satisfied with the neural net V_d^θ.

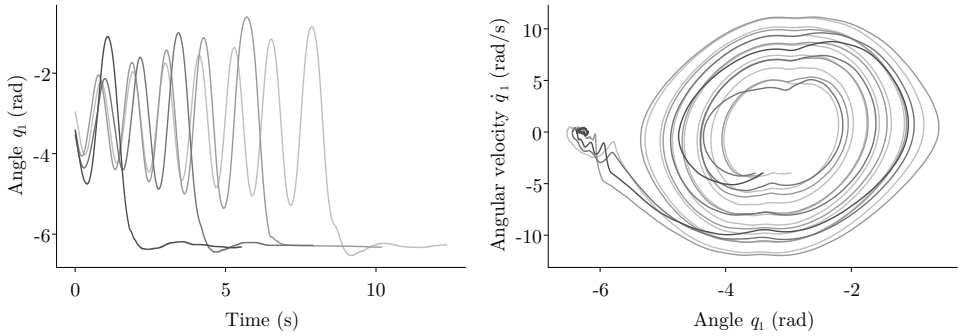


Figure 5.10: Time evolution of inertia wheel pendulum with the data-driven NeuralIdaPbc controller.

The ability to switch between a neural network and SoS polynomials in our framework offers an additional degree of flexibility for control practitioners. Simulation studies for this choice of V_d^θ are shown in Figure 5.9. The trajectories converge to the desired equilibrium and H_d^θ remains a faithful Lyapunov function of the closed-loop system.

We repeat the same real-world experiments performed for NeuralPbc to test the NeuralIdaPbc control law. We run the experiments with the four configuration of system parameters in Table 5.1 to test to robustness of our controller. Figure 5.10 shows four trajectories corresponding to each of the system configurations. We note that the trajectories recorded from hardware do not match the simulated trajectories in Figure 5.9, as the allowable torque is insufficient to overcome the gravity term mgl. However, the controller is still able to add enough energy to swing-up the IWP such that LQR successfully stabilizes $x^\star$.

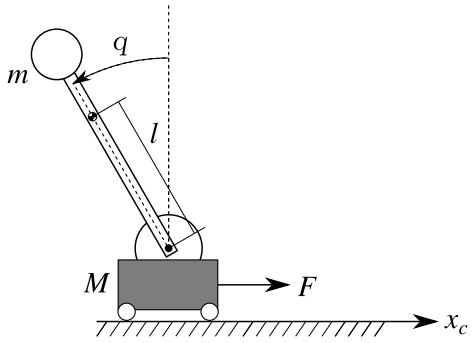


Figure 5.11: Schematic of the Cart-Pole system. The origin of the cart position x is at the center of the track. Cart position is constrained to between -0.4 and 0.4 meters.

5.2 Cart-Pole

The cart-pole system consists of an unactuated simple pendulum, mounted via a revolute joint to a linear cart that is actuated. A schematic of this system is shown in Figure 5.11. The position of the pendulum is denoted by the angle q between the centerline of the arm and its upright position. The center of mass (CoM) of the pendulum is located at a distance l from the center of the revolute joint. The position of the cart is denoted by x_c, and is constrained to remain within the interval $(-0.4, 0.4)$ m. Both the revolute joint and the cart is modeled with viscous friction with coefficients B_p and B_{eq}, respectively.

5.2.1 System Model

With the state $x = (x_c, \dot{x}_c, q, \dot{q})$, the equations of motion of the cart-pole system are

$$\begin{aligned} I_2\ddot{x}_c &= I_1\left(F_c - B_{eq}\dot{x}_c - ml\dot{q}^2 s_q\right) + ml\left(mglc_q s_q - B_p\dot{q}c_q\right), \\ I_2\ddot{q} &= (M+m)\left(mgls_q - B_p\dot{q}\right) + ml\left(F_c c_q - B_{eq}\dot{x}_c c_q - ml\dot{q}c_q s_q,\right) \end{aligned} \tag{5.2}$$

Table 5.3: Physical parameters for the cart-pole system

Parameter	Symbol	Value	Units
Cart mass	M	0.37	kg
Pendulum mass	m	0.23	kg
Pendulum length	L	0.60	m
Length to CoM	l	0.3302	m
Pendulum inertia	I	7.884E-03	kg-m^2
Cart friction coeff.	B_p	4.3	Nm-s
Pivot friction coeff.	B_{eq}	0.0024	Nm-s/rad
Max cart force	$F_{\max}$	5.3606	N

where s_q and c_q denote, respectively, sine and cosine of q; I is the inertia of the pendulum; $I_1 = I + ml^2$; $I_2 = (M+m)I + ml^2\,(M + m\sin^2 q)$; F_c is the actuator force acting on the cart; and g is the gravitational constant. The force is limited by $|F_c| \leq F_{\max}$. The model parameters are provided in Table 5.3.

Hardware Implementation

The hardware experiments for this system are performed on the Quanser Linear Servo Unit with Inverted Pendulum. A detailed description of the system may be found in the user manual published by manufacturer [57].

The NeuralPbc controller, which requires the computation of neural network gradient $\nabla_x H_d^\theta$ with respect to the input, is implemented in `Matlab`. The controller is connected to the hardware through the manufacturer provided `Simulink` program.

5.2.2 NeuralPbc Experiments

The control objective is to swing up the pendulum by moving the cart left and right without falling off the track, i.e. $|x_c| \leq x_{\max} = 0.4$ m must be satisfied. The goal set

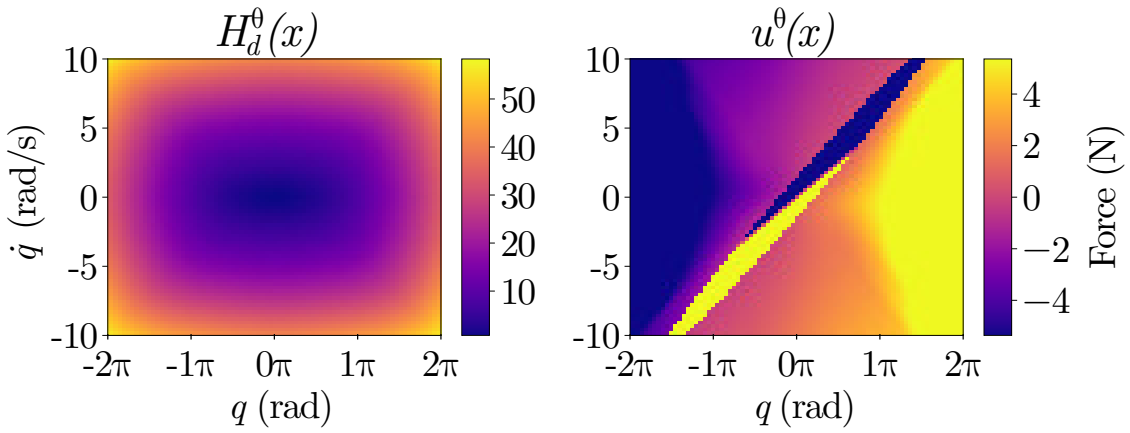


Figure 5.12: (Left) The learned energy-like function $H_d^\theta(x)$, and (right) the corresponding control policy $u^\theta(x)$ for the cart-pole system, projected onto the q-$\dot{q}$ plane with $x_c = \dot{x}_c = 0$. The function H_d^θ attains a minimum at $x^\star$ mimicking the behavior of the standard (quadratic) choice for H_d, e.g. the one given by Eq. (2.13).

$\mathcal{S}$ used during training is defined as

$$\mathcal{S} = \left\{ x \in \mathbb{R}^4 \,:\, \left\| x - \begin{bmatrix} 0 & 0 & 2\pi n & 0 \end{bmatrix}^\top \right\| \leq 0.01,\, n \in \mathbb{Z},\, w \in \mathbb{R} \right\}.$$

We use a neural network for H_d^θ with the following dimension in each layer: (4, 64, 64, 1). Since the cart-pole system is symmetric, we apply the procedure described in Section 3.1 to impose symmetry. In particular, we process the input with $h(x) = x^2$ before feeding it to the neural network. The training procedures in Algorithm 1 are otherwise carried out with the same set of hyperparameters used for the IWP training. The learned controller is used to bring the state close to $x^\star$, where we switch to the LQR controller to stabilize the upright position. The LQR gains are designed by solving the Riccati equation with the cost matrices $Q = R$ chosen as the identity matrix.

We evaluate a total of 2,500 simulations starting from a range of initial states drawn from $\theta \in \pm[30, 150]$ degrees, $\dot{\theta} \in [8, 8]$ rad/s, and zero in other dimensions. The

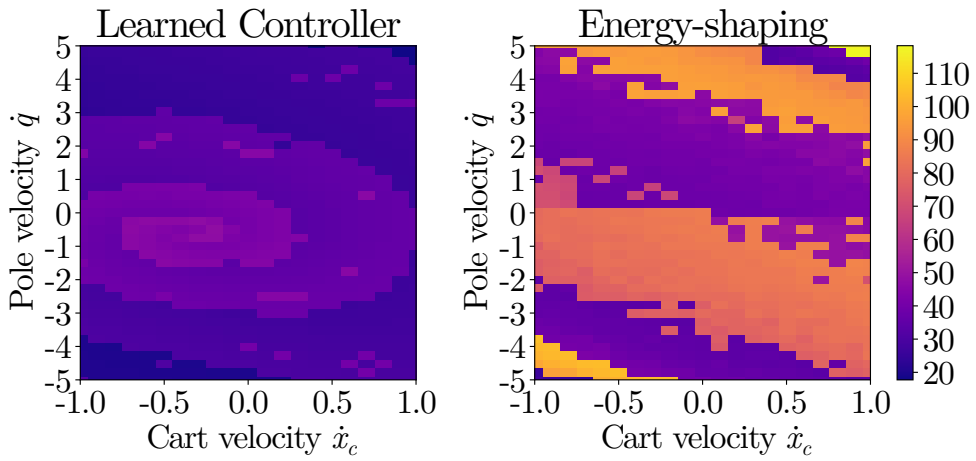


Figure 5.13: Comparison of accumulated quadratic cost (5.3) over an 8-second simulation from the corresponding initial cart and pole velocity in each grid. The initial cart position is $x_c = 0$ and the pole starts at the downward position. The learned controller performs the swing-up while incurring lower cost than the classical energy-shaping controller [3].

learned controller achieves a 100% success rate for the cart-pole system. The trained energy-like function H_d^θ and the corresponding control law are shown in Figure 5.12. To demonstrate the advantages of our approach over the classical energy-shaping control [3], we compare in Figure 5.13 the accumulated cost given by

$$J_{\text{exp}} = \frac{1}{2}\int_0^T x^\top Q x + uRu \, \mathrm{d}t, \quad Q = R = I_n, \tag{5.3}$$

incurred during the swing-up on the cart-pole. 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 running the NeuralPbc controller on hardware are shown in Figure 5.14. The controller swings up the pendulum without going over the cart track limit. The LQR controller successfully catches the swing and stabilizes the

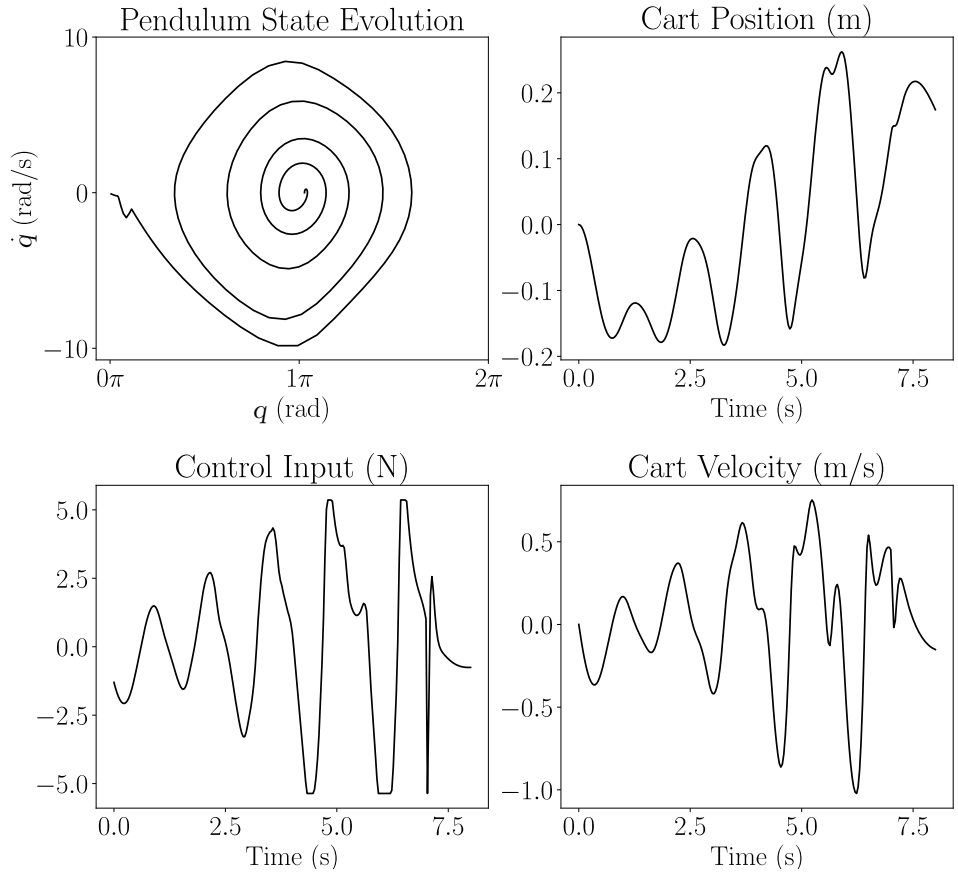
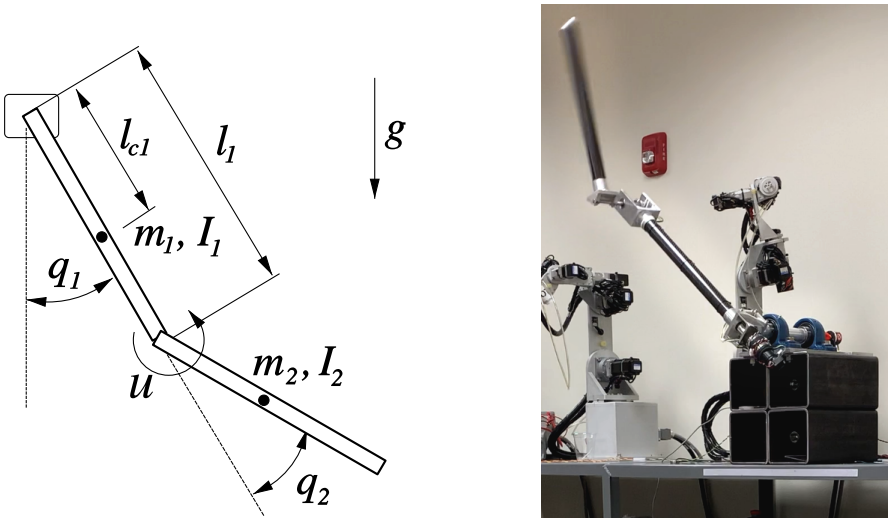


Figure 5.14: Evolution of the closed-loop cart-pole system, recorded on actual hardware. The controller is able to swing up the pendulum while maintaining within the track limits.

upright position. The successful experiments bolster the efficacy and robustness of our approach. During training, the nominal system parameters and the viscous friction model are only approximations of the true system models. The use of an approximation for velocity measurements further deviates the system from the conditions used during training. These results empirically show that our approach is robust against uncertainties to a favorable degree.



(a) Schematic (b) Hardware implementation

Figure 5.15: Schematic of the Acrobot. The elbow joint q_2 is actuated, and the shoulder joint q_1 is not. The control task studied here is the swing-up task using the control law from the NeuralPbc framework.

5.3 Acrobot

The Acrobot is a two-degree-of-freedom planar robotic manipulator with a single actuator at the elbow (joint q_2), and no actuator at the shoulder (joint q_1). The system resembles a gymnast (or acrobat, hence the name) hanging from a parallel bar, and his or her motion is controlled predominantly by the effort at the waist, and not at the wrist. Like the IWP, this system highlights one of the primary challenges in the control problems for underactuated robots. However, for this system, the coupling between the actuated and unactuated degrees of freedom is state dependent, adding complexity to the problem. Figure 5.15 (adapted from [58]) depicts the system and the convention of the configuration variables q_1 and q_2.

5.3.1 System Model

Let the state $x = (q_1, q_2, \dot{q}_1, \dot{q}_2)$. Let c_i denote $\cos q_i$, and s_i denote $\sin q_i$, for $i \in \{1, 2\}$. The equation of motion of the Acrobot is given by

$$M\ddot{q} + C\dot{q} + D = Bu - b\dot{q}, \tag{5.4}$$

where

$$M = \begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2m_3 c_2 & I_2 + m_3 c_2 \\ I_2 + m_3 c_2 & I_2 \end{bmatrix}, \qquad C = \begin{bmatrix} -2m_3 s_2 \dot{q}_2 & -m_3 s_2 \dot{q}_2 \\ m_3 s_2 \dot{q}_1 & 0 \end{bmatrix},$$
$$D = \begin{bmatrix} m_1 g l_{c1} s_1 + m_2 g (l_1 s_1 + l_{c2} s_{1+2}) \\ m_2 g l_{c2} s_{1+2} \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top, \qquad b = \begin{bmatrix} b_1 & b_2 \end{bmatrix}^\top,$$

where $m_3 = m_2 l_1 l_{c2}$. The system parameters are summarized in Table 5.4.

5.3.2 Hardware Implementation

The links l_1, l_2 are constructed using round carbon fiber tubes and machined aluminum parts. The joint q_1 has no actuation and is mounted to the ground via radial ball bearings. The position of q_1 is measured with an optical encoder with a resolution of 4096 pulses per revolution in quadrature encoding mode. The link l_1 connects joint q_1 to joint q_2, and houses the actuator for the joint q_2. The actuator is a `Maxon RE40` brushed DC motor, mounted as close as possible to the axis of rotation of q_1 to minimize the term $m_1 g l_{c1}$. A long driveshaft with bevel gears connects the joint q_2 to the motor through the inside of the link q_1. The position of q_2 is measured using an optical encoder with a resolution of 2000 pulses per revolution in quadrature encoding

Table 5.4: Model parameters for the Acrobot

Parameter	Symbol	Value	Units
Link 1 mass	m_1	2.701	kg
Link 2 mass	m_2	0.405	kg
Link 1 length	l_1	0.600	m
Link 2 length	l_2	0.532	m
CoM of link 1	l_{c1}	0.186	m
CoM of link 2	l_{c2}	0.177	m
Link 1 inertia	I_1	0.3029	kg-m^2
Link 2 inertia	I_2	0.0277	kg-m^2
Joint 1 friction coeff.	b_1	0.05	Nm-s
Joint 2 friction coeff.	b_2	0.0096	Nm-s

mode. The motor torque is controlled by a `Maxon Escon 50/5` servo controller, which is powered by a 48V power supply.

The velocity measurements are estimated from q_1 and q_2 signals using the backward difference of the positions divided by the sample period. The signal processing, motor command generation, and communication are carried out on a Teensy 4.1 development board with a microprocessor based on the ARM Cortex-M7 architecture.

5.3.3 NeuralPbc Experiments

We seek to learn a storage function H_d^θ and obtain the associated controller that swings the system up close to the unstable equilibrium $(q_1^\star, q_2^\star) = (\pm\pi, 0)$. The region of attraction of the LQR controller used for catching the swing and balancing is very small, and the controller must learn to swing the system up without overshooting.

The neural network used for H_d^θ has the dimensions (6, 128, 128, 1), equating to a total of 17,537 parameters. We use the 6-dimensional input that consists of $z = (c_1, s_1, \dot{q}_1, c_2, s_2, \dot{q}_2)$, instead of the robot's state to avoid numerical problems that

sometimes occur from multiple revolutions of q_2 during training. The goal set $\mathcal{S}$ for this system is defined as

$$\mathcal{S} = \left\{ x \in \mathbb{R}^4 \,:\, \left\| x - \begin{bmatrix} \pi n & 2\pi k & 0 & 0 \end{bmatrix}^\top \right\| \leq 0.01,\, n \in \mathbb{Z},\, k \in \mathbb{Z} \right\}.$$

The learned controller is evaluated by simulating the system for 40 seconds from a range of initial states with $10 \leq |q_1|, |q_2| \leq 90$ degrees and zero velocities. A total of 1,156 evenly-spaced samples of different initial (q_1, q_2) are simulated. A trajectory from one of these simulations are shown in Figure 5.16. The performance of the trained controller is evaluated by the ability to catch and balance the swing with LQR controller. There are 1,098 successful simulations, resulting in a success rate of 94.98%. In the failed simulations, the learned controller still adds energy and swings up the system, but LQR is unable to catch it and q_1 continues to swing past $\pm\pi$. Improving the performance of the linear controller will increase our success rate, but we do not pursue the problem in this work.

We further validate the controller by implementing it on the custom-built hardware. This tests the robustness of the learned controller against model uncertainties. In our setup, the joint q_1 has high friction that cannot be accurately represented with a viscous model. The values in Table 5.4 are only rough approximations of the true parameters, and the velocity measurements $\dot{q}_1, \dot{q}_2$ are noisy.

A recorded trajectory using the learned controller to swing up is shown in Figure 5.17. The initial state of the system is near the downward equilibrium with a small $\dot{q}_1$. The controller successfully performs the swing up and brings the system close to the upright equilibrium. To ensure stabilization with LQR, its region of attraction can be incorporated into the training process through the definition of the set distance in

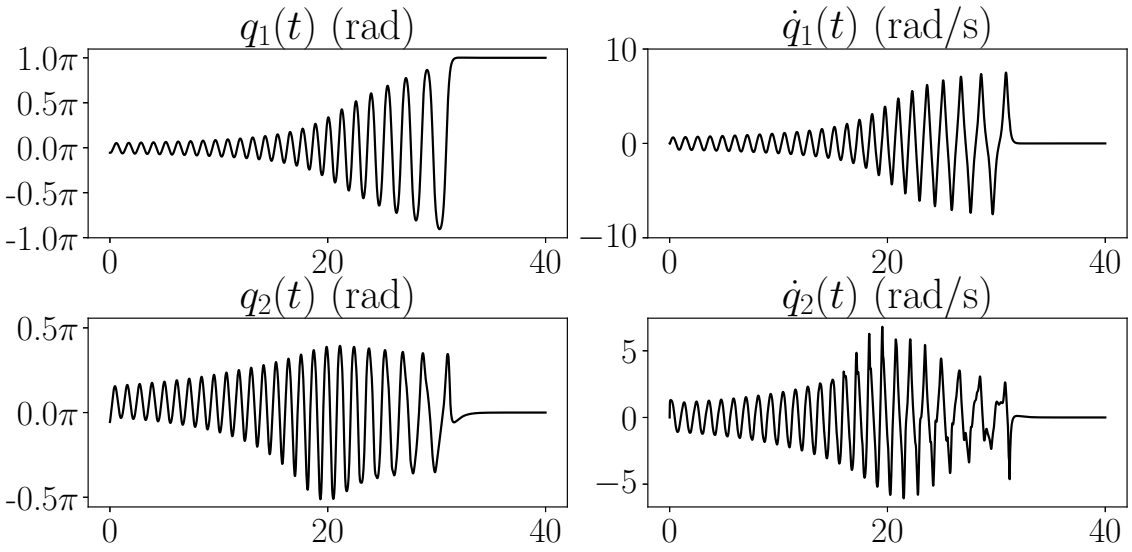


Figure 5.16: Evolution (simulation) of the Acrobot with the learned controller as input. The system starts near the downward equilibrium at $(q_1, q_2) = (-10, 10)$ degrees with near zero velocities. The learned controller swings up the system and brings the state into the region of attraction of the linear controller, allowing it to successfully stabilize at the upright position.

the loss function. With S is the solution to the corresponding Riccati equation of the LQR policy, the quantity $x^\top Sx \geq 0$ can be used to estimate the region of attraction, e.g. using Sum-of-Squares optimization [59]. With each swing, we observe that this quantity becomes smaller, suggesting that our controller can be used in conjunction with LQR and its estimated region of attraction to ensure stabilization of $x^\star$.

5.4 Ball-Beam System

The ball and beam system is depicted in Figure 5.18. The beam of length L is actuated by a revolute joint located at the center. The ball is free to roll on the beam. The position of the ball with respect to the beam center is denoted by q_1. The

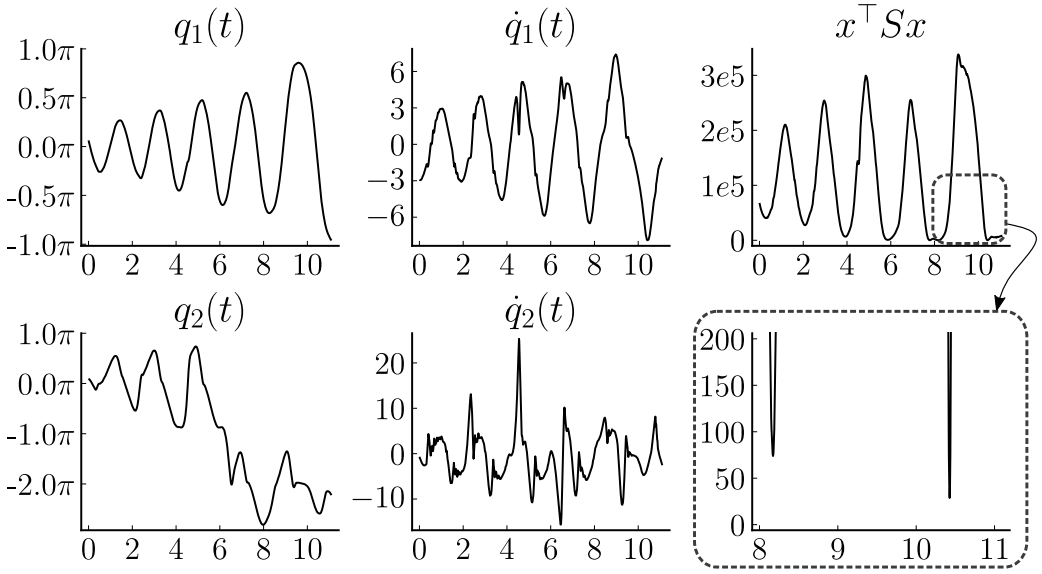


Figure 5.17: Recorded trajectory from hardware implementation with the learned controller. The learned controller adds energy and successfully swings up the system. The quantity $x^\top Sx$, which can be used to estimate the region of attraction of LQR, becomes smaller with each swing.

angle of the beam relative to the horizontal position is denoted by q_2. The control task for this system is to stabilize the equilibrium $q^\star = (0, 0)$.

5.4.1 System Model

The Hamiltonian of the system is $H = \frac{1}{2}p^\top M^{-1}p + V(q)$, where

$$H(q,p) = \frac{1}{2}p^\top M^{-1}p + V(q) = \frac{1}{2}p^\top \begin{bmatrix} 1 & 0 \\ 0 & L^2 + q_1^2 \end{bmatrix}^{-1} p + gq_1 \sin(q_2),$$

and g is the gravitational constant. The input matrix of this system is $G = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top$. The equations of motion of this system can be written as

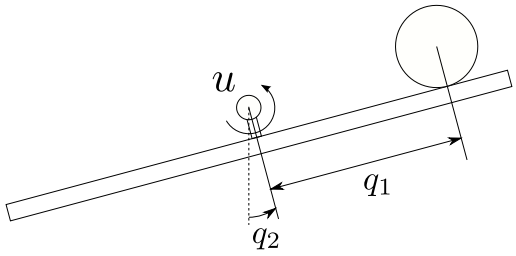


Figure 5.18: Schematic of ball and beam system. The beam is actuated by a revolute joint at the center, and the ball is free to roll on the beam.

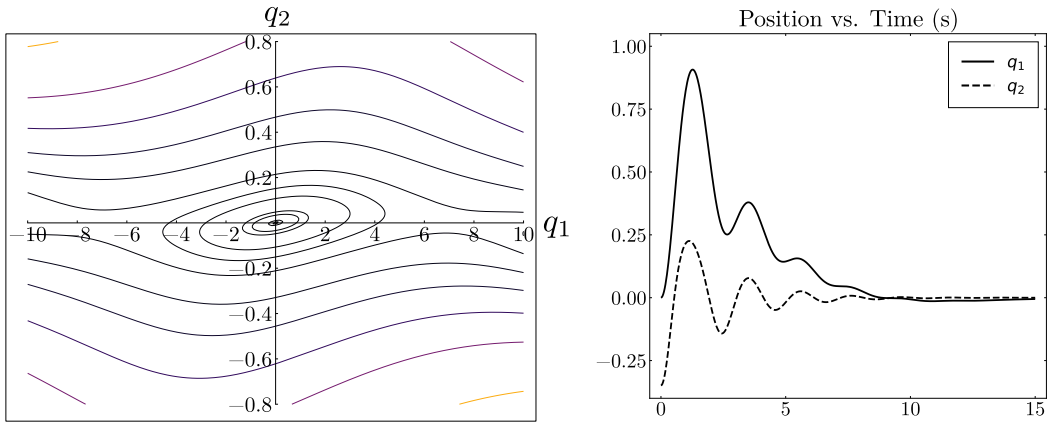
$$\ddot{q}_1 + g \sin q_2 - q_1 \dot{q}_2^2 = 0,$$
$$\left(L^2 + q_1^2\right) \ddot{q}_2 + 2q_1 \dot{q}_1 \dot{q}_2 + g q_1 \cos q_2 = u.$$

where the control input u is the torque applied to the revolute joint at the center of the beam. The length of the beam is chosen as $L = 5$ m. The control task studied in this experiment is to stabilize the equilibrium $x^\star = (q^\star, 0) = 0$.

5.4.2 NeuralIdaPbc Experiments

We begin by selecting an architecture for the three neural networks: $M_d^\theta, U_1^\theta, U_2^\theta$. Since the mass matrix $M(q)$ of this system is a function of q, it is not sufficient to let $U_1^\theta = U_2^\theta = 0$. The architectures for the function approximators are summarized in Table 5.5. The degree of the SoS polynomial V_d^θ is 4 ($d = 2$). In total, there are 3,726 parameters in θ to train.

The collection of training data consists of uniformly sampling the state from $q_1 \in [-L/2, L/2]$ and $q_2 \in [-\pi/2, \pi/2]$. The samples are then processed into batches in the same manner to the IWP sampling process. The only hyperparameters to be tuned are the batch size, how finely to sample the state space, and the termination conditions for the training algorithm.



(a) Level sets of $V_d(q)$ for the ball-beam system. (b) Time evolution of ball-beam system

Figure 5.19: (a) The energy-like function V_d^θ for the ball-beam system after training, which has an isolated minimum at the desired equilibrium $q^\star = (0, 0)$, and (b) the time evolution of (q_1, q_2) of the ball-beam system.

We demonstrate the efficacy of our approach through simulation studies. After training, the swing-up controller is derived according to equations (2.10), (2.20) and (2.17). The damping gain K_v in (2.17) is chosen as the identity matrix. A simulated trajectory executed using the learned controller is shown in Figure 5.19b. For this particular simulation, the system starts at rest with the initial configuration $q(0) = (0, 0.2618)$, i.e. the beam is rotated 15 degrees counter-clockwise, with the ball starting at the center. The learned controller successfully brings system to the desired equilibrium $x^\star$.

Combined with the NeuralIdaPbc experiments for the IWP system, these results suggest that our control design framework is applicable to the same class

Table 5.5: Neural net architectures for the program (4.12) in the ball-beam case study.

	Layer Dimensions	Activation Functions
M_d^θ equation (4.4)	(2, 32, 32, 3)	(ELU, ELU, ELU, Identity)
U_1^θ equation (4.5)	(2, 32, 32, 1)	(ELU, ELU, ELU, Identity)
U_2^θ equation (4.5)	(2, 32, 32, 1)	(ELU, ELU, ELU, Identity)

of mechanical systems where the IDAPBC is applicable. By eliminating the need to analytically solve the nonlinear PDEs, our methods overcome a major hindrance of using IDAPBC and automatically find a stabilizing controller in a data-efficient manner. Our framework provides an alternative, and arguably a more practically applicable methodology to apply IDAPBC techniques on underactuated robotic systems.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

In this dissertation, we have introduced fundamental components for incorporating passivity-based control techniques in machine learning frameworks. First, the NeuralPbc framework is presented as a flexible tool that streamlines the search process for a suitable storage function in the Pbc paradigm. Unlike traditional Pbc techniques, the control synthesis is cast as an optimization problem that incorporates performance metrics defined based on the evolution of the closed-loop system. We formulate an algorithm that efficiently finds a solution the optimization problem using techniques adapted from reinforcement learning approaches.

The second approach presented in this dissertation is the NeuralIdaPbc framework. This method eases the burden of control synthesis in the IdaPbc approach, which requires solving nonlinear partial differential equations in closed-form. NeuralIdaPbc emphasizes stability guarantees and leverages the powerful results of IdaPbc to construct a machine learning framework that transparently connects to classical stability properties. We prove that as the objective function of the learning problem vanishes, the corresponding control policy converge to a faithful IdaPbc control law, which has well-established stability properties.

A series of experiments, both in simulation and on hardware, is designed and carried out in order to study the efficacy and applicability of the NeuralPbc and

NeuralIdaPbc approaches. We show how these methods can be applied to a family of underactuated robots and evaluate their performance through these experiments. The results show that both approaches compare favorably to the traditional control laws designed using Pbc techniques. Furthermore, experiments on hardware empirically suggest that our controllers benefit from the inherent robustness properties of Pbc, as they are able to withstand errors in the nominal dynamical models used during training, as well as measurement noise encountered during control execution.

As a compelling future research direction, the robustness of properties of the NeuralPbc and NeuralIdaPbc methods may be rigorously studied and further improved by accounting for the errors in the dynamical models used during training. Bayesian neural network and the corresponding learning techniques are good candidates for tackling these problems. The deterministic neural network architectures used in NeuralPbc and NeuralIdaPbc may be replaced by Bayesian neural nets, whose weights and biases are modeled as random variables. The training algorithms need to be modified in such away that the probability distributions for these random variables are inferred, instead of finding point estimates through gradient descent. The IWP hardware implementation in this dissertation is designed with robustness tests in mind, serving as a testbed for conducting future research in this area.

REFERENCES

[1] 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.

[2] 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.

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

[4] R. Ortega and M. W. Spong, "Adaptive motion control of rigid robots: A tutorial," *Automatica*, vol. 25, no. 6, pp. 877–888, 1989.

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

[6] J. A. Acosta, R. Ortega, A. Astolfi, and A. D. Mahindrakar, "Interconnection and damping assignment passivity-based control of mechanical systems with underactuation degree one," *IEEE Transactions on Automatic Control*, vol. 50, no. 12, pp. 1936–1955, 2005.

[7] A. D. Mahindrakar, A. Astolfi, R. Ortega, and G. Viola, "Further constructive results on interconnection and damping assignment control of mechanical systems: The acrobot example," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 16, no. 14, pp. 671–685, 2006.

[8] 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.

[9] G. Viola, R. Ortega, R. Banavar, J. Á. Acosta, and A. Astolfi, "Total energy shaping control of mechanical systems: simplifying the matching equations via coordinate changes," *IEEE Transactions on Automatic Control*, vol. 52, no. 6, pp. 1093–1099, 2007.

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

[11] 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.

[12] 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.

[13] 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.

[14] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[15] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[16] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

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

[19] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[20] S. P. Nageshrao, G. A. Lopes, D. Jeltsema, and R. Babuška, "Passivity-based reinforcement learning control of a 2-dof manipulator arm," *Mechatronics*, vol. 24, no. 8, pp. 1001–1007, 2014.

[21] R. Ortega, A. J. Van Der Schaft, I. Mareels, and B. Maschke, "Putting energy back in control," *IEEE Control Systems Magazine*, vol. 21, no. 2, pp. 18–33, 2001.

[22] W. Sirichotiyakul and A. C. Satici, "Data-driven design of energy-shaping controllers for swing-up control of underactuated robots," in *International Symposium on Experimental Robotics*. Springer, 2020, pp. 323–333.

[23] ——, "Combining energy-shaping control of dynamical systems with data-driven approaches," in *2021 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2021, pp. 1121–1127.

[24] ——, "Data-driven passivity-based control of underactuated mechanical systems via interconnection and damping assignment," *International Journal of Control*, 2022.

[25] W. Sirichotiyakul, N. A. Ashenafi, and A. C. Satici, "Robust data-driven passivity-based control of underactuated systems via neural approximators and bayesian inference," in *2022 American Control Conference*. IEEE, 2022.

[26] V. I. Arnol'd, *Mathematical methods of classical mechanics*. Springer Science & Business Media, 2013, vol. 60.

[27] H. Khalil, *Nonlinear Systems*, ser. Pearson Education. Prentice Hall, 2002.

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

[29] G. A. Leonov, "Generalization of the andronov-vitt theorem," *Regular and chaotic dynamics*, vol. 11, no. 2, pp. 281–289, 2006.

[30] J. Hauser and C. C. Chung, "Converse lyapunov functions for exponentially stable periodic orbits," *Systems & Control Letters*, vol. 23, no. 1, pp. 27–34, 1994.

[31] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

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

[33] Y. Cao, S. Li, L. Petzold, and R. Serban, "Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution," *SIAM journal on scientific computing*, vol. 24, no. 3, pp. 1076–1089, 2003.

[34] J. Calver and W. Enright, "Numerical methods for computing sensitivities for odes and ddes," *Numerical Algorithms*, vol. 74, no. 4, pp. 1101–1117, 2017.

[35] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, "Universal differential equations for scientific machine learning," *arXiv preprint arXiv:2001.04385*, 2020.

[36] D. Henrion and A. Garulli, *Positive polynomials in control*. Springer Science & Business Media, 2005, vol. 312.

[37] P. A. Parrilo, *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

[38] A. Prestel and C. Delzell, *Positive polynomials: from Hilberts 17th problem to real algebra*. Springer Science & Business Media, 2013.

[39] M.-D. Choi, T. Y. Lam, and B. Reznick, "Sums of squares of real polynomials," in *Proceedings of Symposia in Pure mathematics*, vol. 58. American Mathematical Society, 1995, pp. 103–126.

[40] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[41] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[42] I. R. Manchester, "Transverse dynamics and regions of stability for nonlinear hybrid limit cycles," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 6285–6290, 2011.

[43] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.

[44] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, S. Thrun, and R. C. Arkin, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[45] B. Conway, "Practical methods for optimal control using nonlinear programming," 2002.

[46] 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.

[47] M. W. Spong, P. Corke, and R. Lozano, "Nonlinear control of the reaction wheel pendulum," *Automatica*, vol. 37, no. 11, pp. 1845–1851, 2001.

[48] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," in *In AISTATS*. Citeseer, 2011.

[49] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[50] P. Hartman, *Ordinary differential equations*. SIAM, 2002.

[51] 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.

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

[53] A. Griewank *et al.*, "On automatic differentiation," *Mathematical Programming: recent developments and applications*, vol. 6, no. 6, pp. 83–107, 1989.

[54] M. W. Spong, "The swing up control problem for the acrobot," *IEEE control systems magazine*, vol. 15, no. 1, pp. 49–55, 1995.

[55] *IEC 61800-7-201: Adjustable speed electrical power drive systems - Part 7-201: Generic interface and use of profiles for power drive systems - Profile type 1 specification*, International Electrotechnical Commission, 2015, rev. 2.0.

[56] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[57] *Linear Servo Base Unit with Inverted Pendulum*, Quanser, 2013, rev. 1.0.

[58] R. Tedrake, *Underactuated Robotics*, 2022. [Online]. Available: underactuated.mit.edu

[59] W. Sirichotiyakul, A. C. Satici, E. S. Sanchez, and P. A. Bhounsule, "Energetically-optimal discrete and continuous stabilization of the rimless wheel with torso," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 59230. American Society of Mechanical Engineers, 2019, p. V05AT07A067.