Linear Quadratic Control using Reinforcement Learning and Quadratic Neural Networks

Soroush Asri

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science in Electrical and Computer Engineering

Concordia University

Montréal, Québec, Canada

January 2024

© Soroush Asri, 2024

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Soroush Asri

Entitled: Linear Quadratic Control using Reinforcement Learning and Quadratic Neural Networks

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

______________________ Chair

Dr. R. Selmic

______________________ External Examiner

Dr. W. Lucia

______________________ Examiner

Dr. R. Selmic

______________________ Supervisor

Dr. L. Rodrigues

Approved by ______________________

Dr. Y. R. Shayan, Chair

Department of Electrical and Computer Engineering

__________ 2023 ______________________

Dr. M. Debbabi, Dean

Faculty of Engineering and Computer Science

Abstract

Linear Quadratic Control using Reinforcement Learning and Quadratic Neural Networks

Soroush Asri

This thesis focuses on the application of reinforcement learning (RL) techniques to design optimal controllers and observers for linear time-invariant (LTI) systems, namely linear quadratic regulator (LQR), linear quadratic tracker (LQT), and linear quadratic estimator (LQE), utilizing measured data. The closed-form solution and wide-ranging engineering applications of the linear quadratic (LQ) problems have made it a preferred benchmark for assessing RL algorithms. The primary contribution lies in the introduction of novel policy iteration (PI) methods, wherein the value-function approximator (VFA) is designed as a two-layer quadratic neural network (QNN) trained through convex optimization. To the best of our knowledge, this is the first time that a convex optimization-trained QNN is employed as the VFA. The main advantage is that the QNN's input-output mapping has an analytical expression as a quadratic form, which can then be used to obtain an analytical linear expression for policy improvement. This is in stark contrast to available techniques that must train a second neural network to obtain the policy improvement. Due to the quadratic input-output mapping of the QNNs and the quadratic form of the value-function in the LQ problems, the QNN is a suitable VFA candidate. The thesis designs the LQR and LQT without requiring the system model. The thesis also designs the LQE correcrtion term provided that the system model is given. The thesis establishes the convergence of the learning algorithm to the LQ solution provided one starts from a stabilizing policy. To assess the proposed approach, extensive simulations are conducted using MATLAB, demonstrating the effectiveness of the developed method. Furthermore, the proposed observer is designed for a nonlinear pendulum with a given linearized model and it is shown that the proposed observer is improved over utilizing only linearized model. This shows the adaptability for nonlinear systems.

Acknowledgements

I would like to express my heartfelt gratitude to all those who have contributed to the completion of this thesis. First and foremost, I extend my deepest appreciation to my thesis supervisor, Dr. Luis Rodrigues, whose guidance, expertise, and unwavering support have been invaluable throughout this journey. His insightful feedback and encouragement have been instrumental in shaping the direction of this research. I am also indebted to MITACS for their generosity and support in funding my research, without which this work would not have been possible. I'm incredibly grateful to SII Canada for their support during my master's project. It really helped with my thesis and gave me great experiences.

My sincere thanks go to my friends, girlfriend, and colleagues for their constant encouragement and understanding during the challenging times of this research. Their willingness to lend a helping hand have made this endeavor more enjoyable. I would like to express my gratitude to Concordia's faculty and staff for their continuous support during my graduate studies. Lastly, I want to express my deepest gratitude to my family for their unending love, support, and patience. Their unwavering belief in my abilities has been the driving force behind my pursuit of knowledge

This thesis would not have come to fruition without the collective efforts and encouragement of all these individuals, and I am truly honored and thankful for their significant contributions to my academic journey.

Contents

List of Figures

List of Tables

Glossary

ANN artificial neural network.

ARE algebraic Riccati equation.

LQ linear quadratic.

LQE linear quadratic estimator.

LQR linear quadratic regulator.

LQT linear quadratic tracker.

LTI linear time-invariant.

NN neural networks.

PE persistent excitation.

PI policy iteration.

QFA Q-function approximator.

QNN quadratic neural network.

RL reinforcement learning.

VFA value-function approximator.

Chapter 1

Introduction

1.1 Motivation

Optimal controllers and observers minimize a given cost function subject to a dynamic model. This design method has received a lot of attention, especially due to its potential in applications such as autonomous vehicle navigation and robotics, economics and management, as well as energy optimization, to cite a few [1]. Traditionally, optimal control and observer design relies on well-established techniques like dynamic programming, calculus of variations, and Pontryagin's maximum principle [2]. While these methods have proven effective in many cases, they often face challenges when dealing with large-scale, nonlinear, or uncertain systems [3][4]. To address these problems data-driven methods such as RL have been studied for optimal control problems [5][6]. RL algorithms learn from their interactions with the system, dynamically adapting the control strategies to minimize the cost function [7]. The rise of RL in artificial intelligence has demonstrated remarkable success in learning complex decision-making strategies in uncertain situations [8].

The motivation for exploring the integration of RL in optimal control and observer design stems from several key factors: [5][9]

1. **Adaptability:** In practical applications, optimal control problems can be complex

with varying dynamics and uncertainties in the system model. While traditional control methods often derive analytical solutions for each specific problem, RL can provide a general real-time data-driven approach to optimize control policies. Therefore, RL can adapt to this varying dynamics and uncertainties, distinguishing it from traditional methods.

2. **Handling Complex Dynamics:** Many real-world systems have nonlinear dynamics and are subject to uncertainties. Traditional optimal control methods struggle to find optimal control solution in such cases, whereas RL algorithms can leverage function approximators and neural networks (NN) to approximate complex optimal control policies efficiently, even in high-dimensional state spaces.

3. **Overcoming Model Inaccuracies:** In practice, obtaining an accurate system model may be challenging. RL can acquire optimal control policies even without the system model.

4. **Combining Traditional with Data-Driven Control Methods:** RL offers a unique opportunity to combine traditional control strategies with data-driven learning approaches. This integration can result in improved performance and ensure safety in critical applications.

The LQ problem is an optimal control task that seeks to find a control strategy that minimizes a quadratic cost function over a given time horizon for a linear dynamic system. The closed-form solution and wide-ranging engineering applications of the LQ problem have made it a preferred benchmark for assessing RL algorithms [10][11]. Consequently, the RL community has shown significant interest in solving LQ problems using different techniques to ultimately address nonlinear optimal control problems effectively [12][9]. As indicated in articles [13][14][15], a considerable number of papers utilize NNs within their RL algorithms to solve optimal control problems. However, employing a artificial neural network (ANN) as a function approximator in RL algorithms presents several noteworthy drawbacks [16]:

1. The optimization of the NN's weights is not a convex optimization and training the NN yields locally optimal weights.

2. Selecting the appropriate architecture for the NN typically involves a trial and error procedure.

3. The NN lacks a straightforward analytical expression.

Hence, this led us to explore the integration of RL techniques and quadratic neural networks (QNN) [17] to solve LQR, LQT, and LQE with the specific objective to overcome the drawbacks associated with employing ANNs.

1.2 Literature Survey

Typically, optimal control and observer problems are solved through two primary approaches. One method, dynamic programming, relies on Bellman's principle of optimality [18]. Another approach involves Pontryagin's maximum principle [2].

However, most real systems exhibit nonlinear dynamics, which can pose challenges when attempting to solve nonlinear optimal control problems. To address this issue, practical approaches often involve approximating nonlinear systems with linear models, enabling the use of an LQ controller with a closed-form solution [19][20]. The LQ problem including LQR [21], LQT [22], LQE [23], LQ Gaussian (LQG) [21] can then be formulated as finding the solution of a Riccati equation [2]. An iterative technique for computing solutions of the Riccati equation is discussed in article [24]. Nevertheless, employing this iterative technique to solve the LQ problem results in an off-line backward-in-time procedure, necessitating an accurate linear model of the system. There are also instances where complex nonlinear systems cannot be adequately approximated by linear models or where the system's dynamics remain unknown.

Therefore, conventional techniques for solving optimal control problems may not be applicable. As a result, researchers have explored data-driven methods such as RL, system identification, imitation learning, Gaussian process regression, and evolutionary algorithms to find approximations to the optimal solution [25][26][27].

1. **Imitation Learning**: In this approach, data from expert demonstrations or optimal control trajectories is used to learn a control policy that imitates the demonstrated behavior. It is particularly useful when access to an optimal control solution is available for a few scenarios [25].

2. **System Identification:** In this approach, data is utilized to acquire a model for the system through diverse methods such as dynamic mode decomposition (DMD), and subsequently, the optimal control is formulated based on the obtained model [26].

3. **Gaussian Process Regression (GPR):** GPR is a non-parametric data-driven method that can be used for system identification and control. It models the underlying dynamics as a Gaussian process and uses Bayesian inference to make predictions about the system's behavior [27].

4. **Evolutionary Algorithms:** Evolutionary algorithms, such as genetic algorithms, can be used to search for control policies. These algorithms explore the solution space by iteratively evolving a population of candidate policies [28].

5. **RL algorithms:** RL is a powerful data-driven approach that enables an agent to learn optimal control policies by interacting with the environment and adjusting its actions or control policies to maximize cumulative rewards [29]. The majority of the research focusing on RL in optimal control explores discrete-time optimal control [6].

This thesis focuses on integrating RL into discrete-time optimal control problems. There are two main RL methods: model-based RL, and model-free RL [30] [14].

1. **Model-Based RL:** In model-based RL, the agent uses a learned or known model of the environment, which includes information about the system dynamics, transition

probabilities [7], and rewards. Model-based methods can be more sample-efficient in some situations because they can use the model to generate hypothetical data for learning, reducing the need for real-world interactions with the environment.

2. **Model-Free RL:** In model-free RL, the agent does not explicitly learn a model of the environment but instead learns a policy or value-function [7] directly from interacting with the environment providing simplicity and robustness in handling complex environments with uncertain dynamics [7]. Model-free methods do not require knowledge of the system dynamics, making them more applicable to a wider range of problems, especially when the dynamics are complex or unknown.

Model-free RL has gained increasing attention due to its interpretation as direct adaptive control [31][32]. Its growing popularity stems from the fact that model-free RL does not rely on system identification, making it a compelling choice for handling complex nonlinear systems [33]. Model-free RL can be divided into two categories: value-based and policy-based algorithms [11].

1. **Value-based RL:** Value-based RL is a broad category of RL algorithms that focus on estimating value-functions, such as the state-value-function (V-function) or the action-value-function (Q-function) [5]. By learning these value-functions, value-based RL algorithms can subsequently derive the optimal policy. One popular approach to estimate the value-function is using the temporal difference (TD) equation [34], where it minimizes the Bellman error by iteratively updating the value-function. The Bellman error is the difference between the estimated value-function and the value-function defined by the Bellman equation. Value-based algorithms that use TD equation are called adaptive dynamic programming (ADP) [6].

2. **Policy-based RL:** Policy-based RL is an approach where the agent directly learns a strategy to select actions in different states, without explicitly estimating the value-function.

ADP is widely favored in RL due to its link with dynamic programming and its capacity for online learning without the system model [35]. The applications of ADP methods, the policy-iteration (PI) and the value-iteration algorithms (VI), to feedback control are discussed in references [6][36]. For optimal control, PI is preferred over VI due to its stability arising from the policy improvement guarantee, which leads to reliable convergence to the optimal policy. Additionally, PI's incremental policy updates help avoid large policy swings and ensure more robust learning in complex environments [37].

The PI algorithm has two phases: policy evaluation and policy improvement. Instead of directly solving the Hamilton-Jacobi-Bellman (HJB) equation, it first evaluates the value-function of a given initial stabilizing policy. The value-function associated with this policy is then minimized to derive an improved policy. The two phases are then repeated until the optimal policy is obtained.

In the motivation section, it was mentioned that LQ problems have been implemented with various RL algorithms to address their wide range of applications and to serve as benchmarks for evaluating the performance of these RL algorithms. Therefore, PI has been applied to solve discrete-time LQR in [38], discrete-time LQG in [39][40], and discrete-time LQT in [41][42] without the system model. References [43][44] use the PI algorithm to obtain the discrete-time LQE to estimate all the states using the system model.

Generally, ANNs are utilized to approximate the value-function in the policy evaluation step [13][14]. However, in addition to the mentioned issues of ANNs in the motivation section, using ANNs for policy evaluation requires employing another ANN to minimize the value-function with respect to the control policy, as ANNs lack a simple analytical input-output mapping [5]. To address the mentioned issues, a two-layer QNN, trained by convex optimization and introduced in [17], can be chosen as the value-function approximator. It is shown in [16][45] that QNNs work well in applications such as regression, classification, system identification and control of dynamical systems. The advantages of using the two-layer QNN as the value-function approximator compared to non-quadratic neural networks are:

1. Two-layer QNNs are trained by solving a convex optimization. Therefore, the global optimal weights are obtained [17].
2. The optimal number of neurons in the hidden layer is obtained by solving the convex optimization problem [16].
3. The input-output mapping of the QNN is a quadratic form [16]. As a result, one can analytically minimize the value-function with respect to the control policy instead of using a second neural network in the policy improvement step.

Consequently, we were encouraged to choose two-layer QNNs as the value-function approximator in a PI algorithm to solve discrete-time LQ problems.

1.3 Contributions

This thesis proposes a novel approach to design discrete-time LQR, LQT, and LQE using the PI algorithm with a two-layer QNN as the policy evaluator. The LQR and LQT are designed without requiring the system model. Replacing the ANN with a two-layer QNN in the PI algorithm, a step taken for the first time, reveals an extensive range of benefits:

1. The value-function can be analytically minimized with respect to the control policy, eliminating the requirement for a second neural network in the policy improvement step.
2. Both the input-output mapping of the two-layer QNN and the value-function of the LQ problems are quadratic. This allows for the direct derivation of a linear control policy. Therefore, the two-layer QNN is a suitable candidate to approximate the value-function.

3. The global optimal weights are achieved by training two-layer QNNs using a convex optimization problem.

4. By solving the convex optimization problem, the optimal number of neurons in the hidden layer is determined.

5. Demonstration that QNN can be used effectively in RL.

The thesis verifies the convergence to LQ problem solutions through both theoretical proofs and practical case studies using MATLAB simulations.

1.4 Thesis Structure

This thesis is organized into several chapters. Firstly, chapter 2 offers an overview of the theoretical concepts necessary for this work. It covers discrete-time LQR, LQT, LQE, as well as reinforcement learning, focusing on the PI algorithm, and a detailed review of the two-layer QNN. In Chapter 3, the proposed algorithm for solving LQR is presented, and it is further extended to address LQT problems. Lastly, chapter 4 outlines the proposed algorithm for solving LQE, leveraging the duality between LQR and LQE [46]. Chapter 5 closes the thesis by presenting the conclusions.

Chapter 2

Theoretical Preliminaries

This chapter gives essential theoretical concepts to establish the foundation for this work. It covers discrete-time LQR, LQT, LQE, and presents reinforcement learning, with a specific focus on the PI algorithm. Additionally, a comprehensive review of the two-layer QNN is provided.

2.1 Reinforcement Learning

First, we introduce some terms in RL [7].

2.1.1 Fundamental Concepts in Reinforcement Learning

- **Agent:** Agent is an entity that interacts with an environment, making decisions based on the observed states to minimize cumulative cost through a learning process.
- **Environment:** The environment in RL is the system where the agent operates and from which it receives observed states and penalties based on its actions.
- **State:** The state s_k represents the situation or configuration of the environment at time step k, containing all relevant information that the agent uses to take actions.

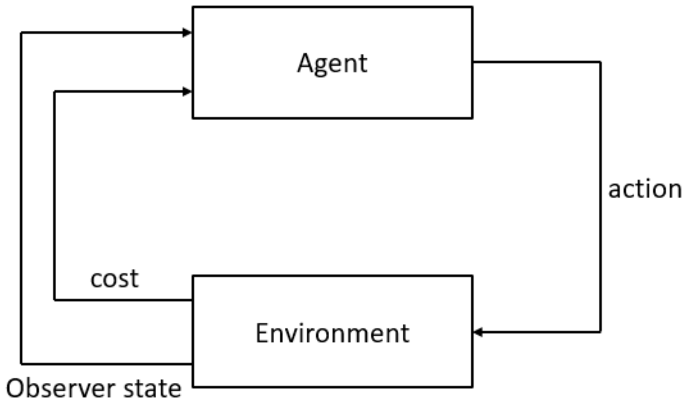


Figure 2.1: Interaction between the agent and the environment

- **Action:** The action a_k is a decision or a move made by the agent at the state s_k.
- **Policy:** A policy $\pi(.)$ is a strategy or rule that an agent follows to make decisions in an environment, specifying how it selects actions based on states to minimize costs.
- **Observed state:** The observed state refers to the state that the agent have access, which serves as input for decision-making and action selection.
- **Local cost:** the local cost $c(s_k, a_k)$ typically refers to the immediate cost associated with taking action a_k in the state s_k. It represents the immediate impact of the chosen action a_k.
- **V-function:** V-function $V^\pi(s_k)$ is a function that estimates the expected cumulative cost an agent experiences from the given state s_k while following the policy $\pi(.)$. It represents the long-term value of being in that state.

  Mathematically, it is written as:

$$V^\pi(s_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \cdot c(s_i, a_i) \tag{2.1}$$

where $0 < \gamma \leq 1$ is a discount factor to prioritize the costs that occur in the near future over those that occur further in the future.

- **Q-function:** Minimizing the V-function to improve the policy may require knowledge of the environment's dynamics [6]. To address this limitation and avoid relying on system dynamics, an alternative approach uses the Q-function [47]. The Q-function $Q^{\pi}(s_k, a_k)$ estimates the expected cumulative cost by taking the action a_k at the state s_k and following the policy $\pi(.)$ thereafter.

The RL objective is to find the optimal policy $\pi^*(.)$ that minimizes $V^{\pi}(s_k)$ for all states s_k. Figure 2.1 shows the interaction between the agent and the environment.

2.1.2 Policy Iteration Method

PI is a RL algorithm with the assumption that a stabilizing policy $\pi_0(.)$ is known. The PI has two steps:

1. **Policy Evaluation:** Policy evaluation estimates $V^{\pi}(.)$ for a given stabilizing policy $\pi(.)$. It involves solving the Bellman equation (2.2), obtained from equation (2.1), for $V^{\pi}(s_k)$.
$$V^{\pi}(s_k) = c(s_k, a_k) + \gamma V^{\pi}(s_{k+1}) \tag{2.2}$$

2. **Policy Improvement:** Once the V-function $V^{\pi}(.)$ has been estimated, the policy improvement step aims to find a better policy $\pi'(.)$ that improves the agent's cumulative cost. The new stabilizing policy $\pi'(.)$ is found by
$$\pi'(s_k) = \arg\min_{\pi} \left[c(s_k, a_k) + \gamma V^{\pi}\left(s_{k+1}\right)\right] \tag{2.3}$$

The PI algorithm using a V-function is given in Algorithm 1.

Algorithm 1 Policy iteration using V-function

```
Select an initial stabilizing policy $\pi_0(.)$. Then, for $j = 0, 1, 2, \ldots$ perform policy evaluation and policy improvement steps until convergence
Policy Evaluation:
    Obtain $V^{\pi_j}(.)$ by solving
        $V^{\pi_j}(s_k) = c(s_k, a_k) + \gamma V^{\pi_j}(s_{k+1})$    (2.4)
Policy improvement:
    Obtain $\pi_{j+1}(.)$ by solving
        $\pi_{j+1}(s_k) = \arg\min_{\pi_j} [c(s_k, a_k) + \gamma V^{\pi_j}(s_{k+1})]$    (2.5)
```

Nevertheless, applying the policy improvement step using Algorithm 1 may demand knowledge of the environment's dynamics, which could be inaccessible. To tackle this challenge, one can estimate the Q-function during policy evaluation instead [47].

This is achieved by employing the definition of the Q-function, which is given as

$$Q^{\pi}(s_k, a_k) = c(s_k, a_k) + \gamma V^{\pi}(s_{k+1}) \tag{2.6}$$

Utilizing Bellman's principle of optimality [6], the equation (2.6) can be written as

$$Q^{\pi}(s_k, a_k) = c\left(s_k, a_k\right) + \gamma Q^{\pi}\left(s_{k+1}, \pi\left(s_{k+1}\right)\right) \tag{2.7}$$

Hence, utilizing equation (2.7), one can estimate the Q-function to carry out PI without requiring knowledge of the system parameters. This leads to the development of Algorithm 2 [48].

Algorithm 2 Policy iteration using Q-function

```
Select an initial stabilizing policy $\pi_0$. Then, for $j = 0, 1, 2, \dots$ perform policy evaluation
and policy improvement steps until convergence
Policy Evaluation:
    Obtain $Q^{\pi_j}(.)$ by solving
        $Q^{\pi_j}(s_k, a_k) = c(s_k, a_k) + \gamma Q^{\pi_j}(s_{k+1}, \pi_j(s_{k+1}))$        (2.8)
Policy improvement:
    Obtain $\pi_{j+1}(.)$ by solving
        $\pi_{j+1}(s_k) = \arg\min_{a_k} Q^{\pi_j}(s_k, a_k)$        (2.9)
```

Solving equations (2.4) and (2.8) can be challenging as the Bellman equation is a Lyapunov function [6]. Given that the Bellman equation is a fixed point equation [6], we can derive $V^{\pi}(\cdot)$ and $Q^{\pi}(\cdot)$ for the stabilizing policy $\pi(\cdot)$ by employing equations (2.10) and (2.11), respectively. These iterative processes, starting with any initial value V_0^{π} and Q_0^{π}, converge to the limit $V_i^{\pi} \to V^{\pi}$ and $Q_i^{\pi} \to Q^{\pi}$ [14]. Consequently, (2.4) and (2.8) can be replaced by iterative quations (2.10) and (2.11), respectively.

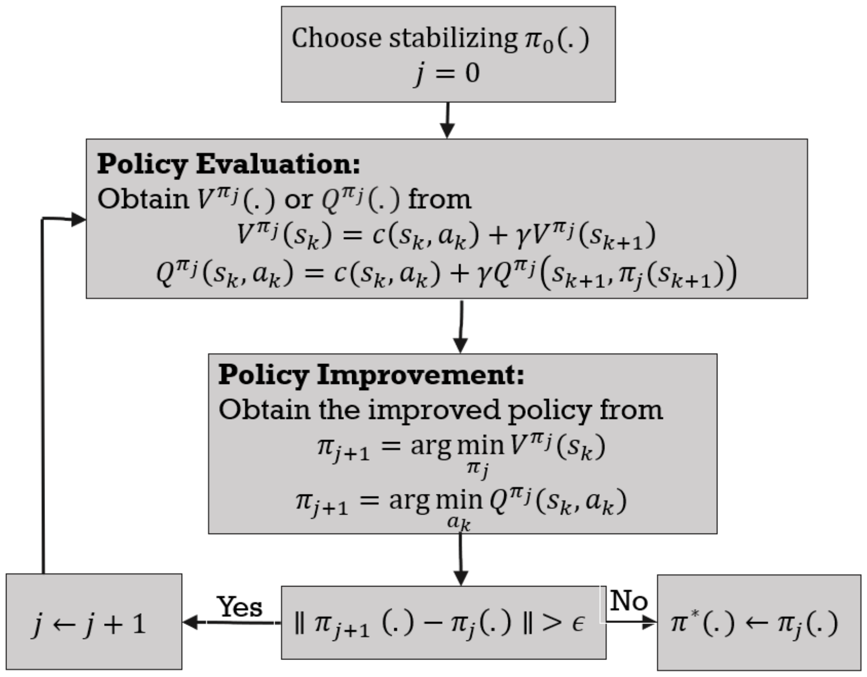
$$V_{i+1}^{\pi}(s_k) = c(s_k, a_k) + \gamma V_i^{\pi}(s_{k+1}) \tag{2.10}$$

$$Q_{i+1}^{\pi}(s_k, a_k) = c(s_k, a_k) + \gamma Q_i^{\pi}(s_{k+1}, \pi(s_{k+1})) \tag{2.11}$$

This thesis uses a two-layer QNN as the VFA to solve iterative equations (2.10) and (2.11) for the LQ problems.

Figure 2.2 illustrates the PI method through a flowchart, offering a simplified and comprehensive overview with the stopping criterion ϵ.

Remark 1. *PI algorithms require persistent excitation (PE) [6][39]. To achieve PE, a probing noise term n_k can be added to the action a_k. It is shown in reference [39] that the solution computed by PI differs from the actual value corresponding to the Bellman equation*



Choose stabilizing $\pi_0(.)$
$j = 0$
Policy Evaluation:
Obtain $V^{\pi_j}(.)$ or $Q^{\pi_j}(.)$ from
$V^{\pi_j}(s_k) = c(s_k, a_k) + \gamma V^{\pi_j}(s_{k+1})$
$Q^{\pi_j}(s_k, a_k) = c(s_k, a_k) + \gamma Q^{\pi_j}(s_{k+1}, \pi_j(s_{k+1}))$
Policy Improvement:
Obtain the improved policy from
$\pi_{j+1} = \arg\min_{\pi_j} V^{\pi_j}(s_k)$
$\pi_{j+1} = \arg\min_{a_k} Q^{\pi_j}(s_k, a_k)$
$\| \pi_{j+1}(.) - \pi_j(.) \| > \epsilon$
Yes
$j \leftarrow j + 1$
No
$\pi^*(.) \leftarrow \pi_j(.)$

Figure 2.2: Overview of policy iteration method

when the probing noise term n_k is added. It is discussed in reference [39] that adding the discount factor γ reduces this harmful effect of n_k.

2.2 Optimal Controllers for Discrete-Time Linear Time-Invariant Systems

In this section, we examine discrete-time LQR, LQT, and LQE, along with the conventional approaches for solving them.

2.2.1 Linear Quadratic Regulators

Consider a discrete-time LTI dynamical system described by the state equation:

$$x_{k+1} = Ax_k + Bu_k \tag{2.12}$$

where x_k is the state vector at time step k, u_k is the control input at time step k, A and B are constant matrices that represent the system dynamics. Assume the system is controllable [49] and all states can be measured without noise.

The objective is to find a sequence of control inputs that minimizes the quadratic cost function

$$V^{\pi}(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \left(x_i^T Q x_i + u_i^T R u_i \right) \tag{2.13}$$

where $0 < \gamma \leq 1$ is the discount factor, Q and R are both symmetric positive definite [50] weighting matrices that penalize the state and control effort, respectively, and $V^{\pi}(x_j)$ is the value-function at time step j,.

The optimal control law can be obtained by solving the discrete-time algebraic Riccati equation (ARE) [51]

$$P^{\pi} = Q + \gamma A^T P^{\pi} A - \gamma^2 (A^T P^{\pi} B)(R + \gamma B^T P^{\pi} B)^{-1} (B^T P^{\pi} A) \tag{2.14}$$

where P^{π} is the solution to the discrete-time ARE, and the optimal control law is given by

$$u_k = -K^{\pi} x_k \tag{2.15}$$

where $K^{\pi} = (R + \gamma B^T P^{\pi} B)^{-1} (\gamma B^T P^{\pi} A)$.

Remark 2. *To solve the ARE* (2.14) *for P^{π}, the iterative equation* (2.16) *can be initiated starting with any P_0^{π}, which leads to the convergence of P_i^{π} towards P^{π} [52].*

$$P_{i+1}^{\pi} = Q + \gamma A^T P_i^{\pi} A - \gamma^2 (A^T P_i^{\pi} B)(R + \gamma B^T P_i^{\pi} B)^{-1} (B^T P_i^{\pi} A) \tag{2.16}$$

The resulting closed-loop system with the optimal control law is

$$x_{k+1} = (A - B K^{\pi}) x_k \tag{2.17}$$

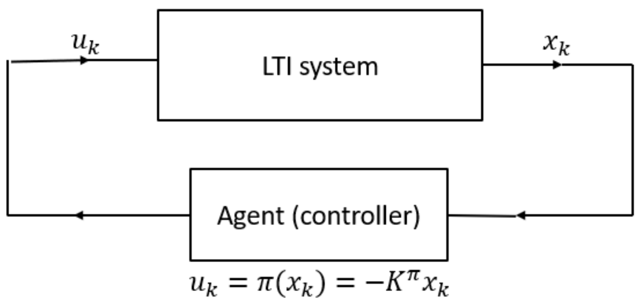


Figure 2.3: Block diagram showing LQR as a RL problem

This linear control law, provides a stable and optimal control strategy that minimizes the specified cost function over an infinite time horizon for the given linear system.

The LQR problem can be considered as a RL problem as shown by the block diagram in Fig 2.3, where the controller is the agent, u_k is the action, x_k is the state, the LTI system is the environment, and the policy is

$$\pi(x_k) = -K^\pi x_k \tag{2.18}$$

Note that the V-function has the following quadratic form [6]

$$V^\pi(x_k) = x_k^T P^\pi x_k \tag{2.19}$$

Based on the definition of the Q-function, the Q-function of the LQR can be expressed as follows

$$\begin{aligned} Q^\pi(x_k, u_k) &= c_k(x_k, u_k) + \gamma V^\pi(x_{k+1}) \\ &= x_k^T Q x_k + u_k^T R u_k + \gamma (A x_k + B u_k)^T P^\pi (A x_k + B u_k) \\ &= \begin{bmatrix} x_k & u_k \end{bmatrix} H^\pi \begin{bmatrix} x_k & u_k \end{bmatrix}^T \end{aligned} \tag{2.20}$$

where

$$H^{\pi} = \begin{bmatrix} Q + \gamma A^T P^{\pi} A & \gamma A^T P^{\pi} B \\ \gamma B^T P^{\pi} A & R + \gamma B^T P^{\pi} B \end{bmatrix} \tag{2.21}$$

As a result, both the Q-function and V-function have quadratic forms. Also note that both H^{π} and P^{π} are symmetric positive definite matrices.

2.2.2 Linear Quadratic Trackers

The LQT is an extension of the LQR that deals with the problem of controlling a LTI system while also tracking a desired reference trajectory. LQT combines the control objectives of stabilization and tracking, where the goal is that the output of the system tracks a specified reference r_k.

Consider a discrete-time LTI system with the state equation

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{2.22}$$

where x_k is the measured state vector, u_k is the input vector, and y_k is the output of the system. A, B, and C are constant matrices that represent the system dynamics. Assume (A, B) is controlable and (A, C) is observable.

The goal is to find a series of control inputs that minimizes the quadratic value-function

$$V^{\pi}(x_k, r_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \left((Cx_i - r_i)^T Q (Cx_i - r_i) + u_i^T R u_i \right) \tag{2.23}$$

where $0 < \gamma \leq 1$ is the discount factor, Q and R are both symmetric positive definite weighting matrices that penalize the tracking error and control effort, respectively.

The reference trajectory r_k is produced by (2.24) where F is a constant matrix.

$$r_{k+1} = Fr_k \tag{2.24}$$

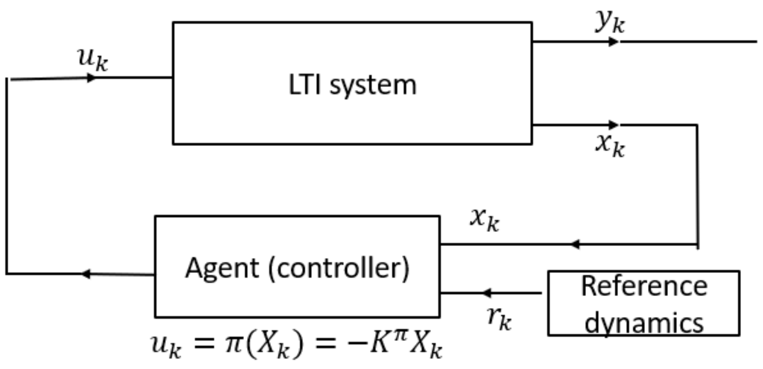


Figure 2.4: Block diagram showing LQT as a RL problem

Define the augmented state X_k [41]

$$X_k = \begin{bmatrix} x_k \\ r_k \end{bmatrix} \tag{2.25}$$

and construct the augmented system as

$$X_{k+1} = \bar{A} X_k + \bar{B} u_k \tag{2.26}$$

where

$$\bar{A} = \begin{bmatrix} A & 0 \\ 0 & F \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{2.27}$$

As a result, the objective is to solve the following optimization problem:

$$\begin{aligned} &\arg\min_{u_i} \sum_{i=k}^{\infty} \gamma^{i-k} \left(X_i^T \bar{Q} X_i + u_i^T R u_i \right) \\ &s.t. \quad X_{i+1} = \bar{A} X_i + \bar{B} u_i \end{aligned} \tag{2.28}$$

where $\bar{Q} = \begin{bmatrix} C & -I \end{bmatrix}^T Q \begin{bmatrix} C & -I \end{bmatrix}$ and I is the identity matrix. Hence, the LQT problem can be formulated as an LQR problem with the state X_k. The optimal control input has the following form

$$u_k = \pi(X_k) = -K^\pi X_k \tag{2.29}$$

where

$$K^\pi = \gamma(R + \gamma \bar{B}^T P^\pi \bar{B})^{-1} \bar{B}^T P^\pi \bar{A} \tag{2.30}$$

and P^π is the solution of the following ARE

$$P^\pi = \bar{Q} + \gamma \bar{A}^T P^\pi \bar{A} - \gamma^2 \bar{A}^T P^\pi \bar{B}(R + \gamma \bar{B}^T P^\pi \bar{B})^{-1} \bar{B}^T P^\pi \bar{A} \tag{2.31}$$

The LQT problem can be considered as a RL problem as shown by the block diagram in Fig 2.4, where controller is the agent, u_k is the action, X_k is the state, the LTI system and the reference signal dynamics make the environment, and the policy is given by

$$u_k = \pi(X_k) = -K^\pi X_k \tag{2.32}$$

Note that the V-function has the following quadratic form

$$V^\pi(X_k) = X_k^T P^\pi X_k \tag{2.33}$$

The Q-function of the LQT can be expressed as

$$\begin{aligned} Q^\pi(X_k, u_k) &= c(X_k, u_k) + \gamma V^\pi(X_{k+1}) \\ &= X_k^T \bar{Q} X_k + u_k^T R u_k + \gamma(\bar{A}X_k + \bar{B}u_k)^T P^\pi (\bar{A}X_k + \bar{B}u_k) \\ &= \begin{bmatrix} X_k & u_k \end{bmatrix} H^\pi \begin{bmatrix} X_k & u_k \end{bmatrix}^T \end{aligned} \tag{2.34}$$

where

$$H^{\pi} = \begin{bmatrix} \bar{Q} + \gamma \bar{A}^T P^{\pi} \bar{A} & \gamma \bar{A}^T P^{\pi} \bar{B} \\ \gamma \bar{B}^T P^{\pi} \bar{A} & R + \gamma \bar{B}^T P^{\pi} \bar{B} \end{bmatrix} \tag{2.35}$$

2.2.3 Linear Quadratic Estimators

Consider the discrete-time LTI system

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{2.36}$$

where x_k is the unknown state vector, u_k is the input vector, y_k is the measured output vector, and A, B, C are system matrices. Assume (A, C) is observable.

The goal is to design an optimal observer for system (2.36) and minimize a quadratic performance index.

The dynamics of the observer are as follows

$$\begin{aligned} \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + w_k^{\pi} \\ \hat{y}_k &= C\hat{x}_k \end{aligned} \tag{2.37}$$

where $\hat{x}_k$ and $\hat{y}_k$ are states and outputs estimates, respectively, and w_k^{π} is the correction term to be designed later. Combining (2.36) and (2.37) results in the following error dynamics

$$\begin{aligned} \tilde{x}_{k+1} &= A\tilde{x}_k - w_k^{\pi} \\ \tilde{y}_k &= C\tilde{x}_k \end{aligned} \tag{2.38}$$

where $\tilde{x}_k = x_k - \hat{x}_k$ and $\tilde{y}_k = y_k - \hat{y}_k$ are the state estimation error and the output estimation error, respectively. The performance index is considered as (2.39) to penalize correction term

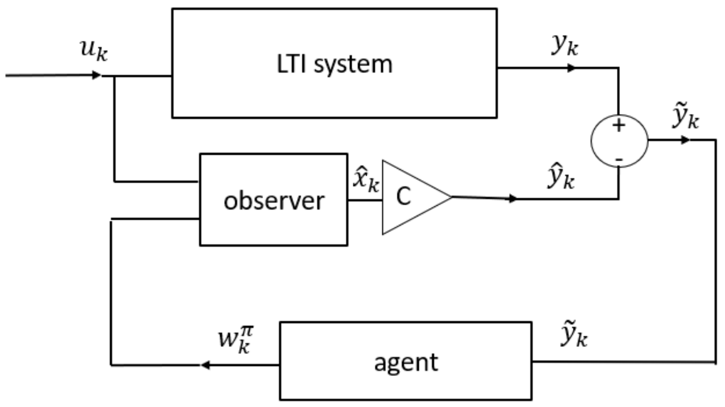


Figure 2.5: Block diagram showing LQE as a RL problem

effort and output estimation error, where $Q > 0$, $R > 0$ are weighting matrices to be chosen.

$$V^\pi(\tilde{x}_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \left(\tilde{y}_i^T Q \tilde{y}_i + (w_i^\pi)^T R w_i^\pi \right) \tag{2.39}$$

Therefore, the goal is to solve the following optimization problem

$$\begin{aligned} \arg\min_{w_k^\pi} \quad & \sum_{i=k}^{\infty} \gamma^{i-k} \left(\tilde{y}_i^T Q \tilde{y}_i + (w_i^\pi)^T R w_i^\pi \right) \\ s.t. \quad & \tilde{x}_{i+1} = A\tilde{x}_i - w_i^\pi \\ & \tilde{y}_i = C\tilde{x}_i \end{aligned} \tag{2.40}$$

which can be rewritten as

$$\begin{aligned} \arg\min_{w_k^\pi} \quad & \sum_{i=k}^{\infty} \gamma^{i-k} \left(\tilde{x}_i^T \bar{Q} \tilde{x}_i + (w_i^\pi)^T R w_i^\pi \right) \\ s.t. \quad & \tilde{x}_{i+1} = A\tilde{x}_i - I w_i^\pi \end{aligned} \tag{2.41}$$

where $\bar{Q} = C^T Q C$, I is the identity matrix. Note that $(A, -I)$ is controllable. Therefore, the optimization problem (2.41) can be considered as an LQR problem for the observer error dynamics. As a result, the optimal correction term can be obtained by solving the discrete-time ARE

$$P^{\pi} = \bar{Q} + \gamma A^T P^{\pi} A - \gamma^2 (A^T P^{\pi})(R + \gamma P^{\pi})^{-1} P^{\pi} A \tag{2.42}$$

where P^{π} is the solution to the discrete-time ARE, and the optimal correction term is given by [44]

$$w_k^{\pi} = -K^{\pi} \tilde{x}_k \tag{2.43}$$

where $K^{\pi} = -\gamma(R + \gamma P^{\pi})^{-1} P^{\pi} A$.

The LQE problem can be considered as a RL problem as shown by the block diagram in Fig 2.5, where w_k^{π} is the action, $\tilde{y}_k$ is the observed state, the LTI system and the observer make the environment, and $\tilde{x}_k$ is the state.

The value-function has the quadratic form

$$V^{\pi}(\tilde{x}_k) = \tilde{x}_k^T P^{\pi} \tilde{x}_k \tag{2.44}$$

2.3 Two-layer Quadratic Neural Networks

This section discusses the training of a two-layer QNN with one output using convex optimization. Consider the neural network in Fig 2.6 with one hidden layer, one output, and a degree two polynomial activation function, where $\mathcal{X}_i \in \mathbb{R}^n$ is the i-th input data given to the neural network, $\hat{\mathcal{Y}}_i \in \mathbb{R}$ is the output of the neural network for the input $\mathcal{X}_i$, $\mathcal{Y}_i \in \mathbb{R}$ is the output label corresponding to the input $\mathcal{X}_i$, L is the number of hidden neurons, $f(z) = az^2 + bz + c$ is the polynomial activation function, and $a \neq 0$, b, c are pre-defined constant coefficients. The notation w_{kj} represents the weight from the k-th input-neuron to the j-th hidden-neuron, and ρ_j represents the weight from the j-th hidden-neuron to the output.

The input-output mapping of the neural network is

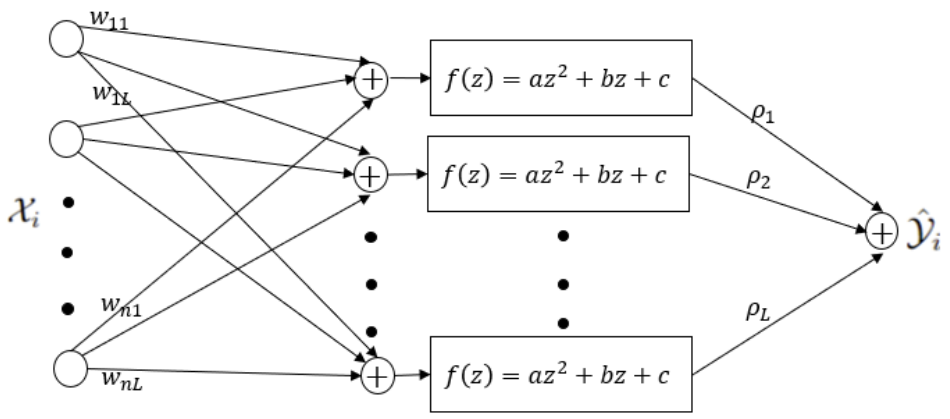


Figure 2.6: Two-layer QNN with one output

$$\hat{\mathcal{Y}}_i = \sum_{j=1}^{L} f(\mathcal{X}_i^T W_j)\rho_j \tag{2.45}$$

where $W_j = \begin{bmatrix} w_{1j} & w_{2j} & \dots & w_{nj} \end{bmatrix}^T$.

2.3.1 Training the QNN with a Convex Optimization

Reference [17] proposes the training optimization

$$\begin{aligned} \min_{W_k,\rho_k} \; & l(\hat{\mathcal{Y}} - \mathcal{Y}) + \beta \sum_{j=1}^{L} |\rho_j| \\ s.t. \; & \hat{\mathcal{Y}}_i = \sum_{j=1}^{L} f(\mathcal{X}_i^T W_j)\rho_j \\ & \|W_k\|_2 = 1, \quad k = 1,2,...,L \quad i = 1,2,...,N \end{aligned} \tag{2.46}$$

where $\beta \geq 0$ is a pre-defined regularization parameter, $l(.)$ is a convex loss function, N is the number of data points, $\hat{\mathcal{Y}} = \begin{bmatrix} \hat{\mathcal{Y}}_1 & \hat{\mathcal{Y}}_2 & \dots & \hat{\mathcal{Y}}_N \end{bmatrix}^T$, and $\mathcal{Y} = \begin{bmatrix} \mathcal{Y}_1 & \mathcal{Y}_2 & \dots & \mathcal{Y}_N \end{bmatrix}^T$.

The optimization problem (2.46) can be equivalently solved by the dual convex optimization

$$\begin{aligned}
\min_{Z^+,Z^-} \quad & l(\hat{\mathcal{Y}} - \mathcal{Y}) + \beta\, Trace(Z_1^+ + Z_1^-) \\
s.t. \quad & \hat{\mathcal{Y}}_i = a\mathcal{X}_i^T(Z_1^+ - Z_1^-)\mathcal{X}_i + b\mathcal{X}_i^T(Z_2^+ - Z_2^-) + c\, Trace(Z_1^+ - Z_1^-), \\
& Z^+ = \begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & Trace(Z_1^+) \end{bmatrix} \geq 0, \quad Z^- = \begin{bmatrix} Z_1^- & Z_2^- \\ (Z_2^-)^T & Trace(Z_1^-) \end{bmatrix} \geq 0, \\
& i = 1, 2, \ldots N
\end{aligned} \tag{2.47}$$

where Z_1^+, Z_2^+, Z_1^-, Z_2^- are optimization parameters [17]. After training the neural network and obtaining Z^+, Z^- from (2.47), the quadratic input-output mapping is

$$\hat{\mathcal{Y}}_i = \begin{bmatrix} \mathcal{X}_i \\ 1 \end{bmatrix}^T H \begin{bmatrix} \mathcal{X}_i \\ 1 \end{bmatrix} \tag{2.48}$$

where

$$H = \begin{bmatrix} a(Z_1^+ - Z_1^-) & 0.5b(Z_2^+ - Z_2^-) \\ 0.5b(Z_2^+ - Z_2^-)^T & c\left[Trace\left(Z_1^+ - Z_1^-\right)\right] \end{bmatrix}$$

Remark 3. *If* $b = c = 0$, *and* $a = 1$ *are chosen, the input-output mapping is*

$$\hat{\mathcal{Y}}_i = \mathcal{X}_i^T(Z_1^+ - Z_1^-)\mathcal{X}_i \tag{2.49}$$

Hence, selecting a QNN with $b = c = 0$ *and* $a = 1$ *as the VFA for LQ problems leads to the least number of parameters required for approximation. As a result, this thesis adopts this parameter choice.*

Remark 4. *Selecting a lower value for* β *is an approach to decrease the cost* $l(\hat{\mathcal{Y}} - \mathcal{Y})$. *However, it's essential to be cautious, as a decreased* β *can potentially result in overfitting, making the neural network more sensitive to noise. [16][45]*

Algorithm 3 Decomposition with Tolerance **tol**

```
procedure DECOMPOSITION($Z$, $tol$)
    Output: list $V$
    Calculate the rank-1 decomposition $Z = \sum_{j=1}^{r} q_j q_j^T$ using eigenvector decomposition,
        retaining eigenvectors corresponding to eigenvalues surpassing the threshold $tol$.
    Create a list of vectors $q = \{q_1, \ldots, q_r\}$
    for $k = 1, \ldots, r-1$ do
        $q_1 \leftarrow q_k$
        if $q_1^T G q_1 = 0$ then
            $v \leftarrow q_1$
        else
            find $j \in \{k+1, \ldots, r\}$ such that $(q_1^T G q_1)(q_j^T G q_j) < 0$ and set $v = \frac{q_1 + \Gamma q_j}{\sqrt{1+\Gamma^2}}$ where
                $$G = \begin{bmatrix} I_n & 0 \\ 0 & -1 \end{bmatrix}, \ \Gamma = \frac{-2q_1^T G q_j + \sqrt{\triangle}}{2q_j^T G q_j}, \quad \triangle = 4\left[\left(q_1^T G q_j\right)^2 - \left(q_1^T G q_1\right)\left(q_j^T G q_j\right)\right], \tag{2.50}$$
            and $I_n$ represents an $n \times n$ identity matrix.
        end if
        Remove $q_k$ from list $q$ and insert $\frac{q_j - \Gamma q_1}{\sqrt{1+\Gamma^2}}$ at the end of the list
        Add $v$ to the list $V$
    end for
    Add last element of list $q$ to list $V$
    Return $V$
end procedure
```

2.3.2 Obtaining the Optimal Weights

One can also adopt the neural decomposition method outlined in references [17][16] to derive the weights of the trained two-layer QNN and determine the optimal number of hidden neurons with Algorithm 3.

In order to acquire the weights, the method involves providing both Z^+ and Z^- matrices as Z to the Algorithm 3. The outcomes of Decomposition(Z^+, tol) and Decomposition(Z^-, tol) are denoted as $\{v_1, \ldots v_{r^+}\}$ and $\{v_{r^++1}, \ldots v_{r^++r^-}\}$, respectively, where

$$v_j = \begin{bmatrix} s_j \\ f_j \end{bmatrix}, \quad s_j \in \mathbb{R}^n, \quad f_j \in \mathbb{R} \tag{2.51}$$

The weights can be obtained by

$$W_j = \frac{s_j}{\|s_j\|_2}, \quad \rho_j = \begin{cases} (f_j)^2, & j \in \{1, 2, \ldots, r^+\} \\ -(f_j)^2, & \text{otherwise} \end{cases} \tag{2.52}$$

The optimal number of hidden neurons is

$$L = r^+ + r^- \tag{2.53}$$

Chapter 3

Linear Quadratic Controller with Quadratic Neural Networks and Reinforcement Learning

In section 2.2, we covered discrete-time LQR and LQT design using conventional methodologies, emphasizing the necessity of a precise system model. This section presents the proposed PI algorithm, which utilizes a two-layer QNN as VFA to design LQR and LQT in the absence of a system model.

3.1 LQR Design

This section designs LQR using the proposed PI algorithm that does not require the system model. The key contribution is to perform PI by designing the policy evaluator as a two-layer QNN. This network is trained through convex optimization. To the best of our knowledge, this is the first time that a QNN trained through convex optimization is employed as the Q-function approximator (QFA). The main advantage is that the QNN's input-output mapping has an analytical expression as a quadratic form, which can then be used to obtain an analytical expression for policy improvement. This is in stark contrast to available techniques in the literature that must train a second neural network to obtain the policy improvement. The thesis establishes the convergence of the learning algorithm to the optimal control provided the system is controllable and one starts from a stabilitzing control policy. A pendulum and a quadrotor example demonstrate the effectiveness of the proposed approach.

3.1.1 Problem Statement

An unknown controllable linear system model is written as

$$x_{k+1} = Ax_k + Bu_k \tag{3.1}$$

where $x_k \in \mathbb{R}^{n_x}$ is the state vector, $u_k \in \mathbb{R}^{n_u}$ is the input vector and A, B are unknown matrices constrained to be such that the pair (A, B) is controllable. Define the policy $\pi(.)$ as a linear mapping from the state vector to the input vector as

$$u_k = \pi(x_k) = -K^{\pi} x_k \tag{3.2}$$

where K^{π} is a matrix to be determined by the designer and x_k is measured without noise and is available for feedback.

Define the local cost as

$$c(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k \tag{3.3}$$

where Q, and R are both positive definite. The value function when one follows the control policy $\pi(.)$ is defined as

$$V^{\pi}(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \left(x_i^T Q x_i + u_i^T R u_i \right) \tag{3.4}$$

where $0 < \gamma \leq 1$ is a discount factor.

The objective is to obtain the optimal policy $\pi^*(.)$ that minimizes $V^\pi(x_k)$ for all states x_k subject to the unknown dynamics of the system. This is done using QNNs as a VFA.

3.1.2 Policy Evaluation

This section presents how to perform policy evaluation by designing a two-layer QNN. The Bellman equation can be written as:

$$Q^\pi(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + \gamma Q^\pi(x_{k+1}, u_{k+1}) \tag{3.5}$$

As discussed in section 2.2, the Q-function has a quadratic form. As a result, a two-layer QNN with coefficients $b = c = 0$, $a = 1$ is the perfect candidate to approximate the Q-function. The policy evaluation step is as follows:

Obtain H^π for the stabilizing policy $\pi(.)$, solving:

$$\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H^\pi \begin{bmatrix} x_k \\ u_k \end{bmatrix} = x_k^T Q x_k + u_k^T R u_k + \gamma \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}^T H^\pi \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix} \tag{3.6}$$

Remark 5. *Note that* (3.6) *is a scalar equation, H^π is symmetric, and*

$$\begin{bmatrix} x_k \\ u_k \end{bmatrix} \in \mathbb{R}^{n_x + n_u} \tag{3.7}$$

Therefore, the matrix H^π has $M = \frac{(n_x+n_u)(n_x+n_u+1)}{2}$ unknown independent elements and $N \geq M$ data samples are needed to obtain H^π from equation (3.6)*.*

We propose to obtain H^π in the policy evaluation step by employing the iterative equation (3.8). The proof of convergence is presented in Lemma 1.

Lemma 1. *One can obtain H^π for the stabilizing policy $\pi(.)$ using the iterative equation* (3.8) *starting with any initial value H_0^π with guaranteed convergence to the limit $H_i^\pi \to H^\pi$.*

$$\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H_{i+1}^\pi \begin{bmatrix} x_k \\ u_k \end{bmatrix} = x_k^T Q x_k + u_k^T R u_k + \gamma \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^\pi \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix} \tag{3.8}$$

Proof. The proof follows the same strategy of the proof of Lemma 1 in reference [14]. Applying equation (3.8) recursively yields

$$\begin{aligned} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H_{i+1}^\pi \begin{bmatrix} x_k \\ u_k \end{bmatrix} =& c(x_k, u_k) + \gamma \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^\pi \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix} \\ =& c(x_k, u_k) + \gamma c(x_{k+1}, u_{k+1}) + \gamma^2 \begin{bmatrix} x_{k+2} \\ u_{k+2} \end{bmatrix}^T H_{i-1}^\pi \begin{bmatrix} x_{k+2} \\ u_{k+2} \end{bmatrix} \\ =& \sum_{j=0}^{i} \gamma^j c(x_{k+j}, u_{k+j}) + \gamma^{i+1} \begin{bmatrix} x_{k+i+1} \\ u_{k+i+1} \end{bmatrix}^T H_0^\pi \begin{bmatrix} x_{k+i+1} \\ u_{k+i+1} \end{bmatrix} \end{aligned} \tag{3.9}$$

It should be noted that the policy $\pi(.)$ is a stabilizing policy, therefore $\lim_{i\to\infty} x_{k+i+1} = 0$ and $\lim_{i\to\infty} u_{k+i+1} = 0$ for all k. Consequently, for any H_0^π,

$$\lim_{i\to\infty} \gamma^{i+1} \begin{bmatrix} x_{k+i+1} \\ u_{k+i+1} \end{bmatrix}^T H_0^\pi \begin{bmatrix} x_{k+i+1} \\ u_{k+i+1} \end{bmatrix} = 0 \tag{3.10}$$

and therefore

$$\begin{aligned} \lim_{i\to\infty} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H_{i+1}^\pi \begin{bmatrix} x_k \\ u_k \end{bmatrix} &= \sum_{j=0}^{\infty} \gamma^j c(x_{k+j}, u_{k+j}) = \\ &= c(x_k, u_k) + \gamma Q^\pi(x_{k+1}, u_{k+1}) \end{aligned} \tag{3.11}$$

Thus, from equations (3.5), (3.6) and (3.11) $H_i^\pi \to H^\pi$ when $i \to \infty$. $\square$

Remark 6. *In practice the condition*

$$\|H_i^\pi - H_{i-1}^\pi\| < \epsilon, \tag{3.12}$$

is used as the stopping criterium of the algorithm.

Due to the implications of Lemma 1, the problem of policy evaluation transforms into the task of computing] H_{i+1}^π from equation (3.8) given H_i^π. This can be done by training a two-layer QNN as follows.

Since it is assumed that one has access to the state x_k, one can calculate $\mathcal{Y}_k$ defined as

$$\mathcal{Y}_k = x_k^T Q x_k + u_k^T R u_k + \gamma \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^\pi \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}. \tag{3.13}$$

Therefore, from (3.8),

$$\mathcal{X}_k^T H_{i+1}^\pi \mathcal{X}_k = \mathcal{Y}_k \tag{3.14}$$

where $\mathcal{X}_k^T = \begin{bmatrix} x_k^T & u_k^T \end{bmatrix}$. Thus, H_{i+1}^π can be obtained from the training of a two-layer QNN as the solution of the convex optimization (2.47) using a set of input data points $\mathcal{X}_k$ and the corresponding labels $\mathcal{Y}_k$, as well as the coeffficients $a = 1,\ b = 0,\ c = 0$.

3.1.3 Policy Improvement

In this section, the policy improvement step is addressed for the stabilizing policy $\pi(\cdot)$ using H^π from the policy evaluation. We first partition H^π as

$$\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H^\pi \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H_{xx}^\pi & H_{xu}^\pi \\ (H_{xu}^\pi)^T & H_{uu}^\pi \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{3.15}$$

Lemma 2. *The policy improvement for the stabilizing policy $\pi(.)$ is given by*

$$\pi'(x_k) = -(H^{\pi}_{uu})^{-1}(H^{\pi}_{xu})^T x_k \tag{3.16}$$

Proof. The improved policy $\pi'(.)$ is obtained by

$$\pi'(x_k) = \arg\min_{u_k} Q^{\pi}(x_k, u_k) \tag{3.17}$$

Since $Q^{\pi}(x_k, u_k)$ is a quadratic form, the ecessary and sufficient conditions of optimality are

$$\begin{aligned} \frac{\partial Q^{\pi}(x_k, u_k)}{\partial u_k} &= 0 \\ \frac{\partial^2 Q^{\pi}(x_k, u_k)}{\partial u_k^2} &> 0 \end{aligned} \tag{3.18}$$

The Q-function can be written as

$$\begin{aligned} Q^{\pi}(x_k, u_k) = \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H^{\pi}_{xx} & H^{\pi}_{xu} \\ (H^{\pi}_{xu})^T & H^{\pi}_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \\ x_k^T H^{\pi}_{xx} x_k + x_k^T H^{\pi}_{xu} u_k + u_k^T (H^{\pi}_{xu})^T x_k + u_k^T H^{\pi}_{uu} u_k \end{aligned} \tag{3.19}$$

Therefore, the solution to the first constraint yields

$$\begin{aligned} \frac{\partial Q^{\pi}(x_k, u_k)}{\partial u_k} = 0 \Leftrightarrow H^{\pi}_{uu} u_k + (H^{\pi}_{xu})^T x_k = 0 \Leftrightarrow \\ u_k = -(H^{\pi}_{uu})^{-1}(H^{\pi}_{xu})^T x_k \end{aligned} \tag{3.20}$$

Note that the matrix H^{π} is positive definite. Therefore, all matrices on the main diagonal of H^{π}, including H^{π}_{uu}, are also positive definite. As a result, the inverse of H^{π}_{uu} does exist. The second constraint in (3.18) is thus satisfied since $H^{\pi}_{uu} > 0$. Therefore, the policy obtained in (3.20) is the unique minimizer $\pi'(x_k)$.

□

3.1.4 The Proposed Algorithm

We have presented a detailed explanation of the policy evaluation and policy improvement procedures. The complete algorithm to design the LQR is presented in Algorithm 4.

Algorithm 4 Designing LQR with QNN

```
Choose $\epsilon$, $N$, $\gamma$.
Select the initial stabilizing policy $\pi_0$. Then, for $j = 0, 1, 2, \ldots$ perform policy evaluation and policy improvement steps until convergence

**Policy Evaluation:**
    $i \leftarrow 0$
    Choose a random $H_0^{\pi_j}$
    **repeat**
        Train the QNN by $N$ data samples with $\mathcal{X}_k$ as the inputs and $\mathcal{Y}_k$ as the output labels with

            $$\mathcal{Y}_k = x_k^T Q x_k + u_k^T R u_k + \gamma \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^{\pi} \begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix}, \quad \mathcal{X}_k = \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{3.21}$$

        Obtain the input-output mapping as

            $$\mathcal{X}_k^T H \mathcal{X}_k = \hat{\mathcal{Y}}_k \tag{3.22}$$

        $i \leftarrow i + 1$
        $H_i^{\pi_j} \leftarrow H$
    **Until** $||H_i^{\pi_j} - H_{i-1}^{\pi_j}|| < \epsilon$
    $H^{\pi_j} \leftarrow H_i^{\pi_j}$
**Policy Improvement:**
    Obtain $\pi_{j+1}$ such that

            $$\pi_{j+1}(x_k) = -(H_{uu}^{\pi_j})^{-1}(H_{xu}^{\pi_j})^T x_k \tag{3.23}$$
```

Figure 3.1 presents an overview of the proposed algorithm.

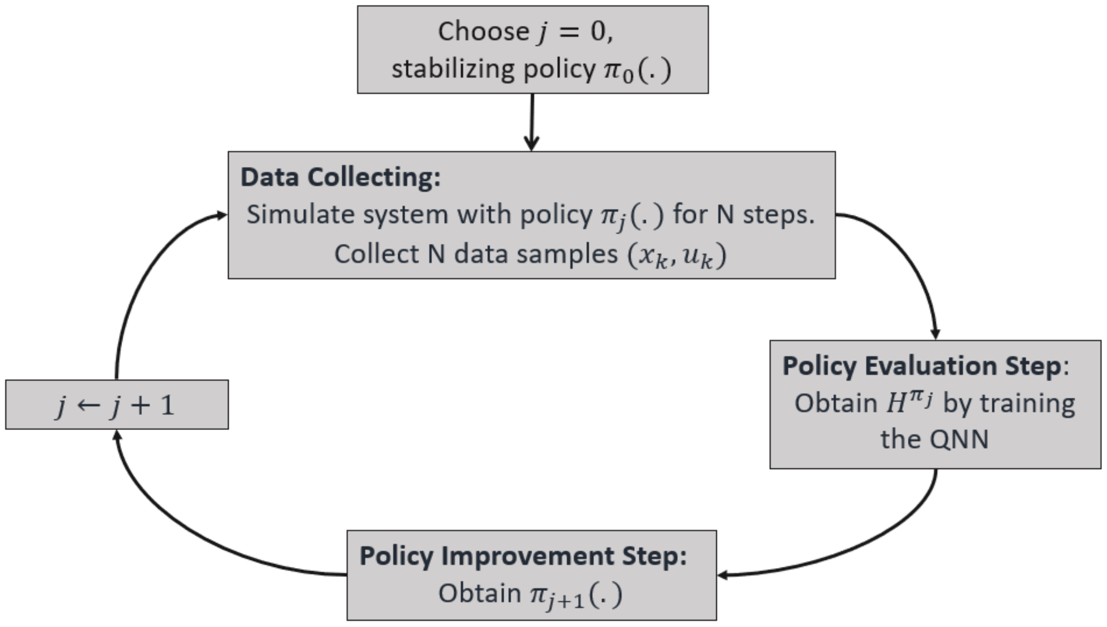

Choose $j = 0$, stabilizing policy $\pi_0(.)$
Data Collecting: Simulate system with policy $\pi_j(.)$ for N steps. Collect N data samples (x_k, u_k)
Policy Evaluation Step: Obtain H^{π_j} by training the QNN
Policy Improvement Step: Obtain $\pi_{j+1}(.)$
$j \leftarrow j + 1$

Figure 3.1: Overview of LQR proposed design

3.1.5 Simulations

This thesis applies Algorithm 4 to address both controlling a simple pendulum and a quadrotor flying at a constant altitude and direction. It is shown that the control policy converges to the LQR solved by conventional methods.

As an illustrative example, we also determine the optimal weights of the QNN for the pendulum's optimal value-function, benefiting from its limited number of weights.

3.1.5.1 Pendulum Example

The simple pendulum is given in Fig 3.2, where g is the gravitational acceleration, l is the length, m is the bob's mass, and $\theta(t)$ is the angle between the pendulum and the vertical axis.

The pendulum can be modeled by:

$$ml^2\ddot{\theta}\left(t\right) + b\dot{\theta}\left(t\right) + mgl\ sin\left(\theta\left(t\right)\right) = u\left(t\right) \tag{3.24}$$

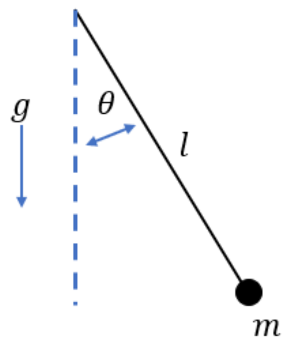


Figure 3.2: The simple pendulum

where $u(t)$ is the torque input, and b is the damping coefficient. Assume $g = 10\,\frac{m}{s^2}$, $m = 1\,kg$, $l = 1\,m$, $b = 0.1\,\frac{kg.m^2}{s}$, and $sin(\theta(t)) \approx \theta(t)$.

Hence, the pendulum can be modeled in state-space as

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -10 & -0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \tag{3.25}$$

where $x_1(t) = \theta(t)$, $x_2(t) = \dot{\theta}(t)$ with the chosen initial state

$$\begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}. \tag{3.26}$$

The proposed method is applied to the pendulum case with sampling time $T_s = 0.05$ seconds. To compare the results of the proposed method with the optimal controller, we need to obtain a discrete-time model. The discretized model given in equation (3.27) with sampling time T_s is calculated using Tustin's method [53].

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \begin{bmatrix} 0.99 & 0.05 \\ -0.50 & 0.98 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \begin{bmatrix} 0.0012 \\ 0.0497 \end{bmatrix} u_k \tag{3.27}$$

Simulation number	K^{π_0}
Simulation 1	$\begin{pmatrix}-0.84 & 4.75\end{pmatrix}$
Simulation 2	$\begin{pmatrix}-0.35 & 1.58\end{pmatrix}$
Simulation 3	$\begin{pmatrix}-0.11 & 0.45\end{pmatrix}$
Simulation 4	$\begin{pmatrix}-0.01 & 0.14\end{pmatrix}$
Simulation 5	$\begin{pmatrix}-0.61 & 3.99\end{pmatrix}$
Simulation 6	$\begin{pmatrix}-0.31 & 2.11\end{pmatrix}$
Simulation 7	$\begin{pmatrix}-1.41 & 5.30\end{pmatrix}$
Simulation 8	$\begin{pmatrix}1.57 & 2.18\end{pmatrix}$
Simulation 9	$\begin{pmatrix}0.14 & 1.30\end{pmatrix}$
Simulation 10	$\begin{pmatrix}3.13 & 2.83\end{pmatrix}$

Table 3.1: LQR pendulum - The random initial stabilizing policies

We choose $\beta = 0.005$, $\gamma = 1$, $N = 100$, $Q = \begin{bmatrix} 5 & 3 \\ 3 & 10 \end{bmatrix}$, $R = 20$ and rewrite the policy $\pi_j(.)$ as:

$$\pi_j(x_k) = \begin{bmatrix} k_1^{\pi_j} & k_2^{\pi_j} \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \tag{3.28}$$

The optimal controller and the optimal value-function can be obtained by the method detailed in section 2.2 as follows

$$P^{\pi^*} = \begin{bmatrix} 137 & -23 \\ -23 & 36 \end{bmatrix} \tag{3.29}$$

$$\pi^*(x_k) = \begin{bmatrix} k_1^* & k_2^* \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} = \begin{bmatrix} 0.0738 & -0.0674 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \tag{3.30}$$

Therefore, it is expected that with the increase in policy number j, $k_1^{\pi_j}$ converges to 0.0738 and $k_2^{\pi_j}$ converges to -0.0674 starting from any stabilizing policy $\pi_0(.)$. To show the convergence for any initial stabilizing policy $\pi_0(.)$, we have started the simulation with ten random initial stabilizing policies. Fig 3.3 shows that in all simulations $\pi_j(.)$ converges to the optimal policy before policy $j = 4$. The random initial stabilizing policies used are

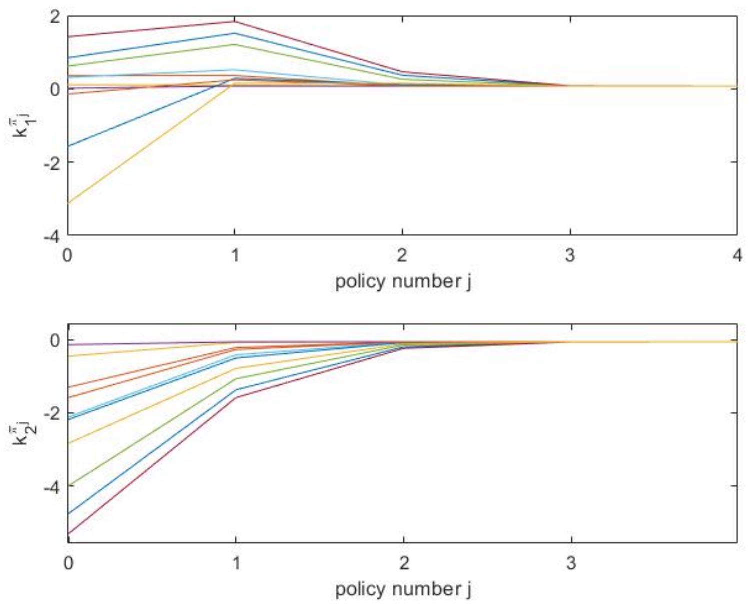


Figure 3.3: Convergence to LQR controller for the pendulum

given in the table 3.1.

Next, the weights to approximate the optimal value-function are calculated. This reveals both the optimal architecture and the ideal number of hidden neurons for the two-layer QNN. After obtaining the optimal policy, Z^+ and Z^- are obtained as:

$$Z^+ = \begin{bmatrix} 138.8 & -23.7 & -1.5 & 0 \\ -23.7 & 36.7 & 1.3 & 0 \\ -1.5 & 1.3 & 20.1 & 0 \\ 0 & 0 & 0 & 195.7 \end{bmatrix}, \quad Z^- = 10^{-7} \times \begin{bmatrix} 0.055 & 0.001 & 0.0001 & 0 \\ 0.001 & 0.056 & -0.0001 & 0 \\ 0.0001 & -0.0001 & 0.057 & 0 \\ 0 & 0 & 0 & 0.168 \end{bmatrix}$$

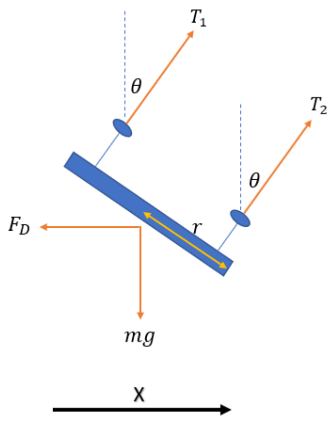


Figure 3.4: Free body diagram of the quadrotor

Hence, the weights of the QNN are given as follows (see algorithm 3 in chapter 2):

$$W = \begin{bmatrix} 0.005 & 0.214 & 0.916 & 0.994 & -0.866 & -0.190 & 0.491 & -0.538 \\ 0.088 & 0.958 & -0.394 & -0.028 & 0.496 & -0.981 & 0.275 & -0.219 \\ -0.996 & 0.188 & 0.072 & -0.099 & 0.041 & -0.027 & -0.826 & -0.813 \end{bmatrix}$$

$$\rho = \begin{bmatrix} 18.11 & 27.06 & 75.26 & 75.26 & 10^{-8} \times -0.41 & 10^{-8} \times -0.42 & 10^{-8} \times -0.42 & 10^{-8} \times -0.42 \end{bmatrix} \tag{3.31}$$

where $W = \begin{bmatrix} W_1 & W_2 & \dots & W_8 \end{bmatrix}$, $\rho = \begin{bmatrix} \rho_1 & \rho_2 & \dots & \rho_8 \end{bmatrix}$. Note that the optimal number of hidden neurons is $L = 8$.

3.1.5.2 Quadrotor Example

To provide another practical example with increased number of states, we use a quadcopter moving in a straight line with a constant altitude. The free body diagram of the quadrotor is given in Fig 3.4, where F_D is the viscous drag force, m is the quadrotor's mass, g is the gravitational acceleration, $\theta(t)$ is the pitch angle, r is the distance from each propeller to the center of mass, $T_1(t)$ and $T_2(t)$ are the thrust forces produced by the left and the right propellers, respectively.

The equations of motion are as follows

$$
\begin{aligned}
(T_1 + T_2)sin(\theta) - F_D &= m\dot{v}_x \\
(T_1 + T_2)cos(\theta) &= mg \\
(T_1 - T_2)r &= I\dot{w}
\end{aligned}
\tag{3.32}
$$

where I represents the moment of inertia, w denotes the pitch rate of change, and v_x is the speed of the quadcopter. The viscous drag force can be modeled as

$$F_D = C_D(v_x - v_w) \tag{3.33}$$

where C_D is the coefficient of the drag, and v_w is the wind speed.

The quadcopter's behavior can be modeled by the state-space

$$
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{C_D}{m} & g & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{r}{I} \end{bmatrix} u + \begin{bmatrix} 0 \\ \frac{C_D v_w}{m} \\ 0 \\ 0 \end{bmatrix}
\tag{3.34}
$$

where x_1 is the horizontal position, x_2 is the horizontal speed, x_3 is the pitch angle, x_4 is the pitch rate of change, and the system input is $u = T_1 - T_2$. Assume $v_w = 0\,\frac{m}{s}$, $C_D = 0.2$, $m = 2\,kg$, $g = 10\,\frac{m}{s^2}$, $I = 0.046\,kg.m^2$, and $r = 0.2\,m$.

As a result, the state-space of the quadrotor is

$$
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.1 & 10 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4.35 \end{bmatrix} u(t)
\tag{3.35}
$$

where the initial state is

$$\begin{bmatrix} x_1(0) \\ x_2(0) \\ x_3(0) \\ x_4(0) \end{bmatrix} = \begin{bmatrix} -10 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.36}$$

Take note that this initial state implies that the objective is to move ten meters forward.

The proposed method is applied for the quadrotor with sampling time $T = 0.1$ seconds. The discretized model (3.37) with sampling timg $T = 0.1$ seconds is obtained using Tustin's method.

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \\ x_{3,k+1} \\ x_{4,k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0.05 & 0.003 \\ 0 & 0.99 & 0.99 & 0.05 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \\ x_{4,k} \end{bmatrix} + \begin{bmatrix} 0.001 \\ 0.011 \\ 0.022 \\ 0.435 \end{bmatrix} u_k \tag{3.37}$$

Rewrite the policy $\pi_j(.)$ as

$$\pi_j(x_k) = -\begin{bmatrix} k_1^{\pi_j} & k_2^{\pi_j} & k_3^{\pi_j} & k_4^{\pi_j} \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \\ x_{4,k} \end{bmatrix} \tag{3.38}$$

We choose the following parameter values

$$\beta = 0.005,\ \gamma = 1,\ N = 100,\ R = 1,$$
$$Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \tag{3.39}$$

The optimal controller and value-function are as follows

$$P^{\pi^*} = \begin{bmatrix} 0.95 & 0.77 & 3.36 & 0.24 \\ 0.77 & 9.27 & 38.31 & 2.83 \\ 3.36 & 38.31 & 313.81 & 24.93 \\ 0.24 & 2.83 & 24.93 & 15.82 \end{bmatrix} \tag{3.40}$$

$$\pi^*(x_k) = -\begin{bmatrix} 0.046 & 0.464 & 4.347 & 2.014 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \\ x_{4,k} \end{bmatrix} \tag{3.41}$$

The Fig 3.5 demonstrates the convergence of $k_1^{\pi_j}$, $k_2^{\pi_j}$, $k_3^{\pi_j}$, $k_4^{\pi_j}$ towards their optimal values. To illustrate this convergence, we conducted five simulations using random initial stabilizing policies. The random initial policies are given in the Table 3.2.

Lastly, the trajectory of the quadrotor's position and its speed over time using the optimal policy are depicted in Fig. 3.6.

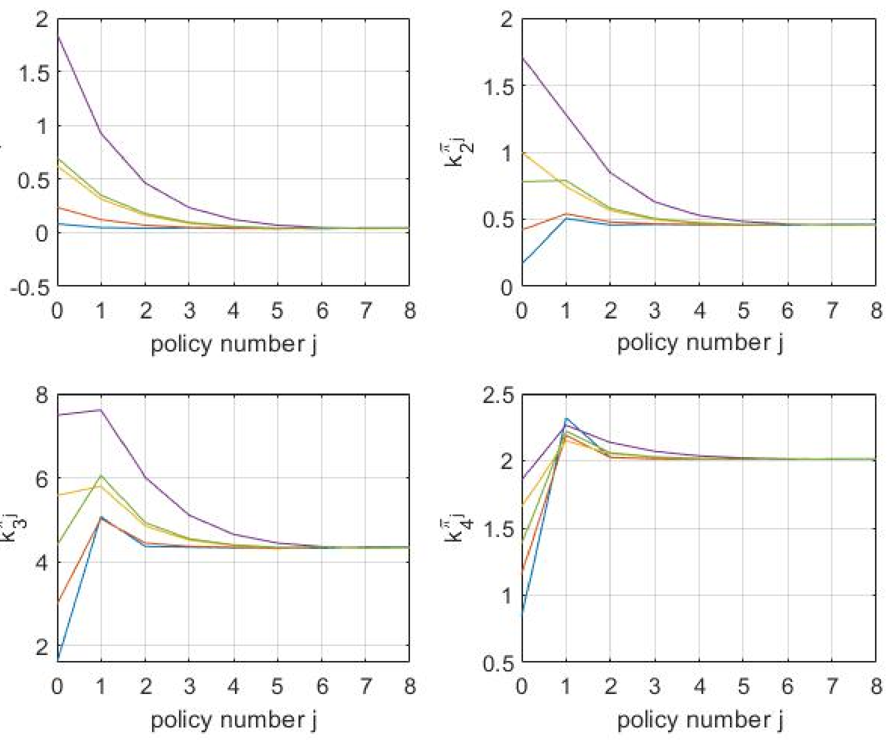


Figure 3.5: Convergence to LQR controller for the quadrotor

Simulation number	K^{π_0}
Simulation 1	$\begin{pmatrix}0.082 & 0.169 & 1.592 & 0.838\end{pmatrix}$
Simulation 2	$\begin{pmatrix}0.236 & 0.420 & 2.978 & 1.156\end{pmatrix}$
Simulation 3	$\begin{pmatrix}0.626 & 0.998 & 5.578 & 1.660\end{pmatrix}$
Simulation 4	$\begin{pmatrix}1.851 & 1.709 & 7.491 & 1.858\end{pmatrix}$
Simulation 5	$\begin{pmatrix}0.701 & 0.781 & 4.380 & 1.379\end{pmatrix}$

Table 3.2: LQR quadrotor - The random initial stabilizing policies

3.2 LQT Design

The LQT is an extension of the LQR problem. The LQT objective is to stabilize the LTI system while making sure the output y_k tracks a desired reference r_k. We formulate an LQT as an LQR problem to design the LQT with the Algorithm 4. It is assumed that the reference trajectory r_k has the linear dynamic (3.42) for some matrix F.

$$r_{k+1} = Fr_k \tag{3.42}$$

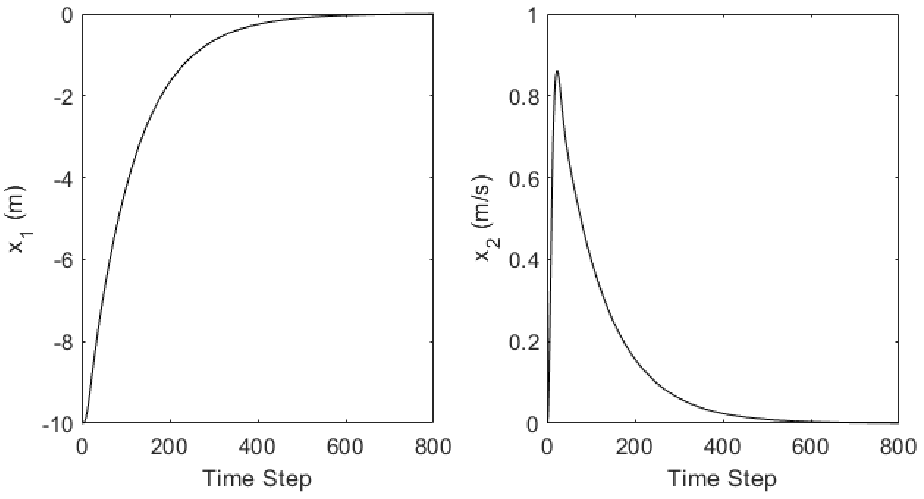


Figure 3.6: Position and speed trajectory using the optimal policy

3.2.1 LQT as an LQR Problem

An unknown controllable, observable linear system model is written as

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{3.43}$$

where $x_k \in \mathbb{R}^{n_x}$ is the state vector, $u_k \in \mathbb{R}^{n_u}$ is the input vector, $y_k \in \mathbb{R}^{n_y}$ is the output vector and A, B, C are unknown matrices constrained to be such that the pair (A, B) is controllable and (A, C) is observable.

The goal is to find a series of control inputs that minimizes the quadratic value-function as follows

$$V^{\pi}(x_k, r_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \left((Cx_i - r_i)^T Q \, (Cx_i - r_i) + u_i^T R u_i \right) \tag{3.44}$$

where $0 < \gamma \leq 1$ is the discount factor, $Q > 0$, $R > 0$.

Define the augmented state $X_k \in \mathbb{R}^{n_x + n_y}$ as

$$X_k = \begin{bmatrix} x_k \\ r_k \end{bmatrix} \tag{3.45}$$

and construct the augmented system as

$$X_{k+1} = \bar{A}X_k + \bar{B}u_k \tag{3.46}$$

where

$$\bar{A} = \begin{bmatrix} A & 0 \\ 0 & F \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{3.47}$$

As a result, the objective is to solve the following optimization problem

$$\begin{aligned} \arg\min_{u_i} \quad & \sum_{i=k}^{\infty} \gamma^{i-k} \left(X_i^T \bar{Q} X_i + u_i^T R u_i \right) \\ s.t. \quad & X_{i+1} = \bar{A}X_i + \bar{B}u_i \end{aligned} \tag{3.48}$$

where $\bar{Q} = \begin{bmatrix} C & -I \end{bmatrix}^T Q \begin{bmatrix} C & -I \end{bmatrix}$ and I is the identity matrix. Hence, the LQT problem can be considered as an LQR problem with the state X_k.

The value-function and the input policy can be written as

$$\begin{aligned} V^\pi(X_k) &= X_k^T P^\pi X_k \\ Q^\pi(X_k, u_k) &= \begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H^\pi \begin{bmatrix} X_k \\ u_k \end{bmatrix} \\ u_k &= \pi(X_k) = -K^\pi X_k \end{aligned} \tag{3.49}$$

for some matrices

$$P^\pi \in \mathbb{R}^{(n_x+n_y)\times(n_x+n_y)},\ H^\pi \in \mathbb{R}^{(n_x+n_y+n_u)\times(n_x+n_y+n_u)},\ K^\pi \in \mathbb{R}^{n_u\times(n_x+n_y)} \tag{3.50}$$

We assume x_k and r_k are measured without noise and are available for feedback.

3.2.2 Policy Evaluation

The policy evaluation step is as follows

Obtain H^{π} for the stabilizing policy $\pi(.)$, such that

$$\begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H^{\pi} \begin{bmatrix} X_k \\ u_k \end{bmatrix} = c(X_k, u_k) + \gamma \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}^T H^{\pi} \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix} \tag{3.51}$$

where

$$c(X_k, u_k) = X_k^T \bar{Q} X_k + u_k^T R u_k \tag{3.52}$$

Remark 7. *Note that* (3.51) *is a scalar equation,* H^{π} *is symmetric, and*

$$\begin{bmatrix} X_k \\ u_k \end{bmatrix} \in \mathbb{R}^{n_x+n_y+n_u} \tag{3.53}$$

Therefore, the matrix H^{π} *has* $M = \frac{(n_x+n_y+n_u)(n_x+n_y+n_u+1)}{2}$ *unknown independent elements and* $N \geq M$ *data samples are needed to obtain* H^{π} *from equation* (3.51)*.*

We suggest calculating H^{π} in the policy evaluation using the iterative equation (3.54). The proof of convergence can be found in Lemma 3.

Lemma 3. H^{π} *for the stabilizing policy* $\pi(\cdot)$ *can be obtained by applying the iterative equation* (3.54)*. Convergence to the limit* $H_i^{\pi} \to H^{\pi}$ *is guaranteed, commencing from an arbitrary initial value* H_0^{π} *provided that* $0 < \gamma < 1$.

$$\begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H_{i+1}^{\pi} \begin{bmatrix} X_k \\ u_k \end{bmatrix} = c(X_k, u_k) + \gamma \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^{\pi} \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix} \tag{3.54}$$

Proof. Applying equation (3.54) recursively yields

$$\begin{aligned}\begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H_{i+1}^{\pi} \begin{bmatrix} X_k \\ u_k \end{bmatrix} &= c(X_k, u_k) + \gamma \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^{\pi} \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix} \\ &= c(X_k, u_k) + \gamma c(X_{k+1}, u_{k+1}) + \gamma^2 \begin{bmatrix} X_{k+2} \\ u_{k+2} \end{bmatrix}^T H_{i-1}^{\pi} \begin{bmatrix} X_{k+2} \\ u_{k+2} \end{bmatrix} \\ &= \sum_{j=0}^{i} \gamma^j c(X_{k+j}, u_{k+j}) + \gamma^{i+1} \begin{bmatrix} X_{k+i+1} \\ u_{k+i+1} \end{bmatrix}^T H_0^{\pi} \begin{bmatrix} X_{k+i+1} \\ u_{k+i+1} \end{bmatrix}\end{aligned} \tag{3.55}$$

For any H_0^{π},

$$\lim_{i \to \infty} \gamma^{i+1} \begin{bmatrix} X_{k+i+1} \\ u_{k+i+1} \end{bmatrix}^T H_0^{\pi} \begin{bmatrix} X_{k+i+1} \\ u_{k+i+1} \end{bmatrix} = 0 \tag{3.56}$$

and therefore

$$\lim_{i \to \infty} \begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H_{i+1}^{\pi} \begin{bmatrix} X_k \\ u_k \end{bmatrix} = \sum_{j=0}^{\infty} \gamma^j c(X_{k+j}, u_{k+j}) \tag{3.57}$$

Also

$$Q^{\pi}(X_k, u_k) = \begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H^{\pi} \begin{bmatrix} X_k \\ u_k \end{bmatrix} = \sum_{j=0}^{\infty} \gamma^j c(X_{k+j}, u_{k+j}) \tag{3.58}$$

Thus, from equations (3.57) and (3.58) $H_i^{\pi} \to H^{\pi}$ when $i \to \infty$. □

The problem of policy evaluation turns into the computation of H_{i+1}^{π} using equation (3.54) with the provided H_i^{π}. This task can be accomplished by training a two-layer QNN as outlined below.

Assuming access to X_k, one can compute $\mathcal{Y}_k$ defined as

$$\mathcal{Y}_k = X_k^T \bar{Q} X_k + u_k^T R u_k + \gamma \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^\pi \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}. \tag{3.59}$$

Therefore, from (3.54),

$$\mathcal{X}_k^T H_{i+1}^\pi \mathcal{X}_k = \mathcal{Y}_k \tag{3.60}$$

where $\mathcal{X}_k^T = \begin{bmatrix} X_k^T & u_k^T \end{bmatrix}$. Thus, H_{i+1}^π can be obtained from the training of a two-layer QNN as the solution of the convex optimization (2.47) using a set of input data points $\mathcal{X}_k$ and the corresponding labels $\mathcal{Y}_k$, as well as the coeffficients $a = 1,\ b = 0,\ c = 0$.

3.2.3 Policy Improvement

We first partition H^π as

$$\begin{bmatrix} X_k \\ u_k \end{bmatrix}^T H^\pi \begin{bmatrix} X_k \\ u_k \end{bmatrix} = \begin{bmatrix} X_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H_{xx}^\pi & H_{xu}^\pi \\ (H_{xu}^\pi)^T & H_{uu}^\pi \end{bmatrix} \begin{bmatrix} X_k \\ u_k \end{bmatrix} \tag{3.61}$$

Lemma 4. *The policy improvement for the stabilizing policy $\pi(.)$ is*

$$\pi'(X_k) = -(H_{uu}^\pi)^{-1}(H_{xu}^\pi)^T X_k \tag{3.62}$$

Proof. The improved policy $\pi'(.)$ is acquired by

$$\pi'(X_k) = \arg\min_{u_k} Q^\pi(X_k, u_k) \tag{3.63}$$

Since $Q^\pi(X_k, u_k)$ is a quadratic form, the necessary and sufficient conditions of optimality are

$$\begin{aligned}\frac{\partial Q^{\pi}(X_k,u_k)}{\partial u_k} &= 0\\ \frac{\partial^2 Q^{\pi}(X_k,u_k)}{\partial u_k^2} &> 0\end{aligned} \tag{3.64}$$

The Q-function can be written as

$$Q^{\pi}(X_k,u_k) = \begin{bmatrix} X_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H^{\pi}_{xx} & H^{\pi}_{xu} \\ (H^{\pi}_{xu})^T & H^{\pi}_{uu} \end{bmatrix} \begin{bmatrix} X_k \\ u_k \end{bmatrix} =$$
$$X_k^T H^{\pi}_{xx} X_k + X_k^T H^{\pi}_{xu} u_k + u_k^T (H^{\pi}_{xu})^T X_k + u_k^T H^{\pi}_{uu} u_k \tag{3.65}$$

Therefore, the solution to the first constraint yields

$$\frac{\partial Q^{\pi}(X_k,u_k)}{\partial u_k} = 0 \Leftrightarrow H^{\pi}_{uu} u_k + (H^{\pi}_{xu})^T X_k = 0 \Leftrightarrow$$
$$u_k = -(H^{\pi}_{uu})^{-1} (H^{\pi}_{xu})^T X_k \tag{3.66}$$

Take note that the matrix H^{π} is a positive definite matrix. Consequently, all matrices situated on the main diagonal of H^{π}, which includes H^{π}_{uu}, are also positive definite. Thus, the inverse of H^{π}_{uu} does indeed exist. The second constraint in (3.64) is satisfied since $H^{\pi}_{uu} > 0$. Therefore, the policy derived in (3.66) is the unique minimizer $\pi'(x_k)$. □

3.2.4 The Proposed Algorithm

The proposed method to design the LQT is presented in Algorithm 5.

Remark 8. *The initial policy $\pi_0(.)$ can be chosen as*

$$\pi_0(x_k, r_k) = -\begin{bmatrix} k^{\pi} & 0 \end{bmatrix} \begin{bmatrix} x_k \\ r_k \end{bmatrix} \tag{3.70}$$

Algorithm 5 Designing LQT with QNN

```
Choose $\epsilon$, $N$, $\gamma$, $F$
Select the initial stabilizing policy $\pi_0$. Then, for $j = 0, 1, 2, \ldots$ perform policy evaluation
and policy improvement steps until convergence

Policy Evaluation:
    $i \leftarrow 0$
    Choose a random $H_0^{\pi_j}$
    repeat
        Train the QNN by $N$ data samples with $\mathcal{X}_k$ as the inputs and $\mathcal{Y}_k$ as the output
            labels with
```

$$\mathcal{Y}_k = X_k^T Q X_k + u_k^T R u_k + \gamma \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}^T H_i^{\pi} \begin{bmatrix} X_{k+1} \\ u_{k+1} \end{bmatrix}, \quad \mathcal{X}_k = \begin{bmatrix} X_k \\ u_k \end{bmatrix} \tag{3.67}$$

```
        Obtain the input-output mapping as
```

$$\mathcal{X}_k^T H \mathcal{X}_k = \hat{\mathcal{Y}}_k \tag{3.68}$$

```
        $i \leftarrow i + 1$
        $H_i^{\pi_j} \leftarrow H$
    Until $||H_i^{\pi_j} - H_{i-1}^{\pi_j}|| < \epsilon$
    $H^{\pi_j} \leftarrow H_i^{\pi_j}$
Policy Improvement:
    Obtain $\pi_{j+1}$ such that
```

$$\pi_{j+1}(X_k) = -(H_{uu}^{\pi_j})^{-1}(H_{xu}^{\pi_j})^T X_k \tag{3.69}$$

where

$$u_k = -k^{\pi} x_k \tag{3.71}$$

derives the state x_k towards the origin.

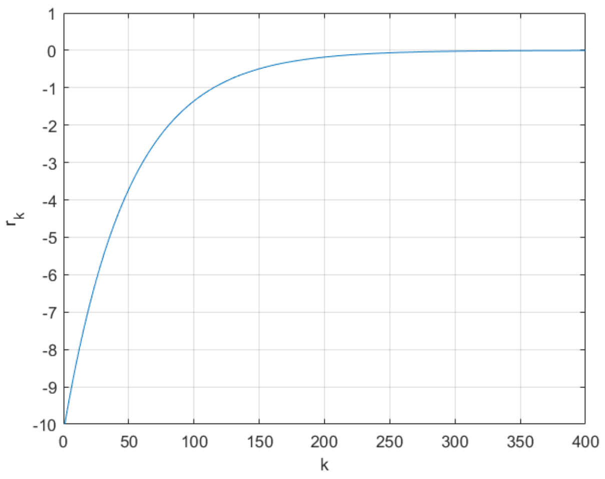


Figure 3.7: The desired trajectory of the quadcopter's poistion

3.2.5 Simulation

Consider the quadrotor described by equation (3.72) with a sampling time of $T_s = 0.1$ seconds, as was presented in section 3.1.5.

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \\ x_{3,k+1} \\ x_{4,k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0.05 & 0.003 \\ 0 & 0.99 & 0.99 & 0.05 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \\ x_{4,k} \end{bmatrix} + \begin{bmatrix} 0.001 \\ 0.011 \\ 0.022 \\ 0.435 \end{bmatrix} u_k$$

$$y_k = x_{1,k} \tag{3.72}$$

where the initial state is

$$\begin{bmatrix} x_{1,0} \\ x_{2,0} \\ x_{3,0} \\ x_{4,0} \end{bmatrix} = \begin{bmatrix} -10 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.73}$$

The reference trajectory r_k has the following linear dynamics

$$r_{k+1} = 0.98 r_k \tag{3.74}$$

Simulation number	K^{π_0}
Simulation 1	$\begin{pmatrix}0.62 & 0.99 & 5.57 & 1.66 & 0\end{pmatrix}$
Simulation 2	$\begin{pmatrix}0.23 & 0.41 & 2.97 & 1.15 & 0\end{pmatrix}$
Simulation 3	$\begin{pmatrix}1.27 & 1.92 & 9.35 & 2.36 & 0\end{pmatrix}$
Simulation 4	$\begin{pmatrix}0.11 & 0.22 & 1.92 & 0.92 & 0\end{pmatrix}$
Simulation 5	$\begin{pmatrix}1.85 & 1.70 & 7.49 & 1.85 & 0\end{pmatrix}$

Table 3.3: LQT quadcopter - The random initial stabilizing policies

with the initial condition $r_0 = -10$. Therefore, the objective is for the quadcopter's position to reach the origin following the trajectory depicted in Fig 3.7
Rewrite the policy $\pi_j(X_k)$ as

$$\pi_j(X_k) = -K^{\pi_j} X_k \tag{3.75}$$

We choose the following parameter values

$$\beta = 0,\ \gamma = 1,\ N = 200,\ R = 1,\ Q = 5$$

The optimal controller and value-function obtained from the conventionol method presented in section 2.2 are as follows

$$P^{\pi^*} = \begin{bmatrix} 36.8 & 13.1 & 32.9 & 4.3 & -34.2 \\ 13.1 & 7.4 & 22.5 & 3.5 & -11.7 \\ 32.9 & 22.5 & 81.3 & 15.4 & -28.5 \\ 4.3 & 3.5 & 15.4 & 4.0 & -3.7 \\ -34.2 & -11.7 & -28.5 & -3.7 & 31.9 \end{bmatrix} \tag{3.76}$$

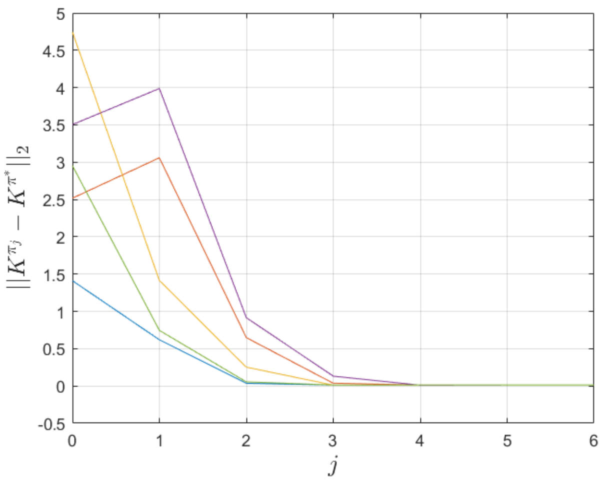


Figure 3.8: Convergence to LQT controller for the quadrotor

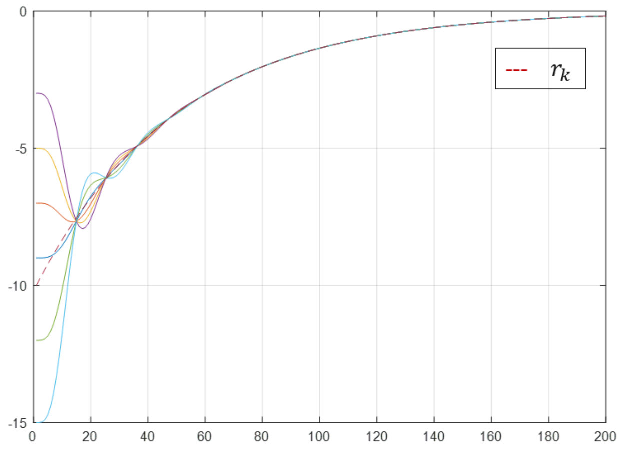


Figure 3.9: Comparison of y_k and r_k using the optimal policy

$$\pi^*(x_k) = -\begin{bmatrix} 1.2496 & 1.0679 & 4.9719 & 1.4235 & -1.0435 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \\ x_{4,k} \\ r_k \end{bmatrix} \tag{3.77}$$

The Fig 3.8 illustrates the convergence of K^{π_j} toward its optimal value K^{π^*}. We performed five simulations with randomly initialized stabilizing policies to visualize the convergence. The random initial policies are given in the Table 3.3. Lastly, the trajectory of the quadrotor's position y_k with different initial conditions using the optimal policy is compared with the desired trajectory r_k in Fig 3.9.

Chapter 4

Optimal Observer by Policy Iteration and Quadratic Neural Networks

This chapter introduces an innovative PI approach to obtain an optimal observer with a quadratic cost function. The observer is designed for systems with a given linearized model and a stabilizing Luenberger observer gain. We utilize a two-layer QNN for policy evaluation and derive a linear correction term using the input and output data. This correction term effectively rectifies inaccuracies introduced by the linearized model employed within the observer design.

A unique feature of the proposed methodology is that the QNN is trained through convex optimization. The main advantage is that the QNN's input-output mapping has an analytical expression as a quadratic form, which can then be used to obtain a linear correction term policy. This is in stark contrast to the available techniques in the literature that must train a second neural network to obtain policy improvement.

It is proven that the obtained linear correction term is optimal for linear systems, as both the value function and the QNN's input-output mapping are quadratic. The proposed method is applied to a simple pendulum, demonstrating an enhanced correction term policy compared to relying solely on the linearized model as section 2.2. This shows its promise for addressing nonlinear systems.

This chapter is organized as follows. Section 4.1 presents the problem statement. After section 4.2 gives the PI algorithm, section 4.3 presents how to perform the PI algorithm using the two-layer QNN. Section 4.4 shows the simulation results.

4.1 Problem Statement

Consider a system that is linear in the input with a linearized model

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{4.1}$$

around the origin, where $x_k \in \mathbb{R}^{n_x}$ is the unknown state vector, $u_k \in \mathbb{R}^{n_u}$ is the input vector, $y_k \in \mathbb{R}^{n_y}$ is the measured output vector, and A, B, C are system matrices. Assume (A, C) is observable. The goal is to design an optimal observer for the nonlinear system that is linear in the input using the linearized model and the available data. The dynamics of the observer utilizing the provided model are

$$\begin{aligned} \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + w_k^\pi \\ \hat{y}_k &= C\hat{x}_k \end{aligned} \tag{4.2}$$

where $\hat{x}_k$ and $\hat{y}_k$ are states and outputs estimates, respectively, and $w_k^\pi = \pi(.)$ is the correction term policy to be designed later.

The value-function following policy $\pi(.)$ is written as

$$V^\pi(\tilde{x}_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \left(\tilde{y}_i^T Q \tilde{y}_i + (w_i^\pi)^T R w_i^\pi \right) = \sum_{i=k}^{\infty} \gamma^{i-k} c(\tilde{y}_i, w_i^\pi) \tag{4.3}$$

where $Q = Q^T \geq 0$, $R = R^T > 0$ are chosen, $c(\tilde{y}_k, w_k^\pi)$ is the local cost at time step k,

Algorithm 6 General Policy Iteration to Design Optimal Observer

```
Select the initial policy $\pi_0$ that stabilizes the observer error dynamics. Then, for $j = 0, 1, 2, \ldots$
perform policy evaluation and policy improvement steps until convergence
Policy Evaluation:
    Solve for $V^{\pi_j}(.)$ such that
        $$V^{\pi_j}(\tilde{x}_k) = \tilde{y}_k^T Q \tilde{y}_k + (w_k^{\pi_j})^T R w_k^{\pi_j} + \gamma V^{\pi_j}(\tilde{x}_{k+1}) \quad (4.5)$$
Policy improvement:
    Obtain $w_k^{\pi_{j+1}}$ such that
        $$w_k^{\pi_{j+1}} = \arg\min_{\pi_j} V^{\pi_j}(\tilde{x}_k) \quad (4.6)$$
```

$0 < \gamma \leq 1$ is the discount factor, $\tilde{y}_k = y_k - \hat{y}_k$, and $\tilde{x}_k = x_k - \hat{x}_k$.

The objective is to design the correction term policy w_k^π that minimizes the value-function using PI algorithm.

The Bellman equation is

$$V^\pi(\tilde{x}_k) = \tilde{y}_k^T Q \tilde{y}_k + (w_k^\pi)^T R w_k^\pi + \gamma V^\pi(\tilde{x}_{k+1}) \tag{4.4}$$

In order to use the PI algorithm, we must add the assumption that an initial policy $\pi_0(.)$ which stabilizes the observer error dynamics is known. The general PI algorithm to obtain the optimal policy $\pi^*(.)$ is given in Algorithm 6.

When the system can be precisely described by the linear model, the error dynamics is

$$\begin{aligned} \tilde{x}_{k+1} &= A\tilde{x}_k - w_k^\pi \\ \tilde{y}_k &= C\tilde{x}_k \end{aligned} \tag{4.7}$$

and the goal is to solve

$$\begin{aligned} \arg\min_{\pi} \quad & \sum_{i=k}^{\infty} \gamma^{i-k} \left(\tilde{y}_i^T Q \tilde{y}_i + (w_i^{\pi})^T R w_i^{\pi} \right), \ \forall k \\ s.t. \quad & \tilde{x}_{i+1} = A\tilde{x}_i - w_i^{\pi} \\ & \tilde{y}_i = C\tilde{x}_i \end{aligned} \tag{4.8}$$

The optimization problem (4.8) can be rewritten as

$$\begin{aligned} \arg\min_{\pi} \quad & \sum_{i=k}^{\infty} \gamma^{i-k} \left(\tilde{x}_i^T \bar{Q} \tilde{x}_i + (w_i^{\pi})^T R w_i^{\pi} \right), \ \forall k \\ s.t. \quad & \tilde{x}_{i+1} = A\tilde{x}_i + (-I_{n_x}) w_i^{\pi} \end{aligned} \tag{4.9}$$

where $\bar{Q} = C^T Q C$ and I_{n_x} is the $n_x \times n_x$ identity matrix. Note that $(A, -I_{n_x})$ is controllable. Therefore, the optimization problem (4.9) can be considered as an LQR problem for the observer error dynamics. Thus, the optimal policy $\pi^*(.)$ is a linear function, and the optimal value-function $V^{\pi^*}(.)$ is quadratic. Therefore, we consider $\pi(\tilde{x}_k) = K^{\pi}\tilde{x}_k$ for some matrix K^{π}, and the corresponding value-function approximator $V^{\pi}(\tilde{x}_k) = \tilde{x}_k^T P^{\pi} \tilde{x}_k$ for a matrix $P^{\pi} > 0$.

Remark 9. *If $\tilde{x}_k$ is known, one can perform algorithm 6 as presented in [38]. However, $\tilde{x}_k$ is unknown and one cannot perform the policy evaluation step. To address this problem, we write the Bellman equation in terms of previous measured data instead of $\tilde{x}_k$ and $\tilde{x}_{k+1}$ and revise the PI algorithm accordingly.*

4.2 Modified Policy Iteration

In this section, we present the proposed PI algorithm by first refining the Bellman equation and then performing policy improvement.

4.2.1 Refining the Bellman Equation

Following the approach in reference [39], we reconstruct $\tilde{x}_k$ by previous measured data and replace $\tilde{x}_k$ in the Bellman equation. Consider the observer error dynamics (4.1) as the expanded state equation [39]

$$\begin{aligned} \tilde{x}_k &= A^{n_x}\tilde{x}_{k-n_x} + \mathcal{C}w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} &= \mathcal{O}\tilde{x}_{k-n_x} + Tw^{\pi}_{k-1,k-n_x} \end{aligned} \tag{4.10}$$

where

$$w^{\pi}_{k-1,k-n_x} = \begin{bmatrix} w^{\pi}_{k-1} \\ w^{\pi}_{k-2} \\ \vdots \\ w^{\pi}_{k-n_x} \end{bmatrix}, \quad \tilde{y}_{k-1,k-n_x} = \begin{bmatrix} \tilde{y}_{k-1} \\ \tilde{y}_{k-2} \\ \vdots \\ \tilde{y}_{k-n_x} \end{bmatrix}, \; T = \begin{bmatrix} 0 & -C & -CA & \dots & -CA^{n_x-2} \\ 0 & 0 & -C & \dots & -CA^{n_x-3} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & & 0 & -C \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$\mathcal{C} = \begin{bmatrix} -I_{n_x} & -A & -A^2 & \dots & -A^{n_x-1} \end{bmatrix}$, and the observability matrix $\mathcal{O}$ is

$$\mathcal{O} = \begin{bmatrix} CA^{n_x-1} \\ CA^{n_x-2} \\ \vdots \\ CA \\ C \end{bmatrix} \in \mathbb{R}^{(n_x n_y)\times n_x}. \tag{4.11}$$

Note that (A, C) is observable, and the observability matrix $\mathcal{O}$ has full column rank n_x. Therefore, $\mathcal{O}^{+} = (\mathcal{O}^T\mathcal{O})^{-1}\mathcal{O}^T$ is its left inverse. Lemma 5 based on reference [39] allows us to replace $\tilde{x}_k$ with measured data.

Lemma 5. *One can reconstruct $\tilde{x}_k$ from measured data as*

$$\tilde{x}_k = \begin{bmatrix} M_w & M_{\tilde{y}} \end{bmatrix} \begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} \tag{4.12}$$

where $M_w = \mathcal{C} - A^{n_x}\mathcal{O}^{+}T$ and $M_{\tilde{y}} = A^{n_x}\mathcal{O}^{+}$.

Proof. Equation (4.10) yields

$$\begin{aligned} \tilde{x}_k - \mathcal{C}w^{\pi}_{k-1,k-n_x} &= A^{n_x}\tilde{x}_{k-n_x} \\ &= A^{n_x} I_{n_x}\tilde{x}_{k-n_x} \end{aligned} \tag{4.13}$$

Note that $I_{n_x} = \mathcal{O}^{+}\mathcal{O}$. Therefore,

$$\tilde{x}_k - \mathcal{C}w^{\pi}_{k-1,k-n_x} = A^{n_x}\mathcal{O}^{+}(\mathcal{O}\tilde{x}_{k-n_x}) \tag{4.14}$$

From (4.10), we can replace $\mathcal{O}\tilde{x}_{k-n_x}$ with $\tilde{y}_{k-1,k-n_x} - Tw^{\pi}_{k-1,k-n_x}$ in equation (4.14) and get

$$\tilde{x}_k - \mathcal{C}w^{\pi}_{k-1,k-n_x} = A^{n_x}\mathcal{O}^{+}\left(\tilde{y}_{k-1,k-n_x} - Tw^{\pi}_{k-1,k-n_x}\right) \tag{4.15}$$

which can be recast as

$$\begin{aligned} \tilde{x}_k &= \begin{bmatrix} \mathcal{C} - A^{n_x}\mathcal{O}^{+}T & A^{n_x}\mathcal{O}^{+} \end{bmatrix} \begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} \\ &= \begin{bmatrix} M_w & M_{\tilde{y}} \end{bmatrix} \begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} \end{aligned} \tag{4.16}$$

□

4.2.2 Policy Evaluation Step

Using Lemma 5 and $V^\pi(\tilde{x}_k) = \tilde{x}_k^T P \tilde{x}_k$, the Bellman equation can be rewritten based on previous measured data as

$$\begin{bmatrix} w^\pi_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T H^\pi \begin{bmatrix} w^\pi_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \tilde{y}_k^T Q \tilde{y}_k + (w_k^\pi)^T R w_k^\pi + \gamma \begin{bmatrix} w^\pi_{k,k-n_x+1} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T H^\pi \begin{bmatrix} w^\pi_{k,k-n_x+1} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix} \tag{4.17}$$

where $H^\pi = \begin{bmatrix} M_w & M_{\tilde{y}} \end{bmatrix}^T P^\pi \begin{bmatrix} M_w & M_{\tilde{y}} \end{bmatrix}$ is a symmetric matrix. One contribution of this chapter is to use the available measurements to train a two-layer QNN with a single output to find the matrix H^π of equation (4.17) and evaluate the policy $\pi(.)$. An overview of two-layer QNNs is provided in section 2.3. The section 4.3 gives the training of the neural network within the algorithm.

Remark 10. *Note that equation* (4.17) *is a scalar equation.*

$$\begin{bmatrix} w^\pi_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} \in \mathbb{R}^{n_x(n_x+n_y)} \tag{4.18}$$

and H^π is symmetric. Therefore, the matrix H^π has $M = \frac{n_x(n_x+n_y)(n_x(n_x+n_y)+1)}{2}$ unknown independent elements and $N \geq M$ data samples are needed to obtain H^π from (4.17).

Remark 11. *To achieve PE in practice, we add a probing noise term n_k such that $w_k^\pi = K^\pi \tilde{x}_k + n_k$. It is shown in reference [39] that the solution computed by PI differs from the actual value corresponding to the Bellman equation when the probing noise term n_k is added. It is discussed that adding the discount factor $0 < \gamma < 1$ to the Bellman equation reduces this harmful effect of n_k.*

4.2.3 Policy Improvement Step

We now address how to improve the policy $\pi(.)$ after evaluating the policy and obtaining the matrix H^π with a two-layer QNN.

The policy improvement step can be written as

$$\pi'(\tilde{x}_k) = \arg\min_{w_k^\pi} \left(\tilde{y}_k^T Q \tilde{y}_k + (w_k^\pi)^T R w_k^\pi + \gamma \begin{bmatrix} w_{k,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T H^\pi \begin{bmatrix} w_{k,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix} \right) \tag{4.19}$$

where $\pi'(.)$ is the improved policy over the policy $\pi(.)$.

Partition $\begin{bmatrix} w_{k,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T H^\pi \begin{bmatrix} w_{k,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}$ as

$$\begin{bmatrix} w_k^\pi \\ w_{k-1,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T \begin{bmatrix} H_{11}^\pi & H_w^\pi & H_{\tilde{y}}^\pi \\ (H_w^\pi)^T & H_{22}^\pi & H_{23}^\pi \\ (H_{\tilde{y}}^\pi)^T & (H_{23}^\pi)^T & H_{33}^\pi \end{bmatrix} \begin{bmatrix} w_k^\pi \\ w_{k-1,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix} \tag{4.20}$$

One can solve (4.19) and get the improved policy as

$$w_k^{\pi'} = -\gamma(R + \gamma H_{11}^\pi)^{-1}(H_w^\pi w_{k-1,k-n_x+1} + H_{\tilde{y}}^\pi \tilde{y}_{k,k-n_x+1}) \tag{4.21}$$

Therefore, we can use the PI Algorithm 7 and obtain the optimal policy $\pi^*(.)$ with data.

Remark 12. *For linear systems, the optimal H^π can be obtained using the following closed-form solution*

$$H^\pi = \begin{bmatrix} M_w & M_{\tilde{y}} \end{bmatrix}^T P^\pi \begin{bmatrix} M_w & M_{\tilde{y}} \end{bmatrix} \tag{4.22}$$

where P^π can be calculated from solving the Riccati equation (4.23) *as discussed in section 2.2*

Algorithm 7 Modified Policy Iteration with Quadratic Value Function

```
Select the initial policy $w_k^{\pi_0}$ that stabilizes the observer error dynamics. Then, for $j = 0, 1, 2, \dots$ perform policy evaluation and policy improvement steps until convergence
Policy Evaluation:
    Find $H^{\pi_j}$ such that
    $\begin{bmatrix} w^{\pi_j}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T H^{\pi_j} \begin{bmatrix} w^{\pi_j}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \tilde{y}_k^T Q \tilde{y}_k + (w_k^{\pi_j})^T R w_k^{\pi_j} + \gamma \begin{bmatrix} w^{\pi_j}_{k,k-n_x+1} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T H^{\pi_j} \begin{bmatrix} w^{\pi_j}_{k,k-n_x+1} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}$
Policy improvement:
    Obtain $\pi_{j+1}$ such that
    $w_k^{\pi_{j+1}} = -\gamma(R + \gamma H_{11}^{\pi_j})^{-1}(H_w^{\pi_j} w_{k-1,k-n_x+1} + H_{\tilde{y}}^{\pi_j} \tilde{y}_{k,k-n_x+1})$
```

with the system model.

$$P^\pi = C^T Q C + \gamma A^T P^\pi A - \gamma^2 A^T P^\pi (R + \gamma P^\pi)^{-1} P^\pi A \tag{4.23}$$

Then the optimal correction term can be designed as

$$w_k^\pi = -\gamma(R + \gamma H_{11}^\pi)^{-1}(H_w^\pi w_{k-1,k-n_x+1} + H_{\tilde{y}}^\pi \tilde{y}_{k,k-n_x+1}) \tag{4.24}$$

4.3 QNNs as the Policy Evaluator

This section shows how a two-layer QNN can be trained to do policy evaluation and how to obtain H^π. Then, the proposed algorithm to find $\pi^*(.)$ without the system model is presented. First, we introduce the following Lemma.

Lemma 6. *Let $\hat{H}_i^\pi$ denote the i-th approximation of H^π for the policy $\pi(.)$ that stabilizes the observer error dynamics. Starting with $i = 1$ and any symmetric $\hat{H}_0^\pi$, iterating through* *equation* (4.25) *will result in* $\hat{H}_i^\pi$ *converging to* H^π *provided* $0 < \gamma \leq 1$.

$$\begin{bmatrix} w_{k-1,k-n_x}^\pi \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T \hat{H}_i^\pi \begin{bmatrix} w_{k-1,k-n_x}^\pi \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \tilde{y}_k^T Q \tilde{y}_k + (w_k^\pi)^T R w_k^\pi + \gamma \begin{bmatrix} w_{k,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T \hat{H}_{i-1}^\pi \begin{bmatrix} w_{k,k-n_x+1}^\pi \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix} \tag{4.25}$$

Proof. Applying the equation (4.25) recursively yields

$$\begin{bmatrix} w_{k-1,k-n_x}^\pi \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T \hat{H}_i^\pi \begin{bmatrix} w_{k-1,k-n_x}^\pi \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \sum_{j=0}^{i-1} \gamma^j c(\tilde{y}_{k+j}, w_{k+j}^\pi) + \gamma^i \begin{bmatrix} w_{k+i-1,k+i-n_x}^\pi \\ \tilde{y}_{k+i-1,k+i-n_x} \end{bmatrix}^T \hat{H}_0^\pi \begin{bmatrix} w_{k+i-1,k+i-n_x}^\pi \\ \tilde{y}_{k+i-1,k+i-n_x} \end{bmatrix}$$

Let $i \to \infty$. Then,

$$\begin{bmatrix} w_{k-1,k-n_x}^\pi \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T \hat{H}_\infty^\pi \begin{bmatrix} w_{k-1,k-n_x}^\pi \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \sum_{j=0}^{\infty} \gamma^j c(\tilde{y}_{k+j}, w_{k+j}^\pi) + \lim_{i\to\infty} \gamma^i \begin{bmatrix} w_{k+i-1,k+i-n_x}^\pi \\ \tilde{y}_{k+i-1,k+i-n_x} \end{bmatrix}^T \hat{H}_0^\pi \begin{bmatrix} w_{k+i-1,k+i-n_x}^\pi \\ \tilde{y}_{k+i-1,k+i-n_x} \end{bmatrix} \tag{4.26}$$

Since $\lim_{i\to\infty} \gamma^i = 0$, for any $\hat{H}_0^\pi$ we also get

$$\lim_{i\to\infty} \gamma^i \begin{bmatrix} w_{k+i-1,k+i-n_x}^\pi \\ \tilde{y}_{k+i-1,k+i-n_x} \end{bmatrix}^T \hat{H}_0^\pi \begin{bmatrix} w_{k+i-1,k+i-n_x}^\pi \\ \tilde{y}_{k+i-1,k+i-n_x} \end{bmatrix} = 0 \tag{4.27}$$

Replacing (4.27) in (4.26) yields

$$\begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T \hat{H}^{\pi}_{\infty} \begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \sum_{j=0}^{\infty} \gamma^j c(\tilde{y}_{k+j}, w^{\pi}_{k+j}) \tag{4.28}$$

According to the Bellman equation (4.17), we also have

$$\begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix}^T H^{\pi} \begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix} = \sum_{j=0}^{\infty} \gamma^j c(\tilde{y}_{k+j}, w^{\pi}_{k+j}) \tag{4.29}$$

As a result of (4.28)(4.29), $\hat{H}^{\pi}_i$ converges to H^{π}. □

Therefore, we can calculate H^{π} if we can obtain $\hat{H}^{\pi}_i$ in equation (4.25) from the previous known $\hat{H}^{\pi}_{i-1}$ in the i-th iteration. We use a two-layer QNN Fig 4.2 to obtain $\hat{H}^{\pi}_i$. More specifically, defining the input $\mathcal{X}_k$ and output label $\mathcal{Y}_k$ for the QNN as

$$\mathcal{X}_k = \begin{bmatrix} w^{\pi}_{k-1,k-n_x} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix},$$

$$\mathcal{Y}_k = \tilde{y}_k^T Q \tilde{y}_k + (w^{\pi}_k)^T R w^{\pi}_k + \gamma \begin{bmatrix} w^{\pi}_{k,k-n_x+1} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T \hat{H}^{\pi}_{i-1} \begin{bmatrix} w^{\pi}_{k,k-n_x+1} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix} \tag{4.30}$$

and given $N >> M$ data points, we train the QNN and obtain the quadratic input-output mapping from (2.48) and set $\hat{H}^{\pi}_i = H$. The detailed methodology is described in Algorithm 8 with a convergence stopping criterium of $||\hat{H}^{\pi_j}_i - \hat{H}^{\pi_j}_{i-1}|| < \epsilon$ for a small ϵ.

Figure 4.1 presents an overview of the proposed algorithm.

Algorithm 8 The PI algorithm with a QNN as the VFA

```
Choose $\epsilon$, $N$, $\gamma$.
Select the initial policy $w_k^{\pi_0}$ that stabilizes the observer error dynamics. Then, for $j = 0, 1, 2, \dots$ do
Policy Evaluation:
    $i \leftarrow 0$
    Choose a random $\hat{H}_0^{\pi_j}$
    repeat
        $i \leftarrow i + 1$
        Train the QNN by $N$ data samples with $\mathcal{X}_k$ as the input and $\mathcal{Y}_k$ as the output label
```

$$\mathcal{X}_k = \begin{bmatrix} w_{k-1,k-n_x}^{\pi_j} \\ \tilde{y}_{k-1,k-n_x} \end{bmatrix},$$

$$\mathcal{Y}_k = \tilde{y}_k^T Q \tilde{y}_k + (w_k^{\pi_j})^T R w_k^{\pi_j} + \gamma \begin{bmatrix} w_{k,k-n_x+1}^{\pi_j} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix}^T \hat{H}_{i-1}^{\pi_j} \begin{bmatrix} w_{k,k-n_x+1}^{\pi_j} \\ \tilde{y}_{k,k-n_x+1} \end{bmatrix} \tag{4.31}$$

```
        Obtain the input-output mapping as
```

$$\mathcal{X}_k^T H \mathcal{X}_k = \hat{\mathcal{Y}}_k \tag{4.32}$$

```
        $\hat{H}_i^{\pi_j} \leftarrow H$
    Until $||\hat{H}_i^{\pi_j} - \hat{H}_{i-1}^{\pi_j}|| < \epsilon$
    $H^{\pi_j} \leftarrow \hat{H}_i^{\pi_j}$
Policy Improvement:
    Obtain $w_k^{\pi_{j+1}}$ such that
```

$$w_k^{\pi_{j+1}} = -\gamma(R + \gamma H_{11}^{\pi_j})^{-1}(H_w^{\pi_j} w_{k-1,k-n_x+1} + H_{\tilde{y}}^{\pi_j} \tilde{y}_{k,k-n_x+1}) \tag{4.33}$$

4.4 Simulation Results

A pendulum is first considered as a simple linear system to show how the proposed method converges to the optimal correction term policy for the LQE (refer to Remark 12). Then, a nonlinear model of the pendulum is considered to highlight the benefits of the proposed method compared to relying solely on the provided linearized pendulum model.

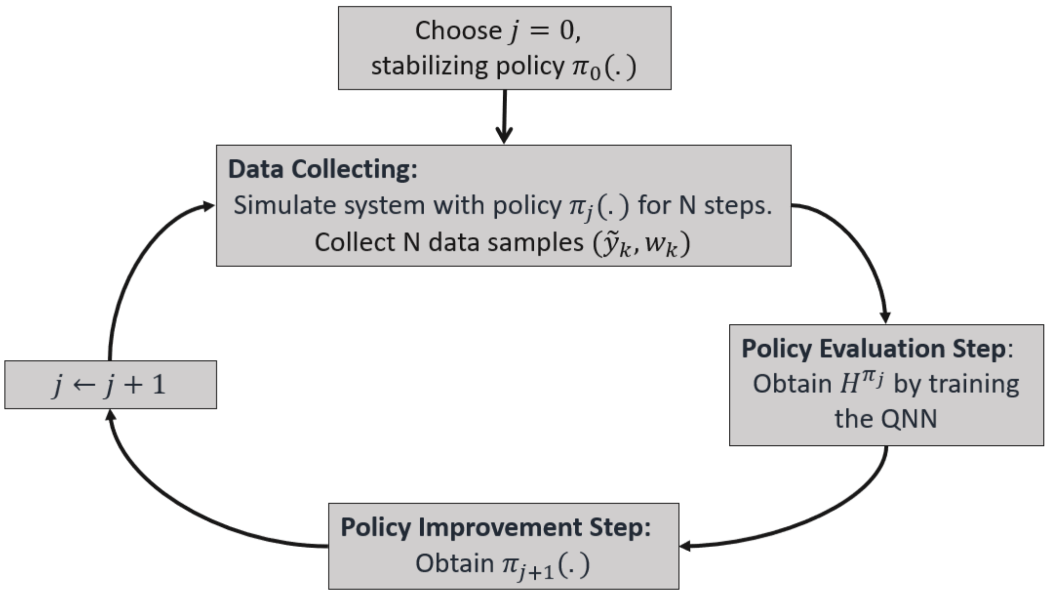

Choose $j = 0$, stabilizing policy $\pi_0(.)$
Data Collecting: Simulate system with policy $\pi_j(.)$ for N steps. Collect N data samples $(\tilde{y}_k, w_k)$
Policy Evaluation Step: Obtain H^{π_j} by training the QNN
Policy Improvement Step: Obtain $\pi_{j+1}(.)$
$j \leftarrow j+1$

Figure 4.1: Overview of LQE proposed design

4.4.1 Pendulum Linear Model Results

Consider a pendulum modeled as (4.34) with sampling time $T_s = 0.1$s.

$$\begin{aligned} \begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} &= \begin{bmatrix} 0.95 & 0.10 \\ -0.98 & 0.94 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \begin{bmatrix} 0.005 \\ 0.098 \end{bmatrix} u_k \\ y_k &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \end{aligned} \tag{4.34}$$

As a result, $M_{\tilde{y}}$ and M_w are obtained as

$$M_w = \begin{bmatrix} -1 & 0 & -0.95 & -0.92 \\ 0 & -1 & 0.98 & 0.95 \end{bmatrix}, \quad M_{\tilde{y}} = \begin{bmatrix} -0.82 & 0.96 \\ 1.89 & -0.99 \end{bmatrix} \tag{4.35}$$

We choose $\gamma = 1,\ N = 200,\ \beta = 0,\ Q = 10$ and $R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

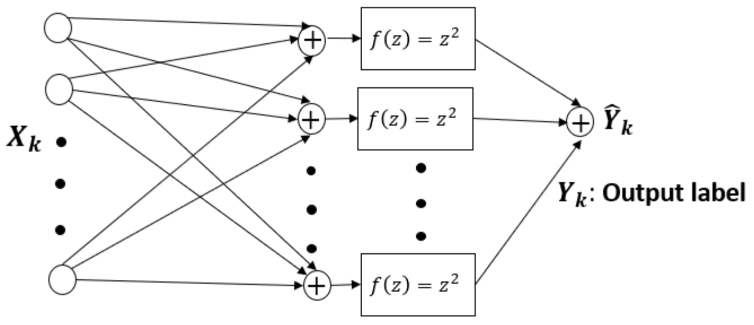


Figure 4.2: QNN as the policy evalautor to obtain $\hat{H}_i^{\pi}$ in i-th iteration

From the closed-form solution for LQE we get

$$P^{\pi^*} = \begin{bmatrix} 1.45 & -0.80 \\ -0.80 & 10.80 \end{bmatrix} \tag{4.36}$$

$$H^{\pi^*} = \begin{bmatrix} 1.4 & -0.8 & 2.2 & 2.1 & 2.7 & -2.2 \\ -0.8 & 10.8 & -11.3 & -11.0 & -21.1 & 11.5 \\ 2.2 & -11.3 & 13.2 & 12.8 & 23.2 & -13.3 \\ 2.1 & -11.0 & 12.8 & 12.4 & 22.6 & -12.9 \\ 2.7 & -21.1 & 23.2 & 22.6 & 42.2 & -23.5 \\ -2.2 & 11.5 & -13.3 & -12.9 & -23.5 & 13.5 \end{bmatrix} \tag{4.37}$$

The objective is to show that with any initial stabilizing policy $\pi_0(.)$, the proposed algorithm 8 converges to the matrix H^{π^*} and therefore the optimal policy $\pi^*(.)$ is achieved. We ran ten simulations with random initial stabilizing policies. It is shown in Fig 4.3 that $\pi_j(.)$ converges to the optimal policy $\pi^*(.)$ in all ten runs of the algorithm since H^{π_j} converges to H^{π^*}.

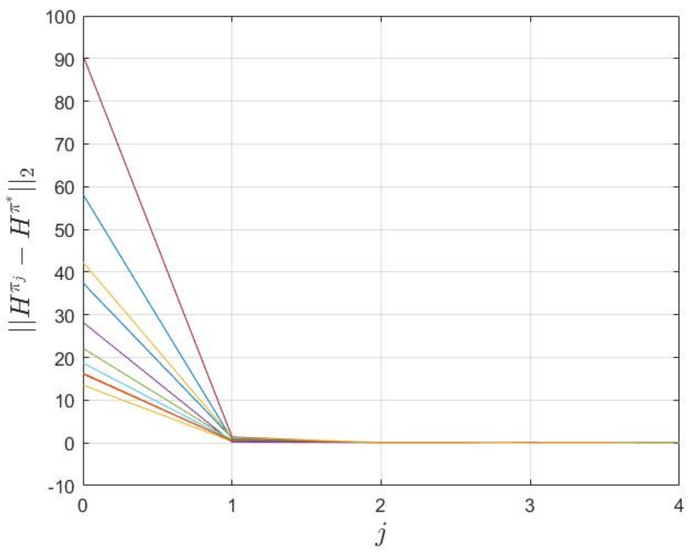


Figure 4.3: LQE pendulum - Convergence of H^{π_j}

4.4.2 Pendulum Nonlinear Model Results

The proposed method is also applied to the nonlinear model of the simple pendulum. The results will show that the data-driven method gives an improved observer compared to obtaining the correction term closed-form solution using the provided linearized model.

Consider the pendulum with dynamic model

$$\ddot{\theta}\,(t) + 0.1\dot{\theta}\,(t) + 10\,sin\,(\theta\,(t)) = u(t) \tag{4.38}$$

where $\theta(t)$ represents the pitch angle and the measured output is $\dot{\theta}\,(t)$.

We choose $N = 200,\ \beta = 0,\ Q = 10$ and $R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The A, B, C matrices in (4.34) which are derived from linearization around

$$\begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.39}$$

are used for the linearized model used in the observer dynamics (4.2).

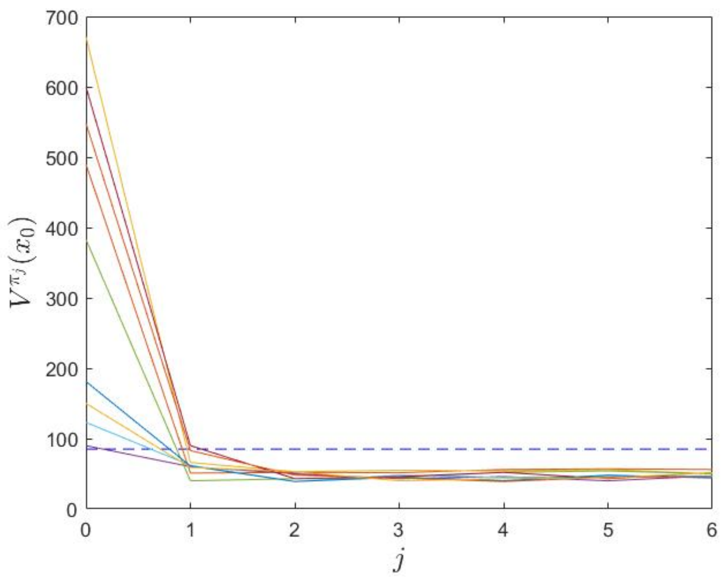


Figure 4.4: LQE pendulum - Cost-to-go from the initial state with $\gamma = 1$

4.4.2.1 Simulation Results with $\gamma = 1$

First, we choose $\gamma = 1$. One can first obtain the correction term policy w_k^π solely using the linearized model with the closed form solution given in (4.37). To compare this result with obtaining the correction term using the proposed data-driven method, we ran ten simulation with different initial stabilizing policies and the initial state

$$x_0 = \begin{bmatrix} \theta(0) \\ \dot{\theta}(0) \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \tag{4.40}$$

. Figure 4.4 shows that the proposed method consistently yields lower initial cost-to-go values than the closed-form solution, wich is shown with a dash line. In this instance, the selection of Q and R matrices prioritizes minimizing the output error. As depicted in Fig 4.5, the proposed approach outperforms the correction term derived from the closed-form formula and reduces the output error over time.

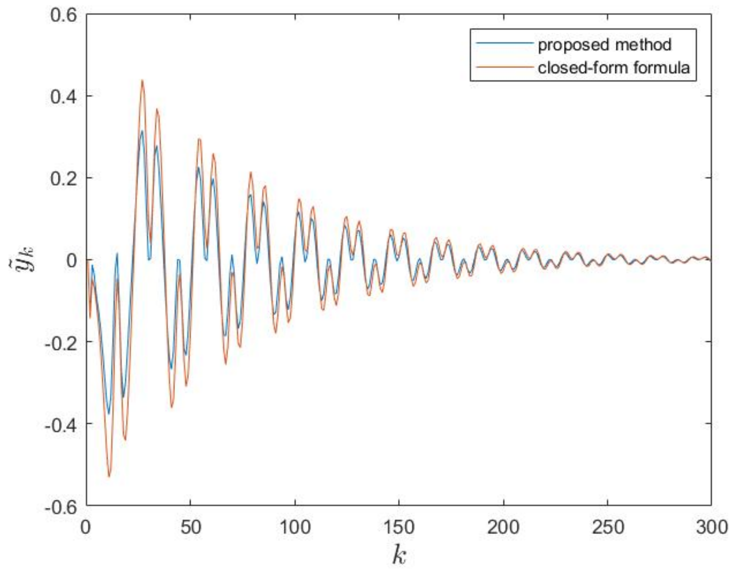


Figure 4.5: LQE pendulum - Output error comparison with $\gamma = 1$

4.4.2.2 Simulation Results with $\gamma = 0.4$

We select $\gamma = 0.4$ to enhance performance by mitigating the impact of noise n_k. Figure 4.6 demonstrates that the proposed method consistently achieves lower initial cost-to-go values compared to the closed-form solution, represented by the dashed line. Additionally, Fig 4.7 showcases how our approach reduces the output error over time. As anticipated, the selection of $\gamma = 0.4$ notably enhances the observer's performance..

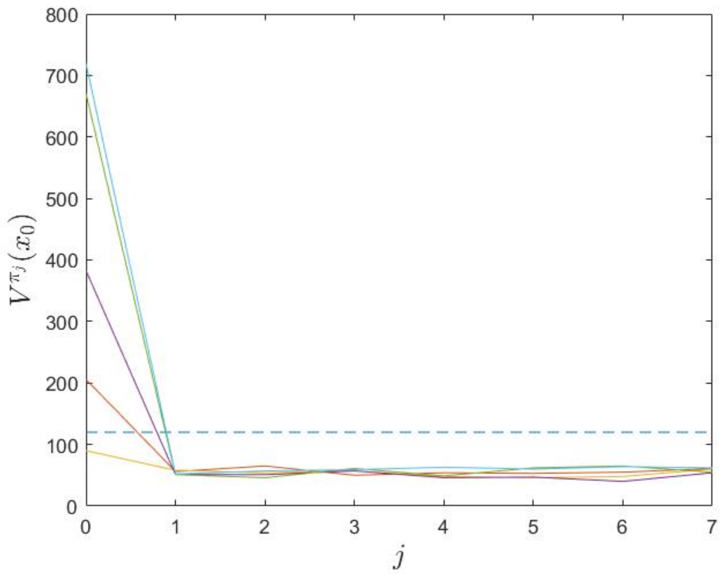


Figure 4.6: LQE pendulum - Cost-to-go from the initial state with $\gamma = 0.4$

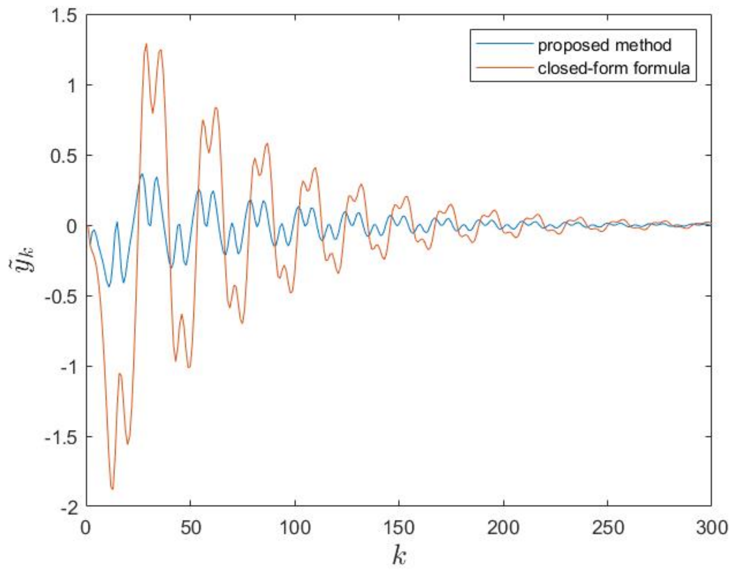


Figure 4.7: LQE pendulum - Output error comparison with $\gamma = 0.4$

Chapter 5

Conclusions and Future Work

In this thesis, the application of RL in optimal control has been explored to address traditional control methods limitations when dealing with nonlinear, uncertain, large-scale systems, or systems with unknown dynamics. The integration of RL with optimal control tackles these challenges by offering adaptability, the ability to handle complex dynamics, and the capacity to overcome model inaccuracies. Many RL algorithms have employed ANNs to approximate complex functions, aiming for broad applicability. However, this approach has drawbacks, including difficulties associated with non-convex optimization, complex architecture selection, and the absence of straightforward analytical input-output mapping.

The primary goal was to utilize QNNs within the RL algorithm to design discrete-time LQR, LQT, and LQE, particularly without the need for system model knowledge. This idea stemmed from (i) observed drawbacks in using ANNs within RL algorithms, (2) both the input-output mapping of the two-layer QNN and the value-function of LQ problems are quadratic, (iii) the effectiveness of QNNs in various fields.

The convergence of the proposed methods to optimal solutions was established through rigorous theoretical proofs and practical case studies. MATLAB simulations confirmed the theoretical claims, validating the proposed methodology with various control problems. The simulations also suggest that the proposed algorithm for optimal observer design yields favorable results when applied to nonlinear systems provided a linearized model is given.

Moving forward, future research should aim to extend this approach to nonlinear systems, leveraging deep QNNs trained via convex optimization. This extension will enhance the methodology's applicability and effectiveness in real-world applications.

Bibliography

[1] A. E. Bryson and Y.-C. Ho, *Applied optimal control: optimization, estimation, and control.* Routledge, 2018.

[2] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control.* John Wiley & Sons, 2012.

[3] M. Gan, J. Zhao, and C. Zhang, "Extended adaptive optimal control of linear systems with unknown dynamics using adaptive dynamic programming," *Asian Journal of Control*, vol. 23, no. 2, pp. 1097–1106, 2021.

[4] Y. Jiang and Z.-P. Jiang, "Global adaptive dynamic programming for continuous-time nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 60, no. 11, pp. 2917–2929, 2015.

[5] D. Bertsekas, *Reinforcement learning and optimal control.* Athena Scientific, 2019.

[6] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

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

[8] C. Szepesvári, *Algorithms for reinforcement learning.* Springer Nature, 2022.

[9] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2017.

[10] N. Matni, A. Proutiere, A. Rantzer, and S. Tu, "From self-tuning regulators to reinforcement learning and back again," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 3724–3740.

[11] F. A. Yaghmaie, F. Gustafsson, and L. Ljung, "Linear quadratic control using model-free reinforcement learning," *IEEE Transactions on Automatic Control*, 2022.

[12] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning*. PMLR, 2016, pp. 1329–1338.

[13] S. G. Khan, G. Herrmann, F. L. Lewis, T. Pipe, and C. Melhuish, "Reinforcement learning and optimal adaptive control: An overview and implementation examples," *Annual reviews in control*, vol. 36, no. 1, pp. 42–59, 2012.

[14] D. Zhao, Z. Xia, and D. Wang, "Model-free optimal control for affine nonlinear systems with convergence analysis," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 4, pp. 1461–1468, 2014.

[15] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[16] L. Rodrigues and S. Givigi, "Analysis and design of quadratic neural networks for regression, classification, and lyapunov control of dynamical systems," *arXiv preprint arXiv:2207.13120*, 2022.

[17] B. Bartan and M. Pilanci, "Neural spectrahedra and semidefinite lifts: Global convex optimization of polynomial activation neural networks in fully polynomial-time," *arXiv preprint arXiv:2101.02429*, 2021.

[18] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[19] B. D. Anderson and J. B. Moore, *Optimal control: linear quadratic methods.* Courier Corporation, 2007.

[20] V. Sima, *Algorithms for linear-quadratic optimization.* CRC Press, 2021.

[21] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[22] J. C. Doyle, "Guaranteed margins for lqg regulators," *IEEE Transactions on automatic Control*, vol. 23, no. 4, pp. 756–757, 1978.

[23] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[24] D. Kleinman, "On an iterative technique for riccati equation computations," *IEEE Transactions on Automatic Control*, vol. 13, no. 1, pp. 114–115, 1968.

[25] D. Tailor and D. Izzo, "Learning the optimal state-feedback via supervised imitation learning," *Astrodynamics*, vol. 3, pp. 361–374, 2019.

[26] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, 2016.

[27] J. Boedecker, J. T. Springenberg, J. Wülfing, and M. Riedmiller, "Approximate real-time optimal control based on sparse gaussian process models," in *2014 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL).* IEEE, 2014, pp. 1–8.

[28] D. Zeidler, S. Frey, K.-L. Kompa, and M. Motzkus, "Evolutionary algorithms and their application to optimal control studies," *Physical Review A*, vol. 64, no. 2, p. 023420, 2001.

[29] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[30] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[31] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation," *Course Notes for MIT*, vol. 6, 2016.

[32] F. L. Lewis and D. Liu, *Reinforcement learning and approximate dynamic programming for feedback control.* John Wiley & Sons, 2013.

[33] H. Liu, B. Kiumarsi, Y. Kartal, A. Taha Koru, H. Modares, and F. L. Lewis, "Reinforcement learning applications in unmanned vehicle control: A comprehensive overview," *Unmanned Systems*, vol. 11, no. 01, pp. 17–26, 2023.

[34] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep rl for model-based control," *arXiv preprint arXiv:1802.09081*, 2018.

[35] D. Liu, S. Xue, B. Zhao, B. Luo, and Q. Wei, "Adaptive dynamic programming for control: A survey and recent advances," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 142–160, 2020.

[36] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[37] B. Pang and Z.-P. Jiang, "Robust reinforcement learning: A case study in linear quadratic regulation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 9303–9311.

[38] S. J. Bradtke, B. E. Ydstie, and A. G. Barto, “Adaptive linear quadratic control using policy iteration,” in *Proceedings of 1994 American Control Conference-ACC’94*, vol. 3. IEEE, 1994, pp. 3475–3479.

[39] F. L. Lewis and K. G. Vamvoudakis, “Reinforcement learning for partially observable dynamic processes: Adaptive dynamic programming using measured output data,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 1, pp. 14–25, 2010.

[40] S. A. A. Rizvi and Z. Lin, “Experience replay–based output feedback q-learning scheme for optimal output tracking control of discrete-time linear systems,” *International Journal of Adaptive Control and Signal Processing*, vol. 33, no. 12, pp. 1825–1842, 2019.

[41] B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, and M.-B. Naghibi-Sistani, “Reinforcement q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics,” *Automatica*, vol. 50, no. 4, pp. 1167–1175, 2014.

[42] B. Kiumarsi, F. L. Lewis, M.-B. Naghibi-Sistani, and A. Karimpour, “Optimal tracking control of unknown discrete-time linear systems using input-output measured data,” *IEEE transactions on cybernetics*, vol. 45, no. 12, pp. 2770–2779, 2015.

[43] J. Na, G. Herrmann, and K. G. Vamvoudakis, “Adaptive optimal observer design via approximate dynamic programming,” in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 3288–3293.

[44] J. Li, Z. Xiao, P. Li, and Z. Ding, “Networked controller and observer design of discrete-time systems with inaccurate model parameters,” *ISA transactions*, vol. 98, pp. 75–86, 2020.

[45] L. Rodrigues and S. Givigi, “System identification and control using quadratic neural networks,” *IEEE Control Systems Letters*, vol. 7, pp. 2209–2214, 2023.

[46] H. Kwakernaak and R. Sivan, *Linear optimal control systems.* Wiley-interscience, 1969, vol. 1072.

[47] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[48] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[49] R. Kalman, P. L. Falb, and M. A. Arbib, "Controllability and observability for linear systems," *IEEE Transactions on Automatic Control*, vol. 9, no. 3, pp. 291–292, 1964.

[50] G. H. Golub and C. F. Van Loan, *Matrix Computations.* Johns Hopkins University Press, 1996.

[51] W. L. Root and H. W. Lee, "A riccati equation arising in stochastic control," *SIAM Journal on Control*, vol. 8, no. 4, pp. 401–414, 1970.

[52] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods.* Dover Publications, 1990.

[53] K. Ogata, *Modern control engineering fifth edition*, 2010.