Model-Free Optimized Tracking Control Heuristic

by

Ning Wang

Thesis submitted in partial fulfillment of the requirements for the

M.A.Sc. Degree

in

Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

Tracking control algorithms often target the convergence of a tracking error. However, this can be at the expense of other important system characteristics, such as the control effort used to annihilate the tracking error, transient response, or steady-state characteristics, for example. Furthermore, most tracking control methods assume prior knowledge of the system dynamics, which is not always a realistic assumption, especially in the case of highly complex systems.

In this thesis, a model-free optimized tracking control architectural heuristic is proposed. The suggested feedback system is composed of two control loops. The first is the tracking loop. It focuses on the convergence of the tracking error. It is implemented using two different model-free control algorithms for comparison purpose: Reinforcement Learning (RL) and the Nonlinear Threshold Accepting (NLTA) technique. The RL scheme reformulates the tracking error combinations into a form of Markov-Decision-Process (MDP) and applies Q-Learning to build the best tracking control policy for the dynamic system under consideration. On the other hand, the NLTA algorithm is applied to tune the gains of a PID controller. The second control loop is in the form of a nonlinear state feedback loop. It is implemented using a feedforward artificial neural network (ANN) to optimize a system-wide cost function which can be flexible enough to encompass a set of desired design requirements pertaining to the targeted system behavior. This may include, for instance, the target overshoot, settling time, rise time, etc. The proposed architectural heuristic provides a model-free framework to tackle such control problems, in the sense that the plant's dynamic model is not required to be known in advance. Yet, at least a subset of the stability region of the optimized gains has to be known in advance so that it can provide a search space for the optimization algorithms. Simulation results on two dynamic systems demonstrate the superiority of the proposed control scheme.

Acknowledgements

I owe my deepest appreciation to my thesis supervisor Dr. Wail Gueaieb and my research mentor Dr. Mohammed Abouheaf for their patient guidance, suggestions and support throughout my graduate studies. This thesis would not have been possible without their advice and encouragement.

Meanwhile, I want to express my gratefulness to my colleagues and friends for their friendship, care and support.

Finally, many thanks and gratitude to my family for their endless support, love, and encouragement.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

ABC	Artificial Bee Colony
AHC	Autonomous Helicopter Control
ANN	Artificial Neural Network
ARE	Algebraic Riccati Equation
CAFVI	Continuous Action Fitted Value Iteration
CoG	Center of Gravity
DDP	Differencial Dynamic Programming
DEA	Differential Evolution Algorithm
DP	Dynamic Programming
IMC	Iternal Model Control
IRL	Itegral Reinforcement Learning
ISE	Itegrated Squared Error
ISE-AB	Itegrated Squared Errors and Absolute Errors
ISE-OS	Itegrated Squared Errors and OverShoot
ISE-ST	Itegrated Squared Errors and Settling Time
LQR	Linear Quadratic Regulator
MDP	Markov Decision Process
NLTA	Nonlinear Threshold Accepting
NN	Neural Network
PD	Proportional-Derivative

PI	Policy Iteration
PID	Proportional-Integral-Derivative
PSO	Particle Swarm Optimization
RC	Resistor-Capacitor
RL	Reinforcement Learning
SCARA	Selective Compliance Assembly Robot Arm
SIMO	Singal Input Multiple Output
SMDP	Simi-Markov Decision Process
TD	Temporal Difference
VI	Value Iteration
WMR	Wheeld Mobile Robot

Chapter 1

Introduction

1.1 Overview

Autonomous tracking is a class of control problems that requires the system to follow a desired reference or behavior without explicit human guidance. Ideally, the tracking process aims at dragging the tracking errors to zero. The problem here is that there are other dynamic characteristics which may also play an important role in the system performance. Moreover, prior knowledge of the system environment may not be available in advance in some situations. In this case, machine learning tools may be quite handy. Their learning process allows the system to accumulate experience and to automatically explore the operating environment.

In this thesis, three machine learning techniques and optimization heuristics are applied to design the control system. Reinforcement learning is a class of the machine learning technique which is concerned with how an agent can learn to choose actions rationally in an unknown environment to accomplish its goal. There is no explicit supervisor to guide the agent's learning process. The system is able to explore the state of the environment and take the appropriate actions to converge towards the target based on the reward returned by the environment. Feedforward artificial Neural Networks represent another class of artificial intelligence tools. They need a set of input-output tuples to train themselves to learn the environment behavior. Another technique, which we will be using in this research

for optimization, is the Nonlinear Threshold Accepting (NLTA) method. It is a heuristic for searching optimal solutions within a given range of search space.

1.2 Motivation

With the increasing complexity of modern autonomous systems, such as unmanned aerial vehicles, autonomous vehicles, unmanned underwater vehicles, etc., it has become more and more difficult to derive precise dynamic models for such systems. To overcome the overwhelming dynamic uncertainties typically characterizing such systems, the development of data-driven control schemes that are less dependent on the prior knowledge of the plant dynamics has become an urgent necessity.

Machine learning techniques have been widely applied to overcome this hurtle. In this thesis, we develop a model-free architectural framework for tracking control problems. To this end, three main model-free paradigms are investigated: Q-learning, nonlinear threshold accepting (NLTA), and artificial neural networks (ANN). The choice of these techniques is motivated by relaxing the requirement of the system's dynamic model.

A model-free control architecture is not the only challenge in hand. Conventional tracking systems often focus on the convergence of a tracking error while neglecting other characteristics which may also be very important from a control perspective. For example, some trackers may show a satisfactory performance in annihilating the tracking error but at the expense of a large control effort, a chattering behavior, or excessively long settling time, for example. It is part of the thesis goals to devise a control architecture that can tackle the tracking problem as well as optimizing a set of designer-specific control objectives in a unified approach.

1.3 Contributions

The main contribution of this thesis is the development of a model-free control architecture to track pre-defined reference signals while optimizing a system-wide cost function. The

proposed controller does not depend on the prior knowledge of the plant dynamics.

The findings of the thesis are partially disseminated in [87, 86].

1.4 Thesis Organization

The rest of the thesis is organized as follows:

- **Chapter 2** provides the necessary background information on the tools on which the research is founded; namely, reinforcement learning, nonlinear threshold accepting (NLTA), and artificial neural networks.
- **Chapter 3** reviews the relevant literature.
- **Chapter 4** details the proposed model-free architectural framework. Then, it demonstrates the system performance on a single-input single-output system.
- **Chapter 5** generalizes the concept to multi-input multi-output systems. After that, it assesses the controller's performance when applied to a two-input two-output system. The results are also benchmarked against another technique suggested in the literature.
- **Chapter 6** summarizes the conclusions and suggest a few possible future research directions.

Chapter 2

Background

The chapter introduces the techniques on which this research is founded; namely, reinforcement learning, artificial neural networks, and the nonlinear threshold acceptance heuristic. In section 2.1, a brief background on reinforcement learning is presented. It covers the goal and reward mechanisms, Markov Decision Processes, and two fundamental solutions for reinforcement learning problems: dynamic programming and temporal-difference learning. Section 2.2 provides an introduction on artificial neural network, while the nonlinear threshold accepting algorithm is discussed in Section 2.3.

2.1 Reinforcement Learning

Reinforcement learning is a computational approach to learn the optimal action (control signal) from the interaction between the agent (controller) and the environment (controlled system). As in several forms of machine learning, the learner is not told which actions to take, it must discover the most beneficial actions by trial-and-error search [79]. Unlike supervised learning, reinforcement learning does not have a knowledgeable external supervisor which provides examples to learn from, the agent must be able to learn from its own experience by interacting with the environment [79].

In the agent-environment interaction model of reinforcement learning, the agent is the learner and decision-maker. The environment comprises everything outside the agent [79].

At each discrete time step $t = 0, 1, 2, 3, \ldots$, the agent takes a state $s_t \in S$ as an input, which represents the state of the environment at t, where S is the set of all possible states. Based on the current state, the agent selects an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of available actions from state s_t. At the next time step $t + 1$, the environment passes on to state s_{t+1} after taking action a_t. The transition is accompanied by an numerical reward $r_{t+1} \in R$ which could also act as a penalty, depending on how "good" or "bad" is taking action a_t under state s_t. Fig. 2.1 shows the agent-environment interaction dynamics. During the learning process, the agent accumulates rewards for each state-action pair to form the agent's experience.

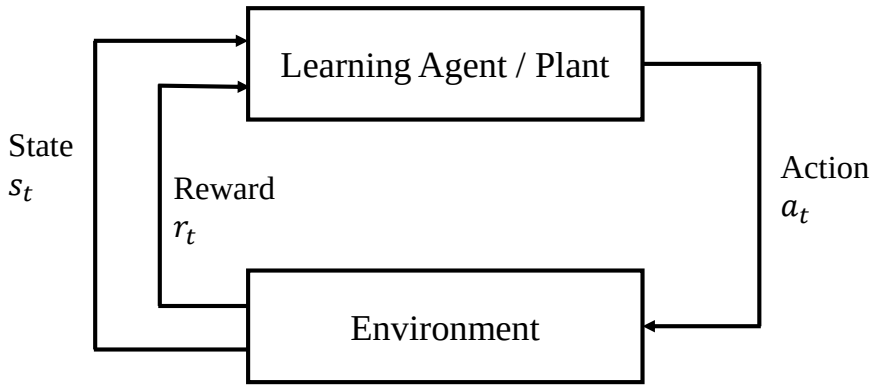


Figure 2.1: The agent-environment interaction scheme in RL

2.1.1 Goal and Rewards

The overall aim of the agent is to learn an action selection policy $\pi(s, a)$ (mapping from every state to every possible action) which yields the maximum long-term accumulated reward [79]. To influence the agent's behavior, a reward mechanism is adopted. The reward function applied is often custom designed based on the problem in hand. It is the goal of the agent to learn the environment's behavior by estimating and maximizing the long-term total expected reward.

Suppose that we have a reinforcement learning task with an associated terminal state

as shown in Fig. 2.2. Then, the cumulative reward R_t is defined as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_{t+T+1} = \sum_{k=0}^{T} r_{t+k+1} \tag{2.1}$$

where T is the terminal time step. Such a cumulative reward definition is applicable to a class of reinforcement learning tasks with a terminal state. In other words, the agent-environment interaction ends when it achieve this special state. In such scenarios, simulations or experiments are broken down to episodes. An episode is a sequence of agent-environment interactions which begin at time $t = 0$ and ends when the environment reaches the terminal state. Nonetheless, in some problems, terminal states do not exist. In such cases, the agent-environment interaction continues indefinitely ($T = \infty$). These are known as continuing tasks [79].

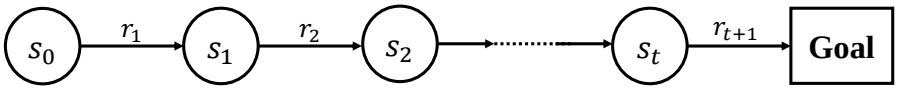


Figure 2.2: State transiting diagram.

A discounting concept may be used to favor immediate rewards over future ones in the cumulative reward function. An example of a discounted cumulative reward function is defined as follows

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \cdots \gamma^T\, r_{t+T+1} = \sum_{k=0}^{T} \gamma^k\, r_{t+k+1} \tag{2.2}$$

where γ is the discount rate, $0 < \gamma \leq 1$. The discount rate affects the present estimation of the future rewards [79]. If γ is close to zero, the agent strongly prefers the immediate reward to the rewards it gets in the future. On the other hand, when γ approaches 1, the agent takes greater consideration of the future rewards. Besides, since $\gamma < 1$, the sum of future rewards and the reward sequence r_k remain bounded as $T \to \infty$.

2.1.2 Markov Decision Processes (MDP)

In the most general cases, the response of the environment at time $t+1$ to the action a_t taken at time t, may depend on the past sequence history of the agent's states and actions. Such dynamics can be described probabilistically as

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} \tag{2.3}$$

for all every possible state $s' \in S$, reward r, and all possible values of early states, actions and rewards, where Pr represents the probability. A markov process is a special case of such systems. A process is said to satisfy the Markov property if its status at time $t+1$ only depends on that at time t. In this context, an environment satisfies the Markov property if the probability of its state s_{t+1} at time $t+1$ only depends on its state s_t and the action a_t. Formally, this defined by

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\} \tag{2.4}$$

for all $s' \in S$, r, $s_t \in S$ and $a_t \in A(s_t)$. Any reinforcement learning task that satisfies the Markov property is called a Markov Decision Process (MDP). Those tasks with finite state and action spaces are then called finite Markov Decision Processes (finite MDP).

Given a state s_t and action a_t at time t, the probability of each potential next state s' is called a transition probability and is defined as

$$P^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{2.5}$$

Similarly, we define the expected value of the next reward associated with taking action a at state s and transiting to state s' as

$$R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.6}$$

This reward function informs the agent how "good" or "bad" it is to take action a at state s immediately. The goal of the agent is to maximize the total long-term reward, not

the immediate reward at any specific time step. To this end, a value function is defined on each state-action pair indicating the long-term evaluation of any state (or desirability of the performance of an action under a state). This function is one of the pillars of almost all reinforcement learning algorithms. Since the value function reflects the long-term performance of the state-action pairs (i.e, particular policies), the agent gives it special consideration. During the learning process, the agent gets the instantaneous reward from the reward function and then uses it to compute the value function for each state s.

Define the state-value function $V^{\pi}(s)$ (the value of a state under a policy π) as the expected reward starting in state s and following policy π thenceforth as follow

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{ R_t \,|\, s_t = s \right\} \\ &= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \,|\, s_t = s \right\} \\ &= E_{\pi}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+2} \,|\, s_t = s \right\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma \, E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+2} \,|\, s_{t+1} = s' \right\} \right] \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma \, V^{\pi}(s') \right] \end{aligned} \tag{2.7}$$

where $\pi(s,a)$ is the probability of taking action a in state s according to policy π, $E_{\pi}\{\}$ is the expected value given that agent adopts policy π.

Similarly, we define the action-value function $Q^{\pi}(s,a)$, which is the expected reward

value of taking action a in state s under policy π, as follow

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\{R_t \,|\, s_t = s,\, a_t = a\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\, s_t = s,\, a_t = a\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|\, s_t = s,\, a_t = a\right\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma\, Q^{\pi}(s',a')\right] \end{aligned} \tag{2.8}$$

Equations (2.7) and (2.8) are the Bellman equations for the state-value equation V^{π} and the action-value equation Q^{π} under policy π, respectively. These equations balance all the possible circumstances by their probability of occurring. They also state that the value of the starting state must equal to the discounted value of the expected next state plus the long-run reward. Fig. 2.3 illustrates the backup relationship of the state-value function and the action-value function in a graphical way. Diagrams like these are called backup diagrams.

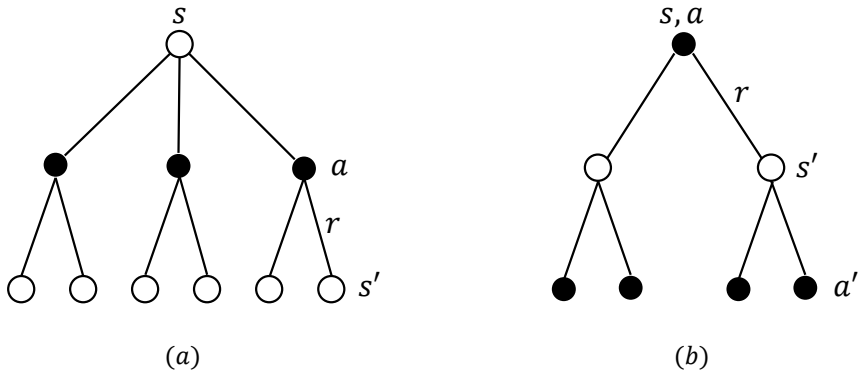


Figure 2.3: Backup diagrams for (a) V^{π} and (b) Q^{π}

Recall that the main goal of reinforcement learning tasks, is finding a policy that provides the highest long term reward. That is to say, they are required to find the policy which has a greater or equal expected cumulative reward than other policies for all states. Value functions validate the efficiency of all possible policies. A better policy associates a greater value function. A policy π is better than or equal to a policy π' if and only if the

expected cumulative reward brought by π is greater than or equal to that brought by π' for all states. In other word, policy π is better or as good as π' if and only if

$$V^{\pi}(s) \geq V^{\pi'}(s) \qquad \forall\, s \in S \tag{2.9}$$

A policy that maximizes the long-term reward is called an optimal policy π^*. An optimum policy may not be unique. The state-value function yielding the maximum reward is defined as the optimal state-value function V^*, and is given by

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.10}$$

Similarly, the optimal action-value function $Q^*(s, a)$ is defined as

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{2.11}$$

The optimal state-value function and the optimal action-value function must satisfy the Bellman equations (2.7) and (2.8), which can be rewritten as

$$V^*(s) = \max_{a} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right] \tag{2.12}$$

$$Q^*(s, a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right] \tag{2.13}$$

Equations (2.12) and (2.13) are called the Bellman optimality equations for the state-value function $V(s)$ and the action-value function $Q(s, a)$, respectively.

2.1.3 Dynamic Programming

The term dynamic programming (DP) refers to a class of algorithms that can be used to figure out the optimal policies when given a perfect dynamical model of the environment as a Markov decision process (MDP). As discussed in the previous section, value functions are the key entity to organize and structure the agent's searching mechanism for the best policy. This section introduces the dynamic programming algorithms that are used to

compute the value functions and to optimize the policies. The most common DP methods within the RL context are the Policy Iteration (PI) and the Value Iteration (VI) paradigms.

Policy Iteration

In the case of a finite MDP problem, the existence of an optimal policy is guaranteed since the number of policies is finite. Once a policy π is given, we can examine whether it is the best policy or not by trying an action $a \neq \pi(s)$ at state s and thereafter following the existing policy π. If this new policy π' with action a is better than to the initial policy π, then the value function $V^{\pi'}$ must be greater than or equals to V^{π} (i.e., $V^{\pi'} \geq V^{\pi}$). In this case, policy π' becomes a more optimal policy for the environment. To take all possible policies into consideration, we can repeat this process to find the optimal policy π^*. This iterative way is called the policy iteration. Suppose that for each state, we have a sequence of approximate value function V_0, V_1, $V2$, ..., where the initial approximation V_0 is chosen arbitrarily, then during the iteration process, the successive approximations are obtained using Bellman equation (2.7) as an update rule:

$$\begin{aligned} V_{k+1}(s) &= E_{\pi} \{ r_{t+1} + \gamma V_k(s_{t+1}) \, s_t = s \} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \, [R^a_{ss'} + \gamma V_k(s') \,] \end{aligned} \tag{2.14}$$

There are two main steps in a policy iteration strategy: policy evaluation and policy improvement. The former evaluates the current policy using the state-value function (2.14), while the latter is used to improve the current policy by trying a new greedy policy π', defined as

$$\pi'(s) \longleftarrow \arg\max_a \sum_{s'} P^a_{ss'} \, [R^a_{ss'} + \gamma V^{\pi}(s')] \tag{2.15}$$

Finally, the result converges to the optimal policy. The complete process is given in Algorithm 2.1 [79]. A small positive threshold $\theta > 0$ is used as a termination condition.

Algorithm 2.1 Policy Iteration Algorithm

Input:

A: Set of possible actions.
S: Set of possible states.
θ: a small positive threshold.

Output:

Learned policy π and value function array V.

Arbitrarily initialize $V(s)$ and choose a policy $\pi(s)$, for all states.

Step 1: Policy Evaluation
repeat
    $\Delta \longleftarrow 0$
    for each state $s \in S$ **do**
        $v \longleftarrow V(s)$
        $V(s) \longleftarrow \sum_{s'} P_{ss'}^{\pi(s)} \left[R_{ss'}^{\pi(s)} + \gamma V_k(s') \right]$
        $\Delta \longleftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$

Step 2: Policy Improvement
for each $s \in S$ **do**
    $b \longleftarrow \pi(s)$
    $\pi(s) \longleftarrow \arg\max_a \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V^{\pi}(s')]$
end for
if π does not converge (i.e. $b \neq \pi(s)$) **then**
    Return to Step 1.
end if

return policy π and value function array V

Value Iteration

Value Iteration is a simpler DP algorithm which solves the Bellman equation (2.12) recursively. Like Policy Iteration methods, it also needs comprehensive information of the state transition probabilities and the rewards for each transition. The update rule of value iteration can be written in a simple backup operation as

$$V_{k+1}(s) = \max_a \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V_k(s') \right] \tag{2.16}$$

for all $s \in S$.

The technique is described by Algorithm 2.2. Similarly to Policy Iteration, Value Iteration also requires an infinite number of iterations to converge to the optimal value function. A small positive threshold θ is introduced as a stopping criterion threshold (to assess the convergence progress by evaluating $\max_{s \in S} |V_{k+1} - V_k|$). After converging to a value function, the output deterministic policy is computed by

$$\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right] \tag{2.17}$$

for all $s \in S$, where $\arg\max_a$ indicates the action at state s that is associated with the maximum state-value function.

Algorithm 2.2 Value Iteration Algorithm

Input:

A: Set of possible actions.
S: Set of possible states.
θ: a small positive threshold.

Output:

Learned value function array V.

Arbitrarily initialize $V(s) \in \mathbb{R}$ for every state $s \in S$
repeat
    $\Delta \longleftarrow 0$
    for each state $s \in S$ **do**
        $v \longleftarrow V(s)$
        $V(s) \longleftarrow \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right]$
        $\Delta \longleftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$ ▷ The deterministic policy π is obtain using: $\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right]$
return value function array V

2.1.4 Temporal-Difference Learning

Like dynamic programming, Temporal Difference (TD) learning methods update the estimations after interactions between the agent and the environment without waiting for

a final outcome (i.e., waiting until the next time step $t+1$). TD methods can learn the optimal policy directly from their own experience. Unlike DP techniques, they do not need a perfect model of the environment [79]. The simplest TD method, known as TD(0), uses the instantaneous agent-environment interaction outcomes, the observed reward r_{t+1} and the estimate $V(s_{t+1})$, to form a target at time $t+1$ in the form of

$$\text{target} = r_{t+1} + \gamma V(s_{t+1}) \tag{2.18}$$

where the term "target" indicates the desirable direction of motion, and γ is the discount factor. The difference between the target and the previous estimate $V(s_t)$, known as TD error, is used to update the previous estimate, such that

$$V(s_t) \longleftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{2.19}$$

where $0 \leq \alpha \leq 1$ is the learning rate. It defines how much the agent learns from the newly discovered information (i.e., the reward and estimated $V(s_{t+1})$). In the case where $\alpha = 0$, the agent does not learn anything, while $\alpha = 1$ means that the newly discovered information are the only important information from which the agent learns. Fig. 2.4 shows the backup rule in a diagram form. TD methods are called bootstrapping methods because they update the state node s based on one sample transition to the immediately following state s'.

Figure 2.4: Backup diagram for TD(0)

There are two main classes of TD methods: on-policy and off-policy TD control techniques. Generally speaking, for on-policy methods, the policy that is being estimated is

used to make decisions during the training process. Nevertheless, off-policy methods do not have an expected policy to be followed. Their estimations are updated independently of the policy. One of the most popular off-policy TD method, known as Q-learning, is introduced below. It is heavily applied in this research, as shall be detailed in the subsequent chapters.

Q-learning

Q-learning, introduced by Watkins [90], is one of the most important developments in reinforcement learning. The objective in Q-learning is to estimate Bellman equation (2.13) (also called Q-value) recursively to get an optimal policy. The simplest form of Q-learning is the one-step Q-learning, where the target is defined as

$$\text{target} = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \tag{2.20}$$

Q-learning also uses the immediate new experience from the interaction as a target. The TD error in this case is the difference between the target and the current Q-value. The update rule for Q-learning methods is given by

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.21}$$

where r is the environment-returned reward of taking action a under state s.

The backup diagram of Q-learning is shown in Fig. 2.5. Since update rule (2.21) is updating a state-action pair, the top node (the root of the backup) must be an action node. Besides, the backup also defines the action associated with the maximum Q-value over all possible actions under the next state. The convergence of Q-learning is guaranteed (as proven in [89]).

Q-learning algorithms come in various forms. To ensure exploring new search areas in the search space, which have not been visited earlier, an ε-greedy algorithm is advised. This is important to avoid being trapped with a sub-optimal policy. During the learning process, the ε-greedy algorithm acts as an action selection guide to trade off between

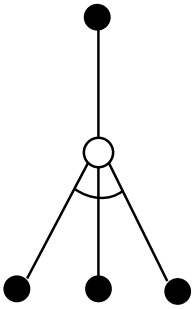


Figure 2.5: Backup diagram for Q-learning

exploiting known policies and exploring new ones, as described in Algorithm 2.3. The agent would either take a random action $a^r \in A$ (exploration) with probability ε, or follow the greedy action $a^g \in A$ (exploitation) with probability $1 - \varepsilon$, as follows

$$a = \begin{cases} a^r & \text{with prob. } \varepsilon \\ a^g & \text{with prob. } (1 - \varepsilon) \end{cases} \tag{2.22}$$

where the greedy action is the one maximizing the Q-value function $Q(s, a)$ under the current state s. That is,

$$a^g = \arg\max_a Q(s, a) \tag{2.23}$$

Q-learning Example

Let us take an example of a 4 $\times$ 4 grid world, shown in Fig. 2.6, to demonstrate how Q-learning can be used to obtain a deterministic optimal policy that will guide the agent to reach the goal state. Each cell in the grid represents a state of the environment. The agent is represented by a rocket. States 1 and 16 in gray are the terminal states of the game, as shown in Fig. 2.6a. There are 16 states in total, as shown in Fig. 2.6b.

The rules in this task are set as follows:

- At each state, the agent can take one of four possible actions: up ($\uparrow$), down ($\downarrow$), left ($\leftarrow$), and right ($\rightarrow$), with equal probabilities.

Algorithm 2.3 Q-Learning

Input:

A: Set of possible actions.
S: Set of possible states.
N_EPISODES: number of episodes.

Output:

Learned Q-table.

```
1: Initialize Q(s, a) arbitrarily.
2: Set the initial state s.
3: for p = 1 to N_EPISODES do                          ▷ Beginning of an episode
4:     repeat
5:         Choose action a based on policy derived from Q (e.g. ϵ-greedy).
6:         Taking action a, observe the next state s′ and the reward r.
7:         Update the action-value function                          ▷ using (2.21)
8:         s ⟵ s′.
9:         Move to next state s.
10:    until s meets a terminating condition.
11: end for
12: return Q-table Q
```

- Executing any one of the possible actions moves the agent one cell in the direction of the specified action.
- Any action that would take the agent off the grid (i.e., hitting the outer edges of the grid) will leave the agent in its current location. For example, executing action up ($\uparrow$) in state 3 keeps the agent in the same state.

The rewards are set to greedily increase as the agent approaches the terminal states, as shown in Fig. 2.7. For instance, assuming that the agent is in state 6, if it executes action up ($\uparrow$) or left ($\leftarrow$), it gets a reward of $r = +100$; and if it chooses to go right ($\rightarrow$) or down ($\downarrow$), it gets a reward of $r = +1$. That is to say, if an action leads the agent to move closer to the terminal states, it gets a higher reward. When the action leads the agent to the terminal states, it gets the highest reward of $r = +100000$.

The learning parameters are chosen as listed in Table 2.1. Since the learning rate $0 \leq \alpha \leq 1$ and the discount factor $0 < \gamma \leq 1$, we choose 0.5 (a middle value) as a starting point to observe its influence on the learning process. Such as choice forces the

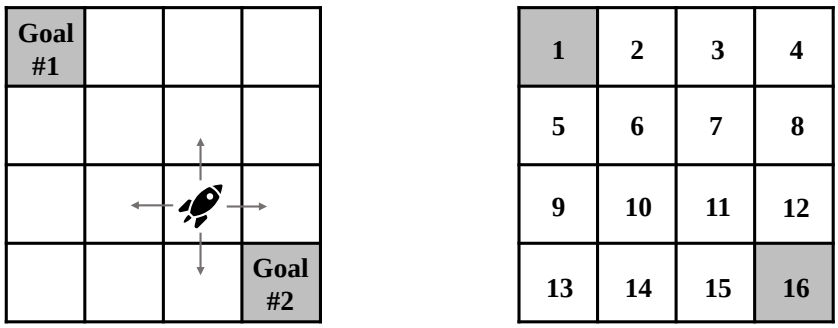


Figure 2.6: An example of Q-learning (a) 4×4 grid world (b) 16 states of the grid world

Goal #1	**100**	**10**	**1**
100	**10**	**1**	**10**
10	**1**	**10**	**100**
1	**10**	**100**	**Goal #2**

Figure 2.7: Reward of the grid world game

learning process to consider both the known and the newly discovered information, and discount the future rewards. As for the exploration probability, we set $\epsilon = 0.9$ to maximize the likelihood of exploring every possible state-action pair. The Q-table is initialized by zeros for all state-action pairs.

After approximately 90 iterations, the algorithm terminates (i.e., entries in the Q-table converge). The result is shown in Table. 2.2. The final optimal policy is the actions associated with the highest Q-value in the Q-table, as grayed out in the table. Fig. 2.8 visually describes the optimal policy to which the algorithm converged. For example, the agent learned that when it is in state 2, it is most rewarding to move left since that action has the highest reward of all 4 possible actions (see Table. 2.2).

This grid world example illustrates how to obtain an optimal control policy using an ε-greedy Q-learning algorithm. It also demonstrates the concepts of state, action, Q-table

Table 2.1: Learning Parameters

Parameters	Values
α	0.5
γ	0.5
ϵ	0.9

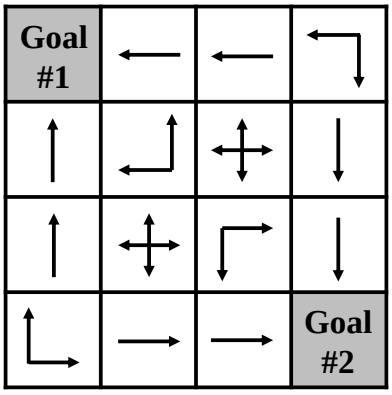


Figure 2.8: The optimal policy

and reward function in a concrete manner.

2.2 Artificial Neural Networks

Inspired by biological neurons, artificial neural networks are composed of multiple layers of interconnected processing elements (known as neurons or nodes) [34]. A feedforward neural network is organized into layers, as shown in Figure 2.9. The first and last layers are known as the input and output layers, respectively, while the layers in between are referred to as the hidden layers. In a feedforward network, each output or hidden node is interconnected with all the nodes in the preceding layer through some weights reflecting the connection strength between the nodes [80]. The higher the weight the stronger is the the firing connection. A neuron computes its output $a \in \mathbb{R}$ according to a firing mechanism

Table 2.2: The learned Q table ($\times 10^{-5}$)

states	Actions			
	up	down	right	left
1	--	--	--	--
2	0.5010	0.2506	0.2506	1.0000
3	0.2506	0.1253	0.1253	0.5010
4	0.1253	0.2506	0.1253	0.2506
5	1.0000	0.2506	0.2506	0.5010
6	0.5010	0.1253	0.1253	0.5010
7	0.2506	0.2506	0.2506	0.2506
8	0.1253	0.5010	0.2506	0.1235
9	0.5010	0.1253	0.1253	0.2506
10	0.2506	0.2506	0.2506	0.2506
11	0.1253	0.5010	0.5010	0.1235
12	0.2506	1.0000	0.5010	0.2506
13	0.2506	0.1253	0.2506	0.1235
14	0.1253	0.2506	0.5010	0.1235
15	0.2506	0.5010	1.0000	0.2506
16	--	--	--	--

defined by the following equation, as illustrated in Figure 2.10:

$$a = f\left(\sum w_i \cdot x_i + b\right) \tag{2.24}$$

where $x_i \in \mathbb{R}$, $i = 1, \ldots, d$, is the output of node i in the preceding layer; $w_i \in \mathbb{R}$ is the firing strength between the two nodes; $b \in \mathbb{R}$ is a bias term; and f is the node's activation function which can be chosen to the designer's preference. The activation function can be linear or nonlinear. In this example, d denotes the number of neurons in the preceding layer. In a supervised learning approach, the training process of a neural network is accomplished by adjusting the connection weights between the different layers [93]. It is based on an optimization process where the goal is to minimize a predefined error measure between the network's output and the target output. The learning process is often performed iteratively over several iterations, called episodes.

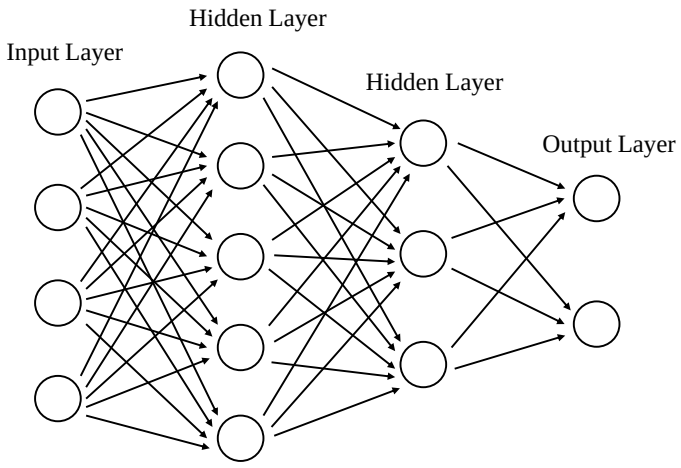


Figure 2.9: Example of feedforward neural network structure

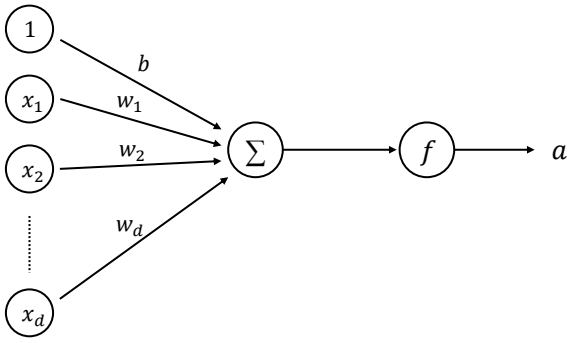


Figure 2.10: Firing mechanism in a feedforward neural network

Neural networks have been widely applied as artificial intelligence tools to solve many complex problems [13, 60, 48, 31, 70]. For instance, they were applied to forecast the electric loads in power systems where they outperformed other standard regression methods [62]. A class of adaptive control problems is solved using neural networks in [19]. An adaptation algorithm that employs an online neural network is developed for nonlinear flight control problems in [40]. A multi-dimensional signal processing problem is solved using an extended quaternionic feedforward neural network scheme in [53]. Algorithms based on Levenberg-Marquardt optimization schemes are utilized for the neural network training in [32, 30].

2.3 Nonlinear Threshold Accepting Algorithm

The nonlinear threshold accepting algorithm (NLTA) is a heuristic technique for solving non-constrained optimization problems. It was originally developed to solve NP-hard problems [56, 54]. It operates by performing a local search for an optimal solution within a predefined search space. At each search step, a candidate solution is randomly selected from the search space before it is compared to the current solution. After that, the algorithm decides on whether to reject the candidate solution, or accept it by adopting it as the best solution found so far instead of its current solution. The evaluation process is based on a nonlinear threshold accepting rule that is inspired by an analog low-pass filter (also known as an RC filter).

The RC filter is an analog circuit that is typically used to filter high-frequency noise within a signal. It consists of a resistor R in series with a capacitor C, as shown in Figure 2.11, where V_{in} and V_{out} are the filter's input and output voltages, respectively. The filter's transfer function is $H(s) = V_{\text{out}}/V_{\text{in}} = 1/(1 + s/\omega_0)$, where $\omega_0 = 1/(RC)$ is the filter's cutoff frequency.

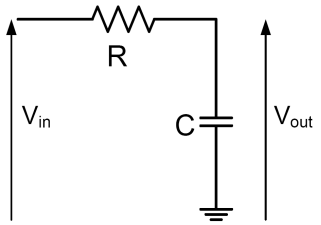


Figure 2.11: RC filter

The NLTA algorithm bases its nonlinear threshold accepting rule on the magnitude of the filter's transfer function in the frequency domain,

$$\|H(j\omega)\| = 1/\sqrt{1 + (\omega/\omega_0)^2}\ , \tag{2.25}$$

for some positive frequency ω. The magnitude function controls the convergence speed of the search process within the domains of the optimized variables.

Consider the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & x \in D \end{aligned} \tag{2.26}$$

where f is an objective function defined by

$$\begin{aligned} f\colon D &\longrightarrow \mathbb{R} \\ x &\longmapsto f(x) \end{aligned}$$

$D = (D_1 \times D_2 \times \cdots \times D_n) \subseteq \mathbb{R}^n$ is the function domain and $x = [x_1, x_2, \ldots, x_n]^T \in D$. Algorithm 2.4 describes how the NLTA heuristic attempts to solve the optimization problem defined in Equation (2.26). At each iteration, the algorithm randomly disturbs one of the elements of x and evaluates the resultant vector based on a nonlinear threshold acceptance rule, as explained in lines 6 to 8 of the algorithm. The frequency ω is decreased after each acceptance (as long as it does not run negative) which increases $\|H(j\omega)\|$ and so decreases the likelihood to accept non-improved solutions in the future [56].

2.4 Summary

In this chapter, we provided a brief introduction on the reinforcement learning (RL), neural networks (NN) and nonlinear threshold accepting (NLTA) technique.

The RL technique is modeled as an agent-environment interaction model, where the aim of the agent is to learn an optimal behavior (policy) to reach its goal as fast as possible. Unlike, supervised learning, such as the one used in feedforward neural networks, the agent in RL learns by a quasi trial-and-error mechanism built on maximizing an expected future reward function. RL comes in several variants. Among the most popular of them are Policy Iteration (PI), Value Iteration (VI), and Q-learning. PI is an algorithm that directly exploits the policy. It continuously evaluates the current policy and tries to improve it in an ongoing iterative process until the resultant policy converges to what is deemed close

Algorithm 2.4 NLTA

Input:

f: Objective function.
D: Search range of x.
$\omega_0 > 0$: a cutoff frequency for the low-pass filter.
$\Delta\omega > 0$: a frequency discount value.
$\omega_1 > 0$: an initial value for ω.
N_ITERATIONS: number of iterations per episode.
N_EPISODES: number of episodes.

Output:

NLTA's best guess of $\arg\min_x f(x)$.

for $p = 1$ to N_EPISODES **do** ▷ Beginning of an episode
  Randomly initialize $x = [x_1, x_2, \ldots, x_n]^T \in D$
  $f_o^p \leftarrow f(x)$ ▷ Smallest objective function value in the episode
  $\omega \leftarrow \omega_1$
  for $i = 1$ to N_ITERATIONS **do** ▷ Beginning of an iteration within an episode
    Randomly select $j \in (1, 2, \ldots, n)$ and $y_j \in D_j$
    Set $x' \in D$ such that $x'_i = x_i$ if $i \neq j$ and $x'_i = y_i$ if $i = j$, for $1 = 1, \ldots, n$
    if $\frac{f(x')}{f_o^p} \leq 1$ **or** $\frac{f(x')}{f_o^p} \leq \frac{1}{\|H(j\omega)\|}$ **then** ▷ The second condition allows for search space exploration to avoid getting trapped in local minima
      $x \leftarrow x'$
      $f_o^p \leftarrow f(x')$
      if $(\omega - \Delta\omega) > 0$ **then**
        $\omega \leftarrow \omega - \Delta\omega$ ▷ To control the convergence speed of the search process
      end if
    end if
  end for
  optimalSolution$(p) \leftarrow x$ ▷ Optimal solution for episode p
  optimalFunctionValue$(p) \leftarrow f_o^p$
end for
$q \leftarrow \arg\min_p$ optimalFunctionValue(p)
$x^o \leftarrow$ optimalSolution(q) ▷ NLTA's optimal solution
return x^o

enough to the optimal policy. On the other hand, the VI algorithm tries to find the optimal policy indirectly through an optimal value function. Q-learning is one of the model-free RL algorithms. Unlike PI and VI, Q-learning does not need a perfect model of the dynamic model environment. The Q-learning algorithm is composed of a series of states, actions and rewards. It uses a Q-value to estimate the "quality" of an action taken at a given state.

The actions with the maximal Q-values are adopted to form the final optimized policy.

Feedforward ANNs are known to be excellent function approximators. They owe their reputation primarily to the Universal Approximation Theorem [33], which states that, in theory and given certain conditions, a feedforward ANN is capable of approximating any real-valued continuous function to any desired degree of accuracy. ANNs base their learning process on supervised learning which requires a set of input-output tuples to guide the network through the training process.

The NLTA heuristic is an iterative local search algorithm which investigates a nonlinear accepting rule to improve the conventional threshold accepting algorithm. It continuously search for a better solution that leads to higher objective function value. The search process starts with a random solution within the search space. It then explores the solution domain by jumping randomly within the neighborhood of the current solution. A new solution is accepted if it satisfies a certain peculiar accepting rule which is based on the principle of an analog low-pass filter.

In this thesis, we will design and compare different control architectures integrating various combinations of Q-learning, NLTA, and ANN, to tackle the tracking control problem without the need for a plant model. This will be discussed further in Chapters 4 and 5.

Chapter 3

Literature Review

This chapter introduces a literature review about machine learning and artificial intelligence approaches, including nonlinear threshold accepting heuristic, neural networks, and reinforcement learning techniques. It also presents some adaptive mechanisms for different nonlinear control systems.

3.1 Machine Learning and Artificial Intelligence

The artificial intelligence sciences tackle and improve the system's abilities to accurately interpret external data, learn from such data, and hence employ this knowledge to self-adapt the dynamic states in order to achieve better performance, specific designed goals and tasks [39]. Additionally, the machine learning field is a subclass of these sciences, it allows computer algorithms to learn how systems can perform the tasks and improve performance automatically within an experience-learning dynamic environment. Machine learning and artificial intelligence tools have been widely used to solve different optimization problems and applications that involve single as well as multi-agent systems. Additionally, they are employed to solve the optimal and cooperative control problems. In this section, literature review about machine learning tools, namely nonlinear threshold accepting heuristic, neural networks, and reinforcement learning is briefly conducted. Finally, some Q-Learning-based applications are highlighted.

3.1.1 Nonlinear Threshold Accepting Method

The Nonlinear Threshold Accepting (NLTA) method is developed by Nahas and Nourelfath [56]. It relies on a nonlinear accepting threshold criterion mainly formed using a transfer function of an analog low-pass filter to solve NP-hard problems in [56]. The NLTA algorithm is employed to solve the non-convex economic dispatch problems in power systems with valve point loading effects, multiple fuel option, prohibition zones, constrained power generation levels, etc. in [55]. Furthermore, it is applied to tackle the redundancy allocation problems by enhancing the reliability of the underlying systems in [57]. This approach is employed to tune the control gains for a multi-area power system generation network in order to solve a combined automatic voltage and load frequency regulation problem in [54]. The results showed that this accepting algorithm outperformed other analytical solutions as well as standard heuristics, such as Internal Model Control (IMC), Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), Differential Evolution Algorithm (DEA), to name a few, in terms of the closed-loop time-response characteristics. This heuristic approach will be detailed and used in the following chapters.

3.1.2 Artificial Neural Networks

Artificial neural networks are computing structures inspired by the human nervous and brain systems. These networks are adopted widely to model how the information or knowledge might be captured, represented and processed to achieve a given goal [36]. Parameters characterizing the neural networks, like the number of layers, the number of neurons in each layer, activation functions, connection weights, and bias values, influence the quality of the trained networks [66]. The neural networks are proved to be useful in various applications. During the last two decades, deep neural network approaches have been utilized in pattern recognition applications. They have been integrated into many machine learning fields [73]. A deep learning mechanism which employed convolution neural networks is used for an intelligent constellation diagram analyzer in [85]. Other deep convolution neural networks employed ImageNet databases for image classification purposes and have shown satisfactory performance in [44]. A sliding-mode control scheme based on neural networks

is developed for a multi-rigid link robot manipulator in order to control the position of that system in [84]. In [91], a tracking controller based on neural networks is designed for omnidirectional wheeled mobile manipulators where a neural network is applied to identify the uncertain characteristics of the system. An adaptive tracking controller based on recurrent neural networks is developed for remotely operated vehicles in [20]. Further, a combined fuzzy neural network approach is used to develop an unmanned control system for a flexible wing aircraft in [6].

3.1.3 Reinforcement Learning

Reinforcement learning is a class of machine learning mechanisms. It is known as a computational approach to learn the best strategies from interactions between the agent and its dynamic environment. The agent or learner discovers the best strategies by balancing exploration and exploitation within the search environment [79, 37, 42]. Unlike supervised learning, RL does not require a knowledgeable external supervisor to learn from. Instead, the agent learns from its own experience formed along the learning process [79, 41]. An off-policy temporal difference learning approach is one that employs Q-Learning to find the optimal strategies to solve the optimization problem [90]. On the other hand, Integral Reinforcement Learning (IRL) structures are developed using online model-free policy iteration algorithms for continuous-time optimization problems to solve the differential graphical games in [3, 5, 4]. Additionally, online value iteration mechanisms are developed in order to solve this type of dynamic games in [7, 9]. In another context, a non-convex Q-Learning technique with eligibility traces is used to solve the non-convex power system economic dispatch problems in [8]. It exhibited competitive results when compared with other standard heuristic solutions for the same problem.

The Q-Learning mechanism is widely employed to solve many optimization problems. Some of its applications are highlighted below.

Box Pushing Task

Generally, the box pushing problem involves three tasks [52]: First, a robot finds a box and distinguishes it from other obstacles, such as walls. Second, it pushes the box across the room while keeping the box in the center of its sight view. Finally, the robot learns to un-wedge from stalled states. This means that it has to learn how to get the box out of the corner or stop pushing the box against a wall. Mahadevan and Connell [52] combined a Q-learning idea with clustering techniques to train a mobile robot to accomplish the box pushing task. They defined three modules and designed different reward functions for each task. The robot was able to learn individual behaviours and react to different dynamic situations after the training process is complete. In addition, Wang et al. [88] employed Q-learning in a more sophisticated multi-robot box pushing application. In this work, multiple robots collaborate in order to push one box to a specified target location, where the walls and obstacles are still considered. They showed that the single-agent Q-learning mechanism outperformed a team-based Q-learning one in a complex dynamic environment. The single-agent Q-learning algorithm was extended to a multi-agent or multi-robot case as well as the team Q-learning process. The simulation results showed that these two methods worked effectively in a simple environment but the single-agent algorithm exhibited better performance when the environment included many obstacles since it employed more exploration which helped the learning process to escape local maxima.

RoboCup

RoboCup soccer is a well-known international robotics competition. The underlying reinforcement learning problems handle many challenges like how to train multiple agents and let them collaborate simultaneously and tackle the uncertainties associated with the state space of the system [78]. Researchers have suggested efficient reinforcement algorithms to solve these challenges in robotic competitions as well as research investigations. Stone et al. [78] introduced an algorithm based on an episodic SMDP Sarsa(λ) combined with linear tile-coding function approximation and variable λ in order to implement higher-level decision making process, called keep-away. In keep-away, the "keepers" have to keep the

ball in their hands as long as possible. The results showed that, after training the robotic team using that algorithm, the agents significantly outperformed a hand-coded policy in a complex, stochastic, dand ynamic environment. Yao et al. [92] combined a Q-learning technique with an online adversarial planning to solve the kicking problem in RoboCup. The simulation results showed that this combined approach not only achieved a satisfactory learning outcome, it also improved the adversary kick problem solution. Riedmiller et al. [69] successfully improved the traditional Q-learning process using Neural Networks to train the robotic team's dribbling skill. Upon making a U-turn with the ball, the neural dribbling optimization unit is shown to be much faster and sharper than the hand-coded one.

Elevator Group Mechanism

Another Q-learning application is the elevator group control. The elevator is driven by a passenger where the arrival patterns vary a lot during the day [24]. In a business-based building, there exist peak-up traffic in the morning and peak-down traffic in the afternoon. In a collage library, up and down traffic volumes depend on the start and finish time of classes. Thus, due to uncertainties in the passenger arrival rates, the states of this elevator system are not fully observed and they are non-stationary. Additionally, in real-life, there could be more than one elevator serving the same building. Hence, it is required to solve challenges like how to control the elevators efficiently and how to reduce the waiting times. To tackle such challenges, Crites and Barto [23] developed an algorithm which employed a team of discrete-event Q-learning agents, where each agent was designed to control one car system. The simulation results showed that the algorithm outperformed other heuristic solutions in order to optimize the up and down traffic [23].

3.2 Control Mechanisms for Nonlinear Systems

This section highlights some control mechanisms developed using classical and machine learning-based control tools for nonlinear systems. The main control objective reviewed

by this section is relevant to the reference-regulation schemes developed for nonlinear systems like unmanned aircraft, robot manipulators, mobile robots, and inverted pendulum systems.

In reference-tracking problems, a reference trajectory is available to the system, and the objective is to let the system follow that trajectory [2, 10, 68]. The solutions are dependent on dynamical forms of the tracking errors (i.e., the difference between the desired trajectory and the actual one).

3.2.1 PID Control Schemes

Proportional-Integral-Derivative (PID) controllers have been widely used in industrial control processes thanks to their simplicity and low cost [11, 45, 61, 76, 54]. Despite being mainly used in linear systems, they were also successfully integrated in some nonlinear systems. They are generally governed by three control gains which have been traditionally determined either analytically or through heuristic approaches. Elmer Sperry introduced a PID control scheme in 1911 and used it to solve the steering problem of an automatic ship. Nicholas Minorsky designed a PID scheme in 1922 for another purpose [14]. The PID control mechanisms have been applied in several applications, like manipulator control in robotic arms [15, 63], control of unmanned aerial vehicles and quadrotors [71, 81], industrial hydraulic regulators [67], temperature control [75], and many others. The closed-loop characteristics of a system with a PID control unit are dependent on the values of the control gains (i.e., K_p, K_i, K_d).

3.2.2 General Reference-Tracking Control Schemes

Many nonlinear tracking control schemes have been utilized in different robotic applications. In [77], a sliding mode feedback control mechanism is developed for a class of nonlinear tracking control problems. It was successfully applied on a two-link robot manipulator. A back-stepping control approach is developed for a two-degree-of-freedom mobile robot in [35] where it exhibited exponential global tracking convergence characteristics. A back-

stepping technique is employed to design an adaptive fuzzy tracking control scheme for a class of uncertain multi-input-multi-output nonlinear systems in [18]. An optimal tracking control structure based on a greedy heuristic dynamic programming iteration algorithm is proposed for a class of nonlinear systems in [94]. In [17], an adaptive tracking control mechanism is developed for robotic systems using Jacobian matrices subject to uncertain kinematic and dynamic environments. Another tracking control scheme that uses error dynamics within a back-stepping framework is discussed in [68]. It aims at solving 2D-trajectory tracking problems for autonomous under-actuated underwater vehicles.

3.2.3 Examples of Reference-Trajectory Tracking Problems

Robot Manipulators

Robot manipulators are widely used in industrial applications [82]. However, their highly nonlinear kinematics and dynamics make the design of the underlying control laws complex and challenging. A nonlinear adaptive torque-based method is developed to control the rigid links of robot manipulators by Craig et al. in [22]. An adaptive Proportional-Derivative (PD) controller is used to solve a robotic arm point-to-point control problem which exhibited global convergence results in [83]. Lewis [46] designed a model-free adaptive intelligent multi-loop controller for rigid robot arms using a PD controller as an outer loop to achieve the position control task and a neural network structure as an inner loop to estimate the robot operation. The simulation results showed that the position control algorithm worked effectively for a sinusoidal trajectory. Another H_∞ reinforcement learning controller based on adaptive critics and a fuzzy wavelet network was developed in [49] for the position-tracking of a robot manipulator with no prior knowledge about its dynamics. Simulations for a three-degree-of-freedom SCARA robot arm confirmed the usefulness of the developed controller. An adaptive Jacobian-based controller is suggested by Cheah et al. in [17] to solve an object-tracking problem of a robot with uncertain kinematics and dynamics. They introduced a novel update mechanism that uses sensory feedback measurements of the end-effector position to update the uncertain kinematics parameters. The experimental results showed that the end-effector successfully followed the desired

trajectory.

Mobile Robots

Mobile robots with non-holonomic dynamics are usually driven by wheels, where independent actuators are used to generate the necessary torque forces for the wheels [43]. Kanayama et al. [38] developed a tracking control method based on Lyapunov stability criteria for a mobile robot. Fierro et al. [27] proposed another robust adaptive controller based on a combined kinematic/torque control law that is developed using back-stepping and neural network structures. A PID controller was employed to achieve a reference-tracking goal in [59]. An adaptive tracking controller for a mobile robot with unknown dynamic parameters is designed using an adaptive back-stepping approach in [29]. Luy [51] designed an actor-critic control scheme based on an online synchronous policy iteration algorithm for a Wheeled Mobile Robot (WMR). The resulting value function is approximated by a critic neural network structure while the control law is approximated by an actor neural network. The convergence and stability properties are verified by means of Lyapunov theory. The simulation results showed the asymptotic convergence of the tracking errors after a relatively small time interval.

Aircraft Control

The uncertainty in the aircraft operating environment makes it challenging to find the right or optimal control policy. Hence, machine learning tools play a vital role to solve different aircraft control problems [2]. A model-based synchronized trajectory-tracking controller that includes a feed-forward compensation unit and a PD feedback controller was designed for an experimental three-degrees-of-freedom helicopter in [74]. Liu et al. [50] proposed an adaptive neural fuzzy inference system for the aircraft's landing task. An online model-free adaptive control mechanism based on a value iteration reinforcement learning process and a neural network was developed for the automatic control problem of a flexible wing aircraft in [2]. A pitch-roll tracking scheme with a roll disturbance-rejection control unit is developed to asymptotically minimize the trajectory-tracking errors in [2].

Autonomous Helicopter Control

Autonomous Helicopter Control (AHC) is a high-dimensional, nonlinear, and highly stochastic control problem [58]. Helicopters are widely used in military, rescue missions, agriculture and many other applications. Further, typical AHC is challenging when it boils down to the involvement of artificial intelligence to design the control schemes. Bagnell and Schmeider [12] used RL search methods to control the "core dynamics" of the helicopter, the pitch, roll and horizontal translations. The controller delivered a good performance in a real-world application. Ng et al. [58] designed a controller based on reinforcement learning approach to control the helicopter in a low-speed sustained inverted hovering mode. Their helicopter was also capable of normal flight. The learning software was shown to be quickly adaptable upon changing the operation mode of the helicopter. Abbeel et al. [1] designed a Differential Dynamic Programming (DDP)-based controller for four new aerobatic maneuvers which are challenging autonomous flight maneuvers.

Balance Systems and Reinforcement Learning

Balance systems are complex, nonlinear and unstable systems typically abstracted by a pole pivoting on a moving cart, robot arm or even a quadrotor [65]. The main objective is to control the underlying moving systems to stabilize the pole so that it maintains an upright position. Schaal [72] pointed out that model-based reinforcement learning embedded within a predictive model scheme can efficiently train the robot to balance the pole. Peters et al. [64] developed an adaptive critics-based pole-balancing mechanism where the policy is improved using the approximated value function in the policy evaluation loop. Another example is by Figueroa et al. [28], where an inverted pendulum is balanced on the top of a flying quadrotor. The control policy was generated using a reinforcement learning algorithm for high-dimensional input-states, called Continuous Action Fitted Value Iteration (CAFVI). This algorithm allowed the controller to learn the best strategies using continuous inputs. The simulation results demonstrated the controller's usefulness.

3.3 Summary

This chapter briefly highlights the literature review about some categories of the artificial intelligence sciences and their applications. In reference to the research conducted in this thesis, some reference-tracking regulation problems are described. This is useful to understand and design intelligent control mechanisms for nonlinear systems throughout the thesis. The necessary details about the nonlinear threshold accepting heuristic, reinforcement learning mechanism, and neural network optimization tools will be provided subsequently.

Chapter 4

Optimized Tracking Control Structures: An Aircraft Case Study

Solving tracking control problems often focuses on finding the optimized control policy that minimizes the tracking errors. However, this approach does not guarantee optimality of the overall tracking performance for the underlying dynamic system. For instance, a controller may be successful in annihilating the tracking error but at the cost of an excessively large or chattering control signal. In this chapter, the tracking problem is reformulated to reflect two sub-optimization tasks. The first assesses the environment for the best tracking control strategy, while the second searches for a feedback signal that would optimize a pre-defined cost function. The tracking algorithm is implemented using two techniques, Reinforcement Learning (RL) and Nonlinear Threshold Accepting Algorithm (NLTA). Meanwhile, a multi-layer feed-forward neural network scheme is introduced to find a feedback control signal that improves the dynamical characteristics of the system during the tracking process. After the offline tuning of the tracking and optimization control strategies at a first stage, the overall control mechanism is applied in real time to the system in hand.

This chapter is organized as follows: Section 4.1 briefly introduces the problem will be solved in this chapter. Section 4.2 presents the overall system architecture. Section 4.3 details the two proposed tracking controllers, which are based on RL and NLTA. An optimization scheme based on a feedforward neural network is introduced in Section 4.4 to

optimize the total cost function. The tracking and optimization units are integrated into an aggregate control system in Section 4.5. Section 4.6 validates the performance of the proposed control schemes using different simulation scenarios. Finally, a few concluding remarks are made in Section 4.7.

4.1 Problem Formulation

A dynamical system can be expressed by a state-space model representing the dynamics governing it. The model can be either linear or nonlinear in the system states and inputs. However, since controlling linear systems is often a much easier task than nonlinear systems, the latter are sometimes linearized into a linear approximation of the original system around a certain operating point. A linear (or linearized) state-space model in the discrete-time domain can be described as follows:

$$X_{k+1} = A\,X_k + B\,u_k \tag{4.1a}$$

$$y_k = C\,X_k + D\,u_k \tag{4.1b}$$

where X_k, u_k and y_k, are the state vector, input vector, and output vector at time index k, respectively. A, B, C and D are the system, input, output and feedforward matrices, respectively.

In a typical tracking problem, a desired output y_k^d is given to the system in advance. The system is then required to follow the desired output by minimizing the difference (i.e., error e) between the desired and the actual output. The tracking error at time index k can be expressed as

$$e_k = y_k^d - y_k \tag{4.2}$$

In the context of this thesis, we define the tracking problem as that of finding the optimal control input u_k so as to minimize $\|e_k\|$ (the Euclidean norm of e_k) as $k \to \infty$.

Signal tracking is a classical problem in control theory. An abundance of solutions has been suggested in the literature. However, the vast majority of the proposed solu-

tions assume full or partial knowledge of the system's state-space model. Although this assumption is realistic in many cases, it is sometimes too difficult or impractical to develop representative models of highly complex systems. The aim of the thesis is tackle the tracking problem without relying on the prior knowledge of the system model. This is accomplished in this chapter by adopting two different methods based RL and NLTA techniques. The RL scheme reformulates the tracking error combinations into a form of Markov-Decision-Process (MDP) that uses Q-Learning to build the best tracking control policy for the dynamic system under consideration. In the second method, the NLTA is applied to tune the gains of a PID controller. Finally, a Q-Table is prepared to mimic the global optimization of the dynamic process in hand, where the control decisions are chosen based on an objective function that penalizes the future dynamics as well as the current control efforts. This table is then utilized to train a multi-layer feedforward neural network scheme in order to achieve a mapping between the measurable states and the underlying best approximate control signal.

4.2 Control Architecture

This section lays out the architecture of the developed optimized tracking mechanisms for a class of dynamic systems. The objective of the tracking control problem is to find the best control strategies by optimizing the tracking error. Two methods are proposed for this purpose. The first is based on RL while the second is based on NLTA. Both controllers are supplemented with a neural network to optimize an overall cost function.

4.2.1 Tracking Control

RL provides a decision making mechanism where the agent learns its best strategy u_ℓ in a dynamic environment in order to transit from one state E_ℓ to a new state $E_{\ell+1}$ while maximizing a reward $R_{\ell+1}$, as shown in Figure 4.1. The RL-based controller is founded on a Markov Decision Process employed to decide on the tracking control signals, as shown in Figure 4.2. Given a vector $X_k \in \mathbb{R}^n$ of measurable states at a discrete time index k,

the tracking process compares the output $r_k^{actual} \in \mathbb{R}$, which is also one of the states (i.e., $r_k^{actual} \in X_{k\{j\}}, j \in \{1, \ldots, n\}$), with its desired value (i.e., reference signal) $r_k^{desired} \in \mathbb{R}$ and computes the resulting tracking error $e_k = r_k^{desired} - r_k^{actual} \in \mathbb{R}$. The respective tracking control signal $u_k^{RL} \in \mathbb{R}$ is decided by

$$u_k^{RL} = max(Q\{E_\ell, u^{RL}\})$$

where $Q(\ldots)$ is a mapping from an evaluation state $E_\ell = \{e_{k-3}, e_{k-2}, e_{k-1}, e_k\}$ to the associated best tracking control strategy u^{RL} from a range of feasible discrete values using an optimized Q-Table learned from an RL process, while ℓ is the state-index.

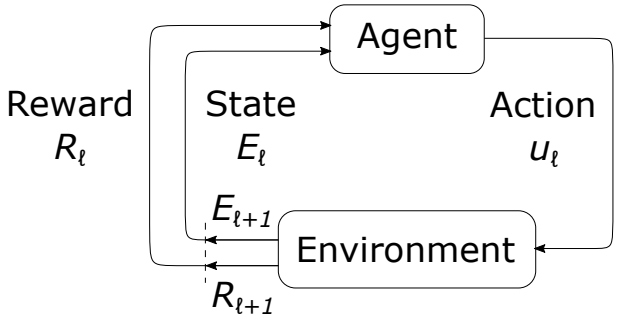


Figure 4.1: Agent-environment interaction.

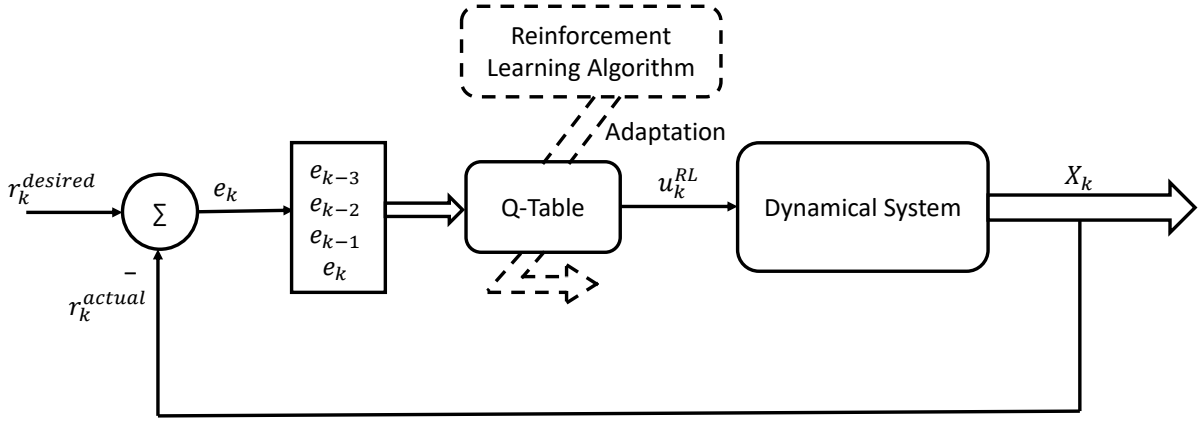


Figure 4.2: RL tracking control scheme.

The NLTA-based controller is similar in architecture as its RL-based counterpart. However, instead of the RL, it uses the NLTA algorithm [54] to automatically tune the gains (K_p, K_i, K_d) of a PID controller, as shown in Figure 4.3. The resulting control signal

$u^{NLTA} \in \mathbb{R}$ takes the following form:

$$u^{NLTA}(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d\, \frac{de(t)}{dt}, \tag{4.3}$$

where $e(t)$ is the continuous-time tracking error at time t. The training of the NLTA-based controller is conducted in continuous time because NLTA is a continuous-time algorithm in nature. However, the controller is discretized into a discrete-time system after the training phase so that it can be integrated into the aggregate control structure, as shall be explained later. That way, the RL- and NLTA-based controllers can be benchmarked on a fair basis.

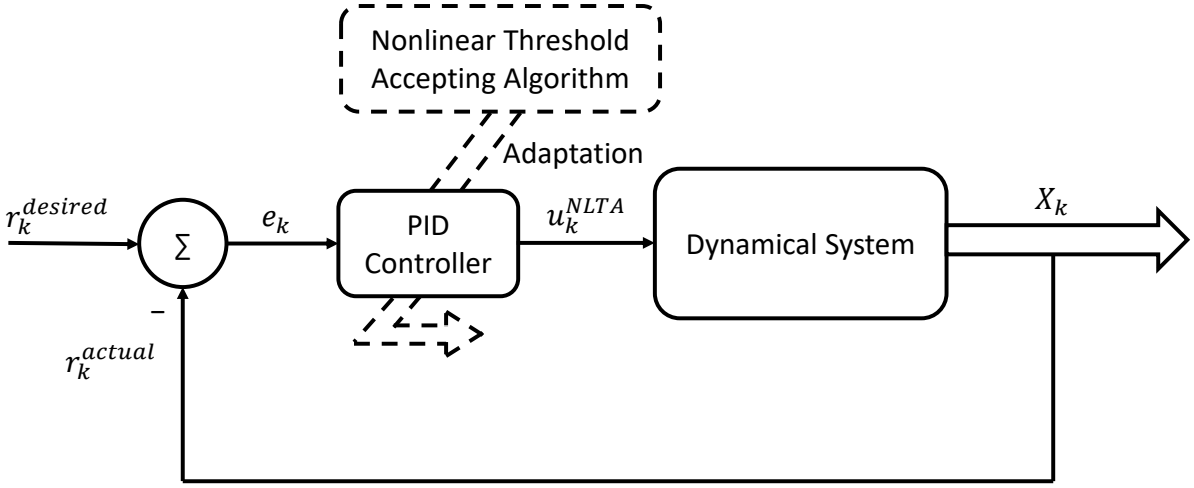


Figure 4.3: NLTA tracking control scheme.

4.2.2 Cost Function Optimization

Standalone tracking processes usually overlook the optimization characteristics of the overall dynamic performance alongside the tracking process. Therefore, it is useful to adjust the dynamic tracking control signal in order to improve the overall optimization features of the underlying control system. In this section, this is achieved through an auxiliary control signal $u_k^{NN} \in \mathbb{R}$, as shown in Figure 4.4. To this end, a deep neural network is trained using an optimized performance criteria. Unlike the tracking control schemes of Figures 4.2 and 4.3, the neural network takes the full state feedback measurements X_k as

input in order to advise the supporting control signal u_k^{NN}, such that

$$u_k^{NN} = f(X_k)$$

where f is the input-output mapping of the neural network.

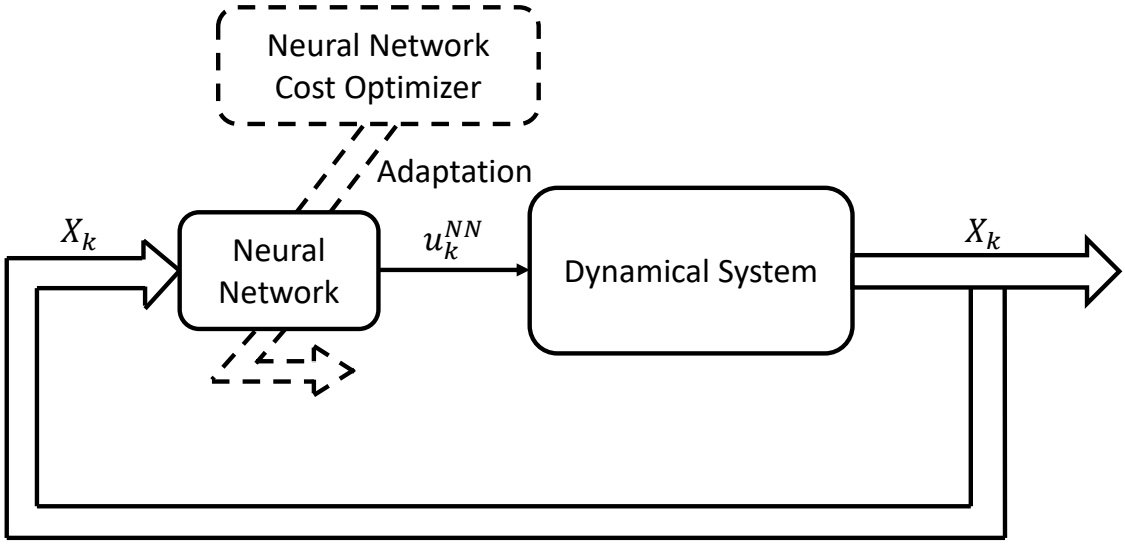


Figure 4.4: Neural network-based optimization control scheme.

4.2.3 Agrregate Control System

Each of the above tracking control schemes is augmented with the optimization control system of Figure 4.4. As such, the overall control laws of the RL- and NLTA-based controllers are defined as $u^T = u^{RL} + u^{NN}$ and $u^T = u^{NLTA} + u^{NN}$, respectively. The dynamical system (i.e., the plant) may be defined in the time domain by either a linearized continuous- or discrete-time state-space model:

$$\dot{X}(t) = A^c\, X(t) + B^c\, u(t) \tag{4.4a}$$

$$X_{k+1} = A^d\, X_k + B^d\, u_k \tag{4.4b}$$

where the superscripts "c" and "d" denote the continuous- and discrete-time matrices, respectively. For the purpose of this study, which is conducted in the discrete-time domain,

the plant model needs to be discretized if presented in the continuous-time domain. In the following, we will detail the learning paradigms of each of the control schemes.

4.3 Tracking Control Algorithms

4.3.1 RL-Based Tracking Control Algorithm

The RL process trains the tracking controller to opt for the optimal policies that minimize the tracking error. This is done through an iterative training process which continuously updates a Q-Table by penalizing or rewarding the taken actions. The table keeps track of the maximum expected future rewards for each feasible state-action pair (E_ℓ, u_ℓ^{RL}), where $E_\ell = [e_{k-3}\ e_{k-2}\ e_{k-1}\ e_k]^T$. An ε-greedy algorithm is applied to trade off between the exploitation and exploration of the training process. The update of the Q-Table entries follow the Q-Learning pattern, defined by [79]

$$Q(E_\ell, u_\ell^{RL}) = Q(E_\ell, u_\ell^{RL}) + \alpha\,[R(E_\ell, u_\ell^{RL}) + \gamma \max_{u_{\ell+1}^{RL}} Q(E_{\ell+1}, u_{\ell+1}^{RL}) - Q(E_\ell, u_\ell^{RL})], \quad (4.5)$$

where ℓ is the agent state index, α is a learning rate, γ is a discount factor, and $R(E_\ell, u_\ell^{RL})$ is the training reward for taking action u_ℓ^{RL} at state E_ℓ. The process is detailed in Algorithm 4.1. Note that the accuracy of the obtained control signals u^{RL} depends on the discretization steps of the states and the tracking errors. The finer is the resolution, the longer the learning may take.

Algorithm 4.1 Reinforcement Learning: Offline Computation of Q-Table

Input:

Minimum and maximum bounds X_{min}, X_{max}, u^{RL}_{min}, u^{RL}_{max}, E_{min}, and E_{max} of the variables X, u^{RL}, and e, respectively.

Discretization steps ΔX, ΔE, and Δu^{RL} of the variables X, e and the control signal u^{RL}, respectively.

Reinforcement learning parameters ε, α and γ.

Output:

Q-Table

1: Initialize the Q-Table randomly
2: **repeat**
3: Update the entries of the Q-Table using (4.5)
4: **until** A convergence criterion is met
5: **return** Q-Table

4.3.2 NLTA-Based Tracking Control Algorithm

With the NLTA-Based Tracking method, a PID controller is applied where the PID gains are tuned using an NLTA algorithm [54]. The NLTA technique adopts the concept of a low-pass filter, whose transfer function is $H(s) = 1/(1 + s/\omega_0)$ for some nominal frequency ω_0. In particular, it bases its heuristics on the magnitude of the transfer function in the frequency domain,

$$\|H(j\omega)\| = 1/\sqrt{1 + (\omega/\omega_0)^2}\ , \tag{4.6}$$

for some frequency ω. The magnitude function controls the convergence speed of the search process within the domains of the optimized variables. The NLTA algorithm optimizes the control gains by minimizing a tracking error-based objective cost function. In this work, the objective function is chosen to be in the form of an Integrated Squared Error (ISE), defined as

$$ISE = \int_0^t e^2(\tau)\,\mathrm{d}\tau \tag{4.7}$$

with $e(\tau)$ being the tracking error at continuous time τ. Full details of how the NLTA is applied to optimize the gains of the PID controller are provided in Algorithm 4.2.

Algorithm 4.2 NLTA: Offline Tuning of the PID Gains

Input:

$(K_{p_{min}}, K_{p_{max}})$: Search range of K_p.
$(K_{i_{min}}, K_{i_{max}})$: Search range of K_i.
$(K_{d_{min}}, K_{d_{max}})$: Search range of K_d.
$\omega_0 > 0$: a nominal frequency for the low-pass filter.
$\Delta\omega > 0$: a frequency discount value.
$\omega_1 > 0$: an initial value for ω.
N_ITERATIONS: number of iterations per episode.
N_EPISODES: number of episodes.

Output:

Optimized PID gains (K_p, K_i, K_d).

1: **for** $p = 1$ to N_EPISODES **do** ▷ Beginning of an episode
2: Randomly initialize the PID gains K_p, K_i, K_d, to some stable values ▷ Stability may be verified by simulating the system in Figure 4.3
3: Simulate the system in Figure 4.3 and calculate the ISE using (4.7)
4: ISE0 $\leftarrow$ ISE
5: Smallest_ISE_p $\leftarrow$ ISE ▷ Smallest ISE in the episode
6: $\omega \leftarrow \omega_1$
7: **for** $i = 1$ to N_ITERATIONS **do** ▷ Beginning of an iteration within an episode
8: Randomly select one PID gain candidate K_p', K_i' or K_d' from its respective range
9: Simulate the system in Figure 4.3 with the PID gain candidate selected in the previous step (line 8), and calculate the ISE using (4.7)
10: ISEV $\leftarrow$ ISE
11: **if** $\frac{\text{ISEV}}{\text{ISE0}} \leq 1$ **or** $\frac{\text{ISEV}}{\text{ISE0}} \leq \frac{1}{\|H(j\omega)\|}$ **then** ▷ The second condition allows for exploration to avoid local minima
12: Replace the PID gain with its candidate value ▷ The one selected in line 8
13: Smallest_ISE_p $\leftarrow$ ISEV
14: **if** $(\omega - \Delta\omega) > 0$ **then**
15: $\omega \leftarrow \omega - \Delta\omega$ ▷ To control the convergence speed of the search process
16: **end if**
17: **end if**
18: **end for**
19: triplet$(p) \leftarrow (K_p, K_i, K_d)$ ▷ Optimized PID gains for episode p
20: Smallest_ISE$(p) \leftarrow$ Smallest_ISE_p
21: **end for**
22: $q \leftarrow \arg\min_p$ Smallest_ISE(p)
23: **return** triplet(q)

4.4 Neural Network Optimization Algorithm

A nonlinear state feedback control law u^{NN} is developed to optimize the dynamic performance of the control scheme. It is added to the tracking control effort to form the aggregate control signal. Although the Q-Table in its simplest form can be used to control the dynamic system, it results in non-smooth discrete actions, and consequently an undesired scattered dynamic performance [6]. Therefore, the added neural network trained using this Q-Table is able to generate a continuous (non-discrete) control signal which can smooth out the system's behavior.

To this end, two feedforward multi-layer perceptrons are trained offline for comparison purpose. Only one of them is eventually integrated in the closed-loop control system to generate an optimized value of u^{NN}, as shall be detailed later. The training data for each neural network is prepared using a separate Q-Table with a similar structure to the one employed in the RL-based tracking controller, except that this time the full state vector is considered, instead of the tracking error combination.

The reason behind trying two neural networks is the ability to test and compare the following two objective cost functions:

$$F_1(k) = \frac{1}{2}\left(X_k^T\, S\, X_k + X_{k+1}^T\, S\, X_{k+1}\right) + Z\left(u_k^{NN_1}\right)^2 \tag{4.8a}$$

$$F_2(k) = X_{k+1}^T\, S\, X_{k+1} + Z\left(u_k^{NN_2}\right)^2 \tag{4.8b}$$

where $S \in \mathbb{R}^{n\times n}$ and $Z \in \mathbb{R}$ are weighting factors reflecting the designer's preference of how the state and the control effort are prioritized. Notice how the second performance index F_2 gives no importance to the current state, unlike F_1. Instead, it is more driven by the future states. In the following, we will denote the neural networks trained using objective functions F_1 and F_2 by NN_1 and NN_2, respectively; while their outputs are denoted by u^{NN_1} and u^{NN_2}.

The training samples are arranged in two steps. The first associates the states to all possible control actions; while the second applies an optimization criteria to decide on the control action to be applied at any given state [6]. The complete training process of the

neural networks is presented in Algorithm 4.3.

It is worth to mention that, unlike the RL algorithm, the discretization steps ΔX_j, $j = 1, \ldots, n$, and Δu^{NN} for the measured states and the control signals, respectively, can be refined as needed without running into massive calculation overhead [6]. This is because the optimization process associated with each state X_j follows a different optimization and approximation procedure. It is accomplished in one episode, unlike the RL approach which employs successive search episodes, as explained earlier.

Algorithm 4.3 Neural Network: Offline Energy Optimization Scheme

Input:

Minimum and maximum bounds $X_{j_{min}}$, $X_{j_{max}}$ of every state $X_j \in X$, $j = 1, 2, \ldots, n$, and its discretization step ΔX_j.

Minimum and maximum bounds u^{NN}_{min} and u^{NN}_{max} of u^{NN}, and its discretization step Δu^{NN}.

Output:

Two trained neural networks, NN_1 and NN_2

1: Discretize the action space $[u^{NN}_{min}, u^{NN}_{max}]$ into N_u discrete actions: $[u^{NN}_1, u^{NN}_2, \ldots, u^{NN}_{N_u}]$.

2: Discretize the state space into a single row of N_X entries: $[X_{1_{min}}, X_{1_{min}} + \Delta X_1, X_{1_{min}} + 2\Delta X_1, \ldots, X_{1_{max}}, X_{2_{min}}, X_{2_{min}} + \Delta X_2, X_{2_{min}} + 2\Delta X_2, \ldots, X_{2_{max}}, \ldots, X_{n_{min}}, X_{n_{min}} + \Delta X_n, X_{n_{min}} + 2\Delta X_n, \ldots, X_{n_{max}}]$.

3: Create Q-Tables $Q_1(1 \ldots N_u, 1 \ldots N_X)$ and $Q_2(1 \ldots N_u, 1 \ldots N_X)$, whose rows and columns correspond to the possible discrete actions and states formed in lines 1 and 2, respectively.

4: Populate Q_1 and Q_2 using cost functions (4.8a) and (4.8b), respectively.

5: Create and initialize two feedforward multi-layer perceptrons, NN_1 and NN_2, with n inputs (corresponding to samples of states $X_1, X_2, \ldots, X_n$) and one output (corresponding to the action u).

6: Train NN_1 and NN_2 using the data in Q_1 and Q_2, respectively.

7: **return** NN_1 and NN_2

4.5 Aggregate Control Scheme

After the offline training of the RL- and the NLTA-based trackers along with the NN optimizer, they are integrated in two feedback systems of the same architecture, as shown in Figure 4.5. The first system uses the RL-based tracker u^{RL} while the second system employs the NLTA-based tracker u^{NLTA}. One of the goals of this work is to compare the performances of both systems. The aggregate control signal u_k^T applied to the dynamic system under consideration is either $u_k^T = u_k^{RL} + u_k^{NN}$ or $u_k^T = u_k^{NLTA} + u_k^{NN}$, depending on the type of tracking controller used. In an abuse of notation, the following notation is adopted in the rest of the paper: $u_k^T = u_k^{RL/NLTA} + u_k^{NN}$. Note that u_k^{NN} is either $u_k^{NN_1}$ or $u_k^{NN_2}$, depending on the NN optimizer used.

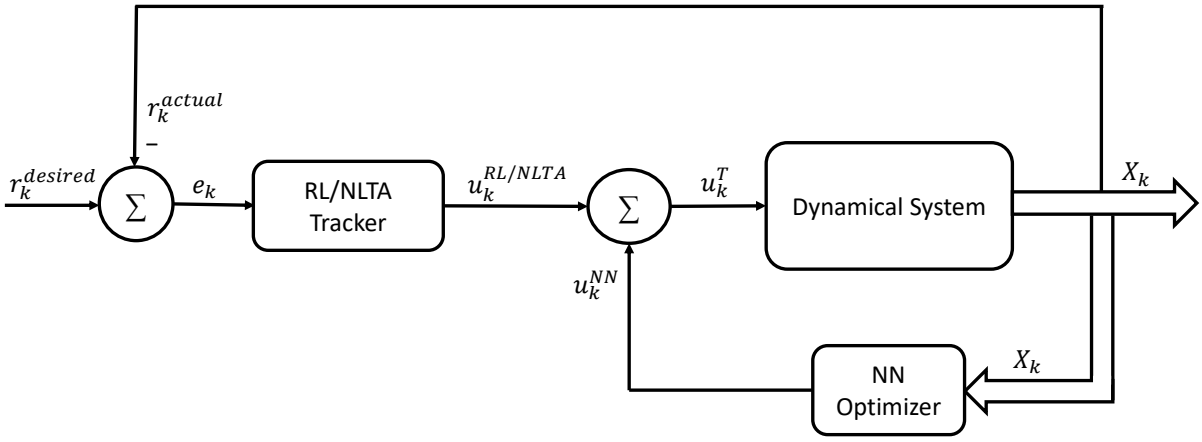


Figure 4.5: Combined tracking control scheme.

The two systems operate in discrete time. To adapt the NLTA-based tracker to this architecture, it needed to be discretized, since it is a continuous-time controller in nature. In other words, we needed to discretize the continuous-time PID controller whose gains were optimized using the NLTA algorithm.

The transfer function of a discrete-time PID controller in the z-domain is

$$\frac{u^{NLTA}(z)}{e(z)} = K_p + K_i \cdot T_s \cdot \frac{z}{z-1} + \frac{K_d}{T_s} \cdot \frac{z-1}{z},$$

with T_s being the sampling period. This yields

$$u^{NLTA}(z)\,(1-z^{-1}) = K_p\,(1-z^{-1})\,e(z) + K_i \cdot T_s \cdot e(z) + \frac{K_d}{T_s}\,(1-2z^{-1}+z^{-2})\,e(z).$$

Converting to a difference equation leads to the following discrete-time control expression:

$$u_k^{NLTA} = u_{k-1}^{NLTA} + K_p\,(e_k - e_{k-1}) + K_i \cdot T_s \cdot e_k + \frac{K_d}{T_s}\,(e_k - 2e_{k-1} + e_{k-2}).$$

4.6 Simulation Results and Analysis

4.6.1 Simulation Setup

The proposed control architecture is applied to control the navigation of an autonomous flexible wing aircraft. This type of aircraft is described as a two-body system which is composed of a wing and a pilot/fuselage connected by a hang strap [26]. In a manned system, the pilot controls the aircraft by pushing, pulling, and rolling the control-bar which adjusts the pilot's center relative to that of the wing [21]. The unmanned control process of flexible wing aircraft possesses many challenges. This is mainly because the system is extremely difficult to model due to its continuously varying aerodynamics [21, 25, 26].

The motion of an unmanned flexible wing aircraft can be decoupled into longitudinal and lateral motion frames. Herein, only the lateral motion control is tackled to validate the performance of the proposed tracking schemes. A vector of measurable states $X = [v \quad \dot{\phi} \quad \dot{\varphi} \quad \phi \quad \varphi]^T$ is considered, where v, ϕ and φ are the lateral velocity, roll attitude, and yaw attitude, respectively.

The aircraft is required to follow a desired rolling maneuver (i.e., $r_k^{desired}$) and to undergo continuous opposite banking turns as follows:

$$r_k^{desired} = \phi_k^{desired} = \frac{15\pi}{180}\,\sin(4\,\pi\,k/T)\ \text{rad}, \tag{4.9}$$

with the following initial conditions: $X_0 = \begin{bmatrix} 10\,\text{m/s} & 0.5\,\text{rad/s} & 0.5\,\text{rad/s} & 0 & 0 \end{bmatrix}^T$, $u_0^{RL} =$

$u_0^{NLTA} = u_0^{NN} = 0$, where T is the total number of simulation iterations. Note that $T = T_d/T_s$, with $T_d = 200\,\text{s}$ and $T_s = 0.01\,\text{s}$ being the total duration of the simulation and the sampling time, respectively.

The learning parameters of the RL-based tracker are listed in Table 4.1. The reward function R is determined by a scalar convex cost criteria $\delta_k = e_k^2 + 0.9\,e_{k-1}^2 + 0.8\,e_{k-2}^2 + 0.7\,e_{k-3}^2$, such that

$$R = \begin{cases} 100\left(1 + e^{50\,(\delta_k - \delta_{k+1})}\right) & \text{, if } \delta_{k+1} < \delta_k \\ 10^5 & \text{, if } \delta_{k+1} = \delta_k = 0\,\cdot \\ -10^5 & \text{, otherwise} \end{cases}$$

This function assigns a positive reward if the dynamic cost δ_{k+1} is less than δ_k, where the highest reward is achieved at equilibrium.

Table 4.1: Parameters of the RL-based Tracker

Parameters	**Values**
X_{min}, X_{max}	$\pm[20\,\text{m/s} \quad 2\,\text{rad/s} \quad 2\,\text{rad/s} \quad 0.44\,\text{rad} \quad 0.44\,\text{rad}]^T$
u_{min}^{RL}, u_{max}^{RL}	$\pm 0.44\,\text{rad}$
E_{min}, E_{max}	$\pm 0.44\,\text{rad}$
ΔX	$[4\,\text{m/s} \quad 0.4\,\text{rad/s} \quad 0.4\,\text{rad/s} \quad 0.055\,\text{rad} \quad 0.055\,\text{rad}]^T$
ΔE	$0.035\,\text{rad}$
Δu^{RL}	$0.055\,\text{rad}$
α	0.5
γ	0.9
ϵ	0.9

The NLTA algorithm parameters and the optimized PID gains obtained after applying the NLTA algorithm are listed in Tables 4.2 and 4.3. In this case, the search range of the PID gains is selected around an initial value ($Kp = 9.9345, Ki = 4.6778, Kd = 3.5585$) determined by Matlab's PID Tuner so as to minimize the settling time. This tool can search for control gains that compromise between having larger closed-loop bandwidth (i.e., faster response) and having enough gain and phase margins to tackle the robustness of the tuning outcome.

Table 4.2: Parameters of the NLTA-based Tracker

Parameters	**Values**
$(K_{p_{min}}, K_{p_{max}})$	(0,30)
$(K_{i_{min}}, K_{i_{max}})$	(0,15)
$(K_{d_{min}}, K_{d_{max}})$	(0,12)
ω_0 [rad/s]	2×10^3
$\Delta\omega$ [rad/s]	0.0002
ω_1 [rad/s]	2.2×10^3
N_EPISODES	10
N_ITERATIONS	10^5

Table 4.3: Optimized PID Control Gains

$\boldsymbol{K_p}$	$\boldsymbol{K_i}$	$\boldsymbol{K_d}$
29.7770	12.7673	10.6736

The structure and training parameters of the neural network optimizers NN_1 and NN_2 are detailed in Table 4.4.

The weighting matrices S and Z are selected as to weigh the importance of the different dynamic signals in the cost functions (4.8a) and (4.8b). They are set to

$$S = \begin{bmatrix} 0.0025 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 5.1653 & 0 \\ 0 & 0 & 0 & 0 & 5.1653 \end{bmatrix}, \quad Z = 0.0517$$

4.6.2 Performance Analysis

Three simulation scenarios are considered. The first adopts the dynamic model of the aircraft at nominal trim speed and flight condition. In the second scenario, a sudden variation in the dynamic model is applied at time $t_s = 20\,\mathrm{s}$ (corresponding to $k_s = t_s/T_s = 2,000$). It aims at assessing the controller's performance in the face of a sudden drop in the aircraft's payload, for example. Finally, a more aggressive simulation scenario is

Table 4.4: Parameters of the Neural Network Optimizer

Parameters	**Values**
X_{min}, X_{max}	$\pm[20\,\text{m/s} \quad 2\,\text{rad/s} \quad 2\,\text{rad/s} \quad 0.44\,\text{rad} \quad 0.44\,\text{rad}]^T$
u^{NN}_{min}, u^{NN}_{max}	$\pm 0.44\,\text{rad}$
ΔX	$(X_{max} - X_{min})/20$
Δu^{NN}	$(u^{NN}_{max} - u^{NN}_{min})/20$
Number of hidden layers	1
Number of hidden neurons	13
Size of data set	$4,084,000$ samples
Training data set	70%
Validation data set	15%
Testing data set	15%
Learning algorithm	Levenberg-Marquardt

considered, where the dynamics of the aircraft are allowed to vary around their nominal values (A^d_1 and B^d_1) at each evolution step k. The variation of each state (shown in Fig. 4.6) is drawn from a normal distribution of zero mean and a variance of 0.5. The dynamics of the three scenarios are summarized in Table 4.5, where for every element (i, j) of matrices $\widehat{A^d_k}$ and $\widehat{B^d_k}$, is defined as

$$\widehat{A^d_k}(i,j) = (1+\sigma_A)\,\widehat{A^d_{1_k}}(i,j)$$
$$\widehat{B^d_k}(i,j) = (1+\sigma_B)\,\widehat{B^d_{1_k}}(i,j)$$

where $\sigma_A, \sigma_B \sim \mathcal{N}(0, 0.5)$.

Table 4.5: Simulation Scenarios

Model 1	**Model 2**	**Model 3**
$X_{k+1} = A^d_1 X_k + B^d_1 u_k$	$X_{k+1} = A^d_1 X_k + B^d_1 u_k,\ k \le k_s$ $X_{k+1} = A^d_2 X_k + B^d_2 u_k,\ k > k_s$	$X_{k+1} = \widehat{A^d_k} X_k + \widehat{B^d_k} u_k$

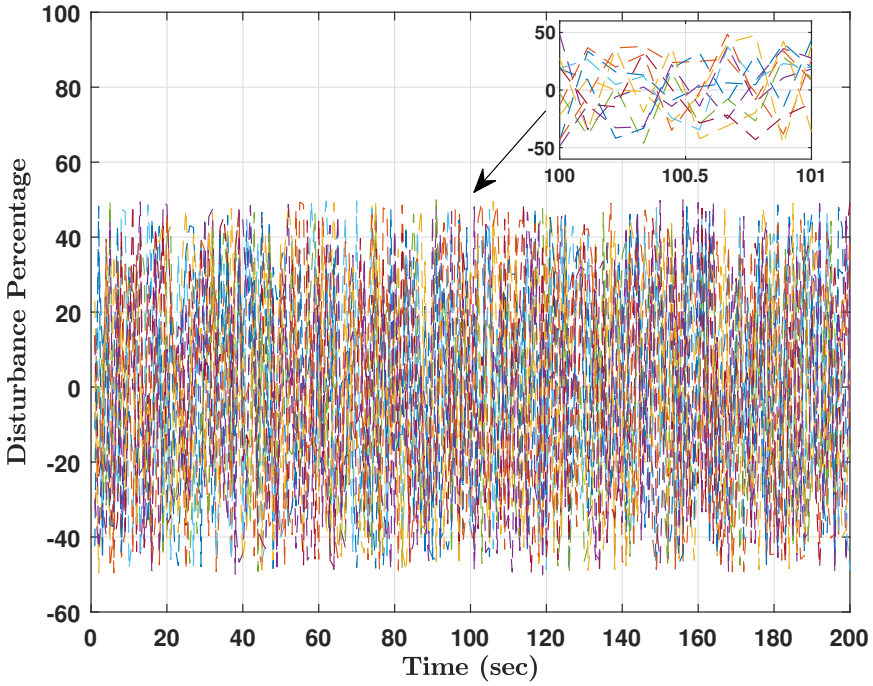


Figure 4.6: Variations in the dynamics.

The discrete-time state space matrices are adopted from [21] as

$$A_1^d = \begin{bmatrix} 0.9977 & -0.0028 & -0.1069 & 0.0971 & -0.0131 \\ -0.0131 & 0.8092 & 0.0677 & 0 & 0 \\ 0.0026 & 0.0333 & 0.9802 & 0 & 0 \\ 0 & 0.0090 & 0 & 1 & 0 \\ 0 & 0 & 0.0099 & 0 & 1 \end{bmatrix}, \quad B_1^d = \begin{bmatrix} 0 \\ 0.0324 \\ -0.0036 \\ 0 \\ 0 \end{bmatrix}$$

$$A_2^d = \begin{bmatrix} 0.9987 & -0.0028 & -0.0940 & 0.1230 & -0.0170 \\ -0.0090 & 0.8112 & 0.0637 & -0.0006 & 0.0001 \\ 0.0031 & 0.0402 & 0.9752 & 0.0002 & 0 \\ 0 & 0.0090 & 0.0003 & 1 & 0 \\ 0 & 0.0002 & 0.0099 & 0 & 1 \end{bmatrix}, \quad B_2^d = \begin{bmatrix} 0.0002 \\ 0.0251 \\ -0.0044 \\ 0.0001 \\ 0 \end{bmatrix}$$

At first, both neural network optimizers are compared based on their accumulated cost $\sum_{k=1}^{T} F_i(k)$, $i \in \{1, 2\}$, to decide on the one to adopt in the optimization loop of the

aggregate control system (Figure 4.5). The results are illustrated in Figure 4.7. They show that NN_2 yields a lower cumulative dynamic cost compared to that of NN_1. As such, NN_2 is chosen as the control optimizer in the subsequent simulations. In the following, we will refer to the RL- and NLTA-based trackers with and without the neural network optimizer as RL, RL+NN, NLTA, and NLTA+NN.

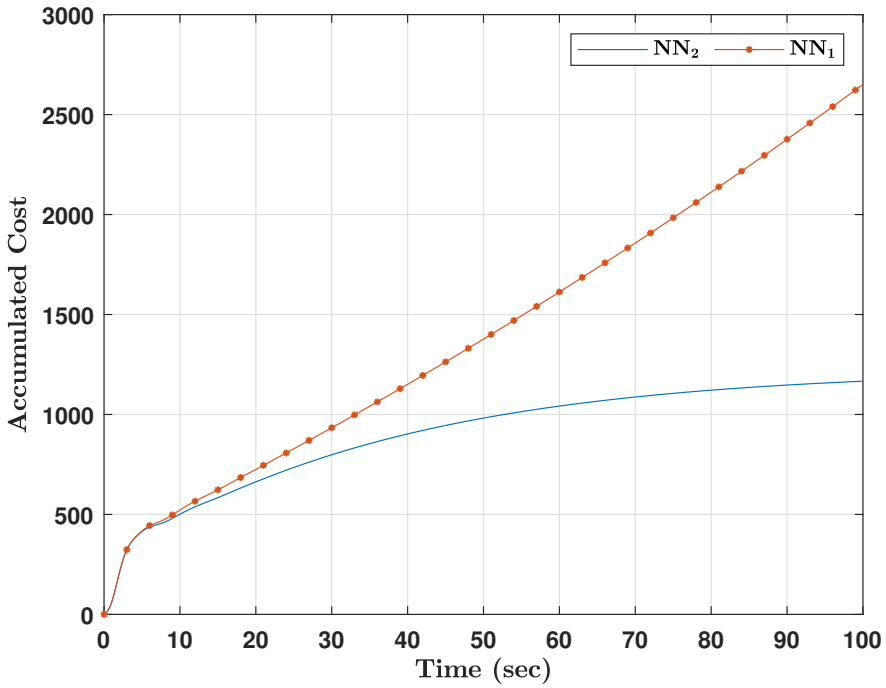


Figure 4.7: Accumulated cost using the objective functions defined in (4.8a) and (4.8b).

The aircraft banking simulation results of the first scenario are shown in Figures 4.8 to 4.10. The RL- and NLTA-based trackers with and without the neural network optimizer are able to asymptotically stabilize the aircraft around the desired banking trajectory $\phi_k^{desired}$. The tracking errors and the control signals of RL and RL+NN are characterized by a chattering behavior within the envelopes $[-0.005\,\text{rad}, 0]$ and $[-0.45\,\text{rad}, 0.45\,\text{rad}]$, respectively. Nevertheless, the tracking errors generated NLTA and NLTA+NN are asymptotically convergent. It also worth noticing that the control signals of the latter controllers are significantly smoother than their counterparts dispatched by RL and RL+NN. Figure 4.9 clearly demonstrates that the neural network optimizer contributes to reducing the cumulative tracking error, however small this contribution might be. The total energy cost

$\sum_{k=1}^{T} X_{k+1}^{T} S X_{k+1} + Z \left(u_k^T\right)^2$, of the four controllers across the three scenarios are summarized in Table 4.6. From this measure, we can deduce that the controllers share similar performances due to the insignificant differences in their total costs. It is interesting to notice that, the NLTA+NN variant of the controller achieved the lowest total energy cost.

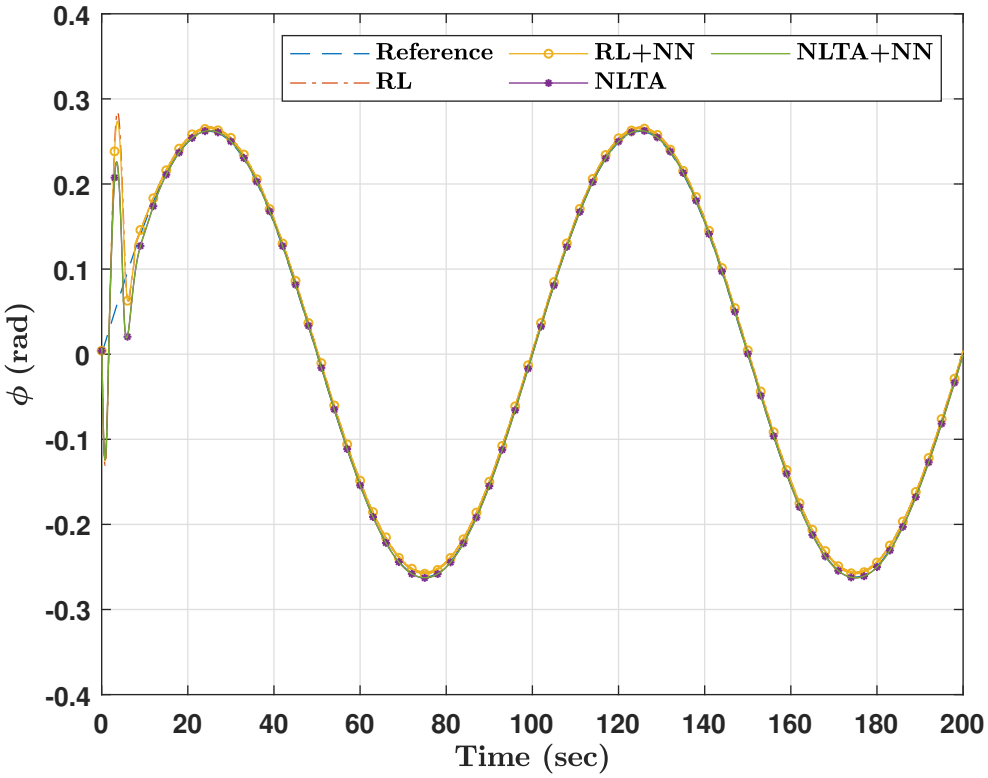


Figure 4.8: Roll trajectory tracking of Model 1.

Table 4.6: Total Energy Cost for the Three Scenarios

Method	Model 1	Model 2	Model 3
RL	1.5997×10^5	1.7493×10^5	1.6159×10^5
RL+NN	1.5978×10^5	1.7477×10^5	1.6086×10^5
NLTA	1.5832×10^5	1.7359×10^5	1.6005×10^5
NLTA+NN	$\mathbf{1.5825 \times 10^5}$	$\mathbf{1.7352 \times 10^5}$	$\mathbf{1.5998 \times 10^5}$

The same remarks are confirmed by the simulation of the second scenario. The results are depicted in Figures 4.11 to 4.13. The proposed control structures initially took a few seconds to converge to the reference signal from the initial condition. However, once on track, it is interesting to notice that they were virtually insensitive to the abrupt change in the system dynamics at time $t = 20\,\text{s}$.

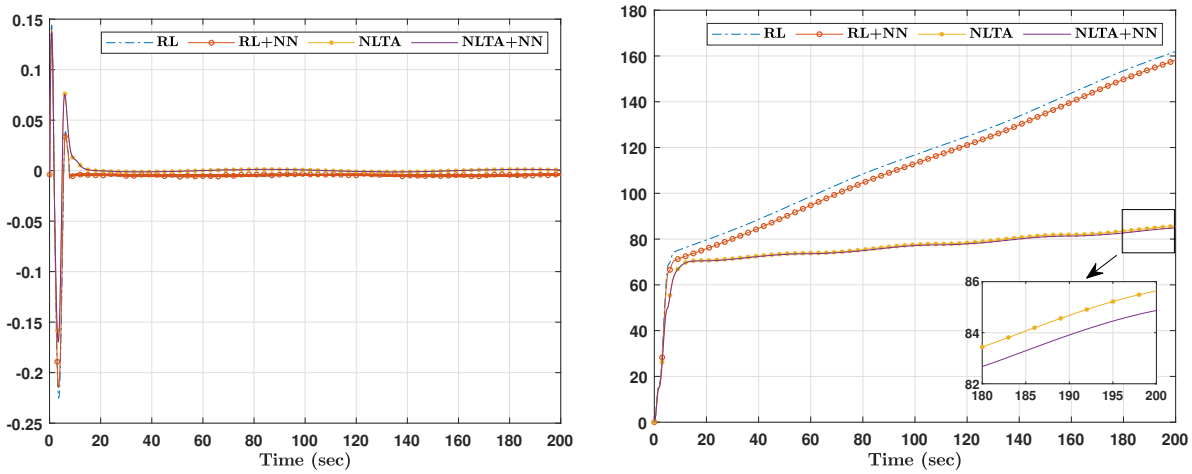
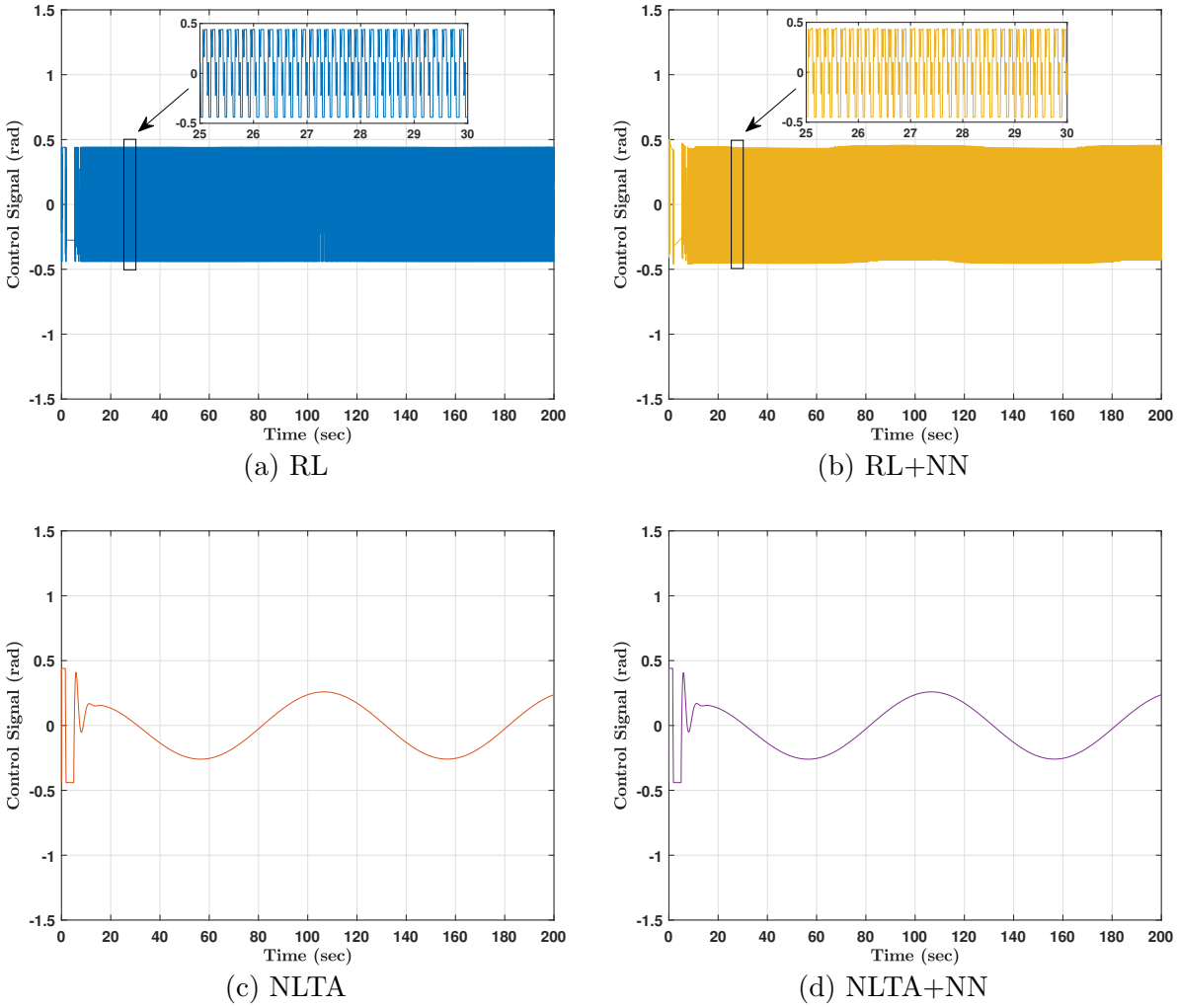


Figure 4.9: (a) Tracking error (rad), and (b) Cumulative tracking error (rad) for Model 1.

The third scenario emulates an aggressive test case, where the dynamics are time-variant this time with an excessive disturbance term. The results are revealed in Figures 4.14 to 4.17. The NLTA and NLTA+NN controllers effectively guide the aircraft towards the desired trajectory with less fluctuations than RL and RL+NN. This is clearly illustrated in Figure 4.17c. This scenario articulates the advantage of the neural network optimizer. It is in such a case that the difference between the trackers with and without the neural network optimizer is most significant.

4.7 Conclusion

The chapter presents a synergetic integration approach of machine learning tools to provide a solution for a class of adaptive tracking control problems. The proposed family of controllers solve the underlying problem using two control loops: one for tracking and one to optimize an overall performance measure. The controllers are tested on a navigation mechanism of a flexible wing aircraft with uncertain/varying dynamics. After an offline training process, they demonstrated a high ability to simultaneously minimize the tracking error and the total energy cost in different test cases of varying complexities. They also succeeded to maintain the system's stability in the face of excessive disturbances. The neural network optimizer proved to well complement the trackers to better annihilate the cumu-



(a) RL (b) RL+NN

(c) NLTA (d) NLTA+NN

Figure 4.10: Control signals for Model 1.

lative error as well as further reduce the total energy cost. Although the four controllers showed a comparable performance in terms of cumulative cost, NLTA and NLTA+NN tend to generate smoother control signals and asymptotically convergent tracking errors. It is worth pointing out that the proposed control structures do not depend on the mathematical model of the system dynamics. The plant can be treated as a black box. Nevertheless, the search space of the control gains has to be a subset of a known region of stability.

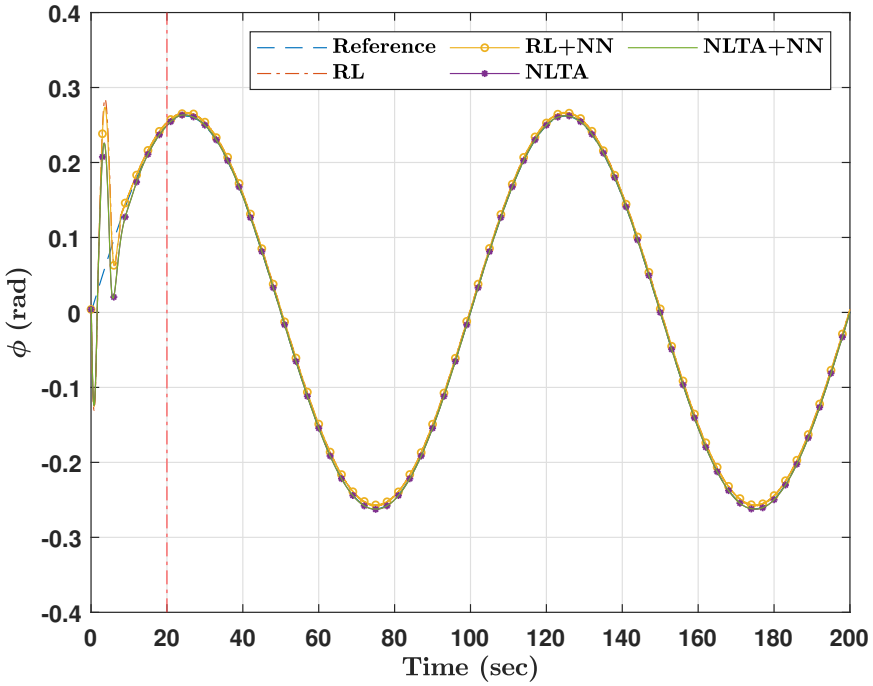
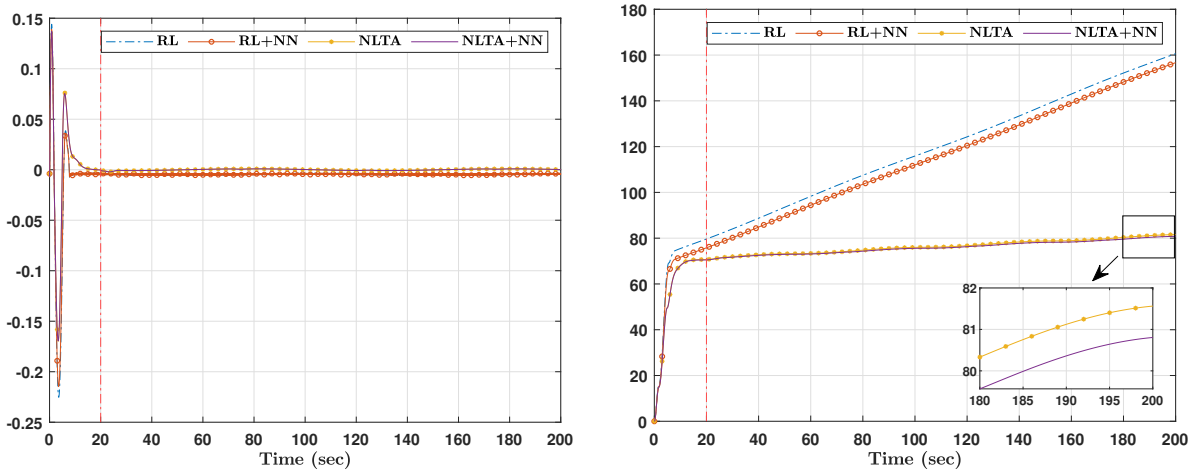
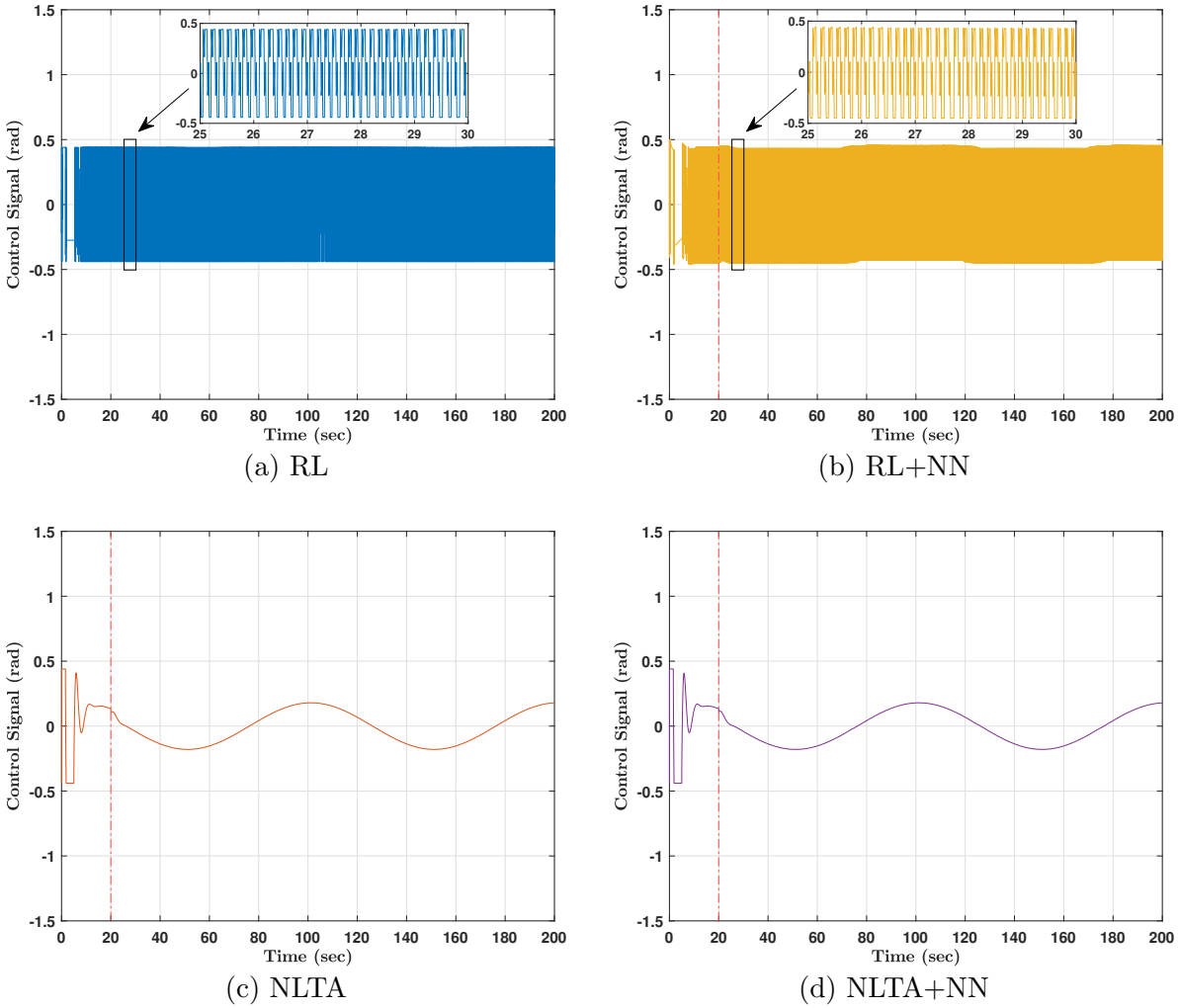


Figure 4.11: Roll trajectory tracking of Model 2.



(a) (b)

Figure 4.12: (a) Tracking error (rad), and (b) Cumulative tracking error (rad) for Model 2.



(a) RL (b) RL+NN

(c) NLTA (d) NLTA+NN

Figure 4.13: Control signals for Model 2.

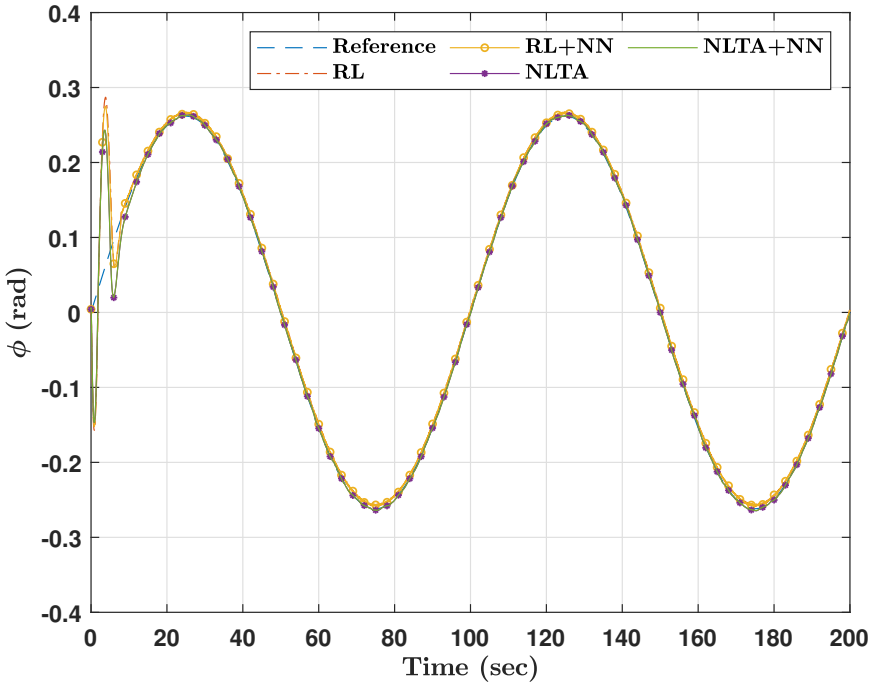
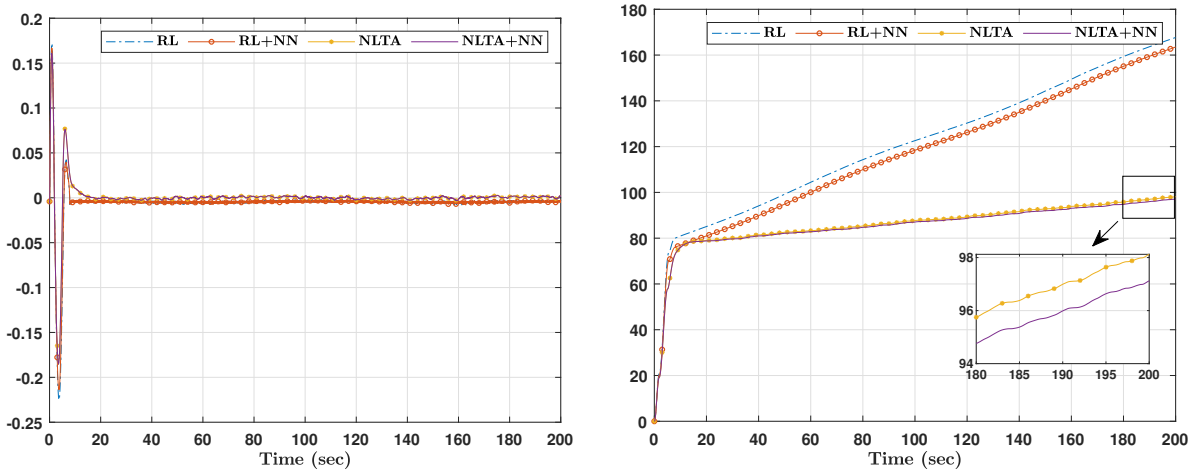
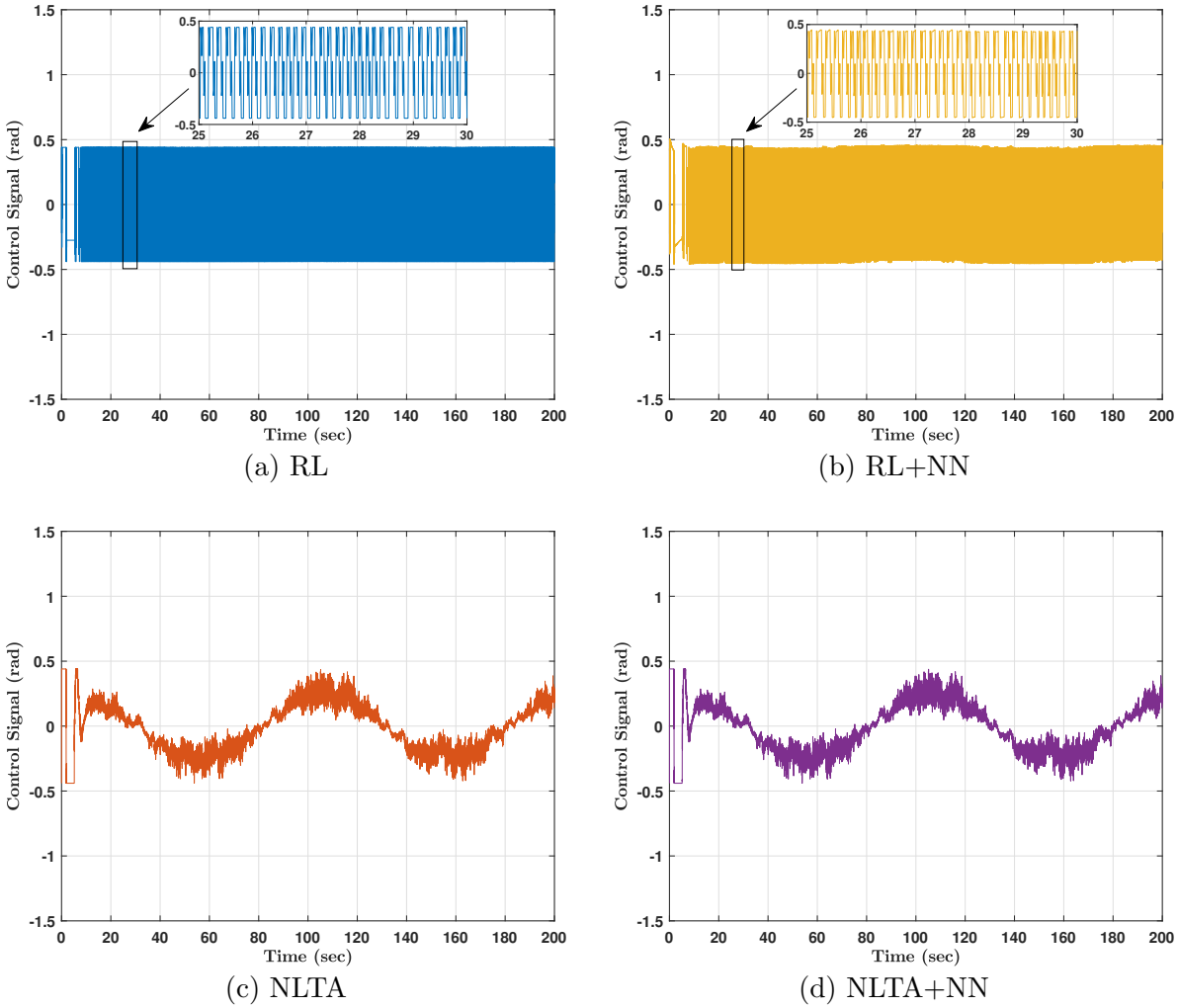


Figure 4.14: Roll trajectory tracking of Model 3.



(a) (b)

Figure 4.15: (a) Tracking error (rad), and (b) Cumulative tracking error (rad) for Model 3.



(a) RL (b) RL+NN

(c) NLTA (d) NLTA+NN

Figure 4.16: Control signals for Model 3.

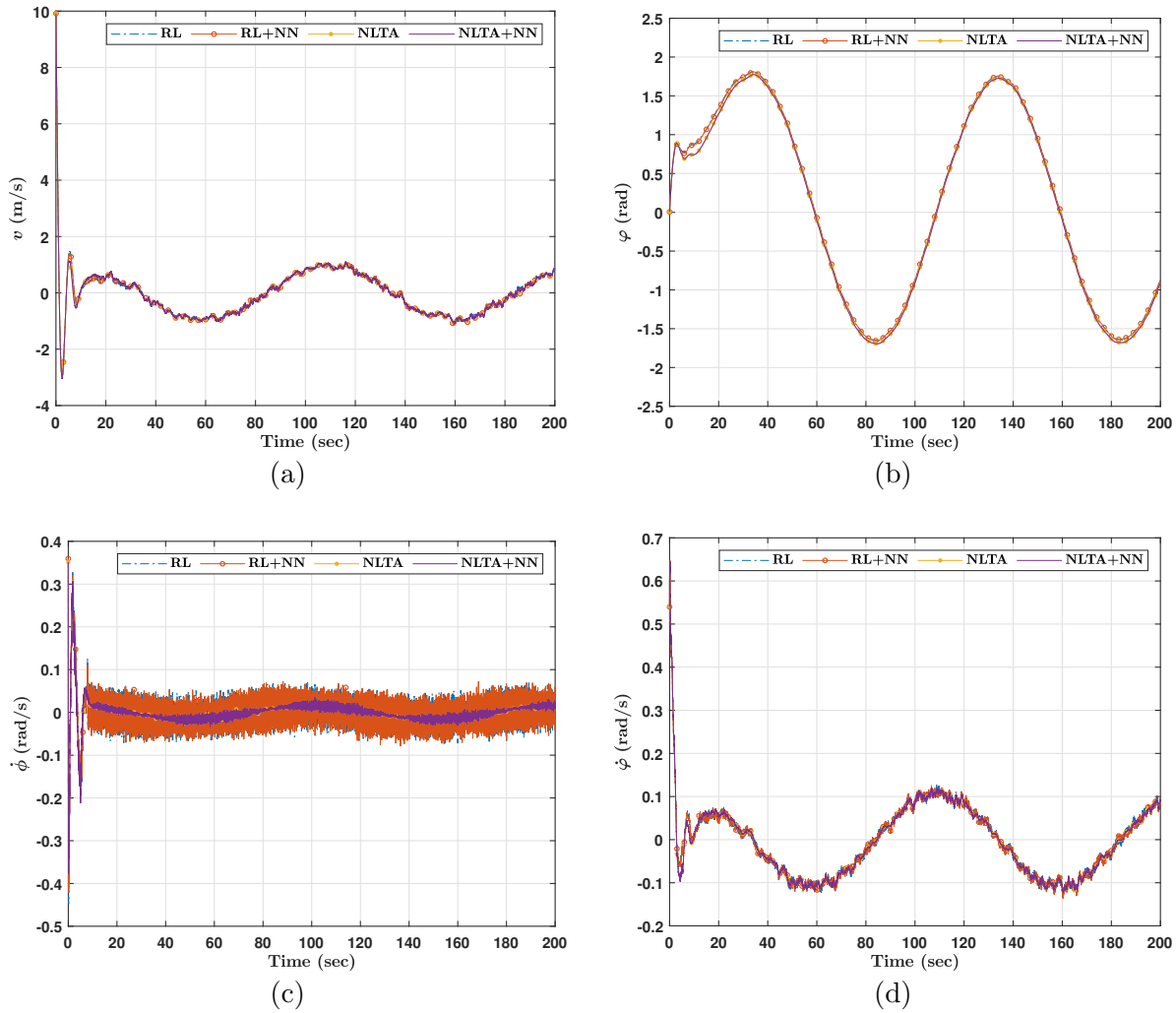


Figure 4.17: System states (ϕ is ommited since it is plotted in Figure 4.14).

Chapter 5

Optimized Tracking Control Structures: A Balance System Case Study

Building on the encouraging performance of the NLTA-based control architecture in the previous chapter, the architecture is generalized in this chapter by extending it to a multi-output system. The goal is to show that this type of architecture can also be employed in systems where there are multiple signals to track. To do so, a balance system is chosen as an application testbed. Balance systems are usually systems in which the center of mass is balanced above a pivot point. This simple concept is at the heart of several interesting applications. For instance, the Segway Personal Transporter uses a motorized platform to stabilize a person standing on top of it. When the rider leans forward, the transportation device propels itself along the ground while maintaining its upright position. Another example is a rocket, in which a gimballed nozzle at the bottom of the rocket is used to stabilize the body of the rocket above it.

5.1 Balance System

Balance systems are typically abstracted using an inverted-pendulum-cart system. A moving free body diagram of such as system is illustrated in Fig. 5.1 [65]. The system involves an inverted pendulum mounted on the center top of a wheeled cart. The pendulum swings left and right as the cart slides back and forth along the x-direction due to a force u horizontally applied on the cart. The symbols M and m refer to the mass of the cart and the point mass rigidly connected at top of the pendulum respectively. l is the length of the inverted pendulum. x denotes the displacement of the cart relative to its initial position. θ is the directed angle of the inverted pendulum spanned from the vertical or upright reference.

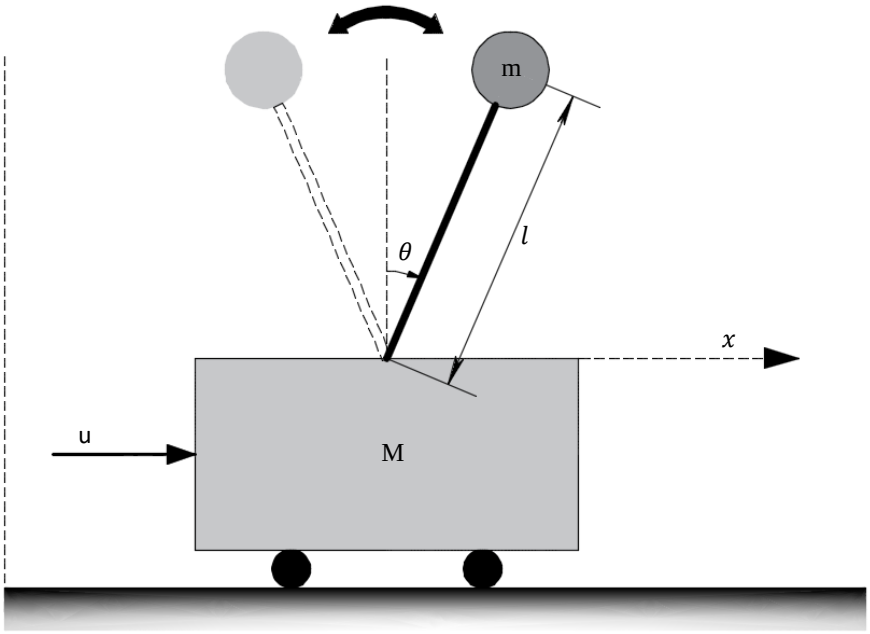


Figure 5.1: Inverted-pendulum-cart free body diagram.

Neglecting friction, the dynamical equations of the system are arranged as follows [65]:

$$(M+m)\,\ddot{x} \;-\; m\,l\,\sin\theta\,(\dot{\theta})^2 \;+\; m\,l\,\cos\theta\,\ddot{\theta} \;=\; u \tag{5.1}$$

$$m\,\ddot{x}\,\cos\theta + m\,l\,\ddot{\theta} = m\,g\,\sin\theta \tag{5.2}$$

where g is the gravitational acceleration. A linearized state space of the system around

the angle of equilibrium $\theta = 0$ can be formulated as

$$\dot{X} = A\,X \,+\, B\,u, \tag{5.3}$$

where

$$X = \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} \qquad A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)g}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-mg}{M} & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ \frac{-1}{Ml} \\ 0 \\ \frac{1}{M} \end{bmatrix}$$

The control objective is to force the cart and the pendulum to follow their reference trajectories, x^{ref} and θ^{ref}, respectively, while minimizing a predefined objective function.

5.2 Position-Angle Control Mechanism

5.2.1 PID Control System

We will now outline the proposed NLTA-based approach for the position-angle control mechanism of the balance system. There are two tracking errors of interest in this case: $e_x = x^{ref} - x$ and $e_\theta = \theta^{ref} - \theta$. The goal is to stabilize both errors at zero. As such, each error signal is controlled with a PID controller whose gains are optimized offline using the NLTA algorithm.

The two PID control loops are shown in Fig. 5.2. The underlying control signals u^x_{PID} and u^θ_{PID}, generated by the PID control loops, are given by

$$u^x_{PID}(t) = K^x_p\, e_x(t) + K^x_i \int_0^t e_x(\tau)\, d\tau + K^x_d\, \frac{de_x(t)}{dt} \tag{5.4}$$

$$u^\theta_{PID}(t) = K^\theta_p\, e_\theta(t) + K^\theta_i \int_0^t e_\theta(\tau)\, d\tau + K^\theta_d\, \frac{de_\theta(t)}{dt} \tag{5.5}$$

where K_p, K_i, K_d are the proportional, integral and derivative gains of each PID controller,

respectively. The aggregate input force applied on the cart is then

$$u(t) = u_{PID}(t) = u^x_{PID}(t) + u^\theta_{PID}(t)$$

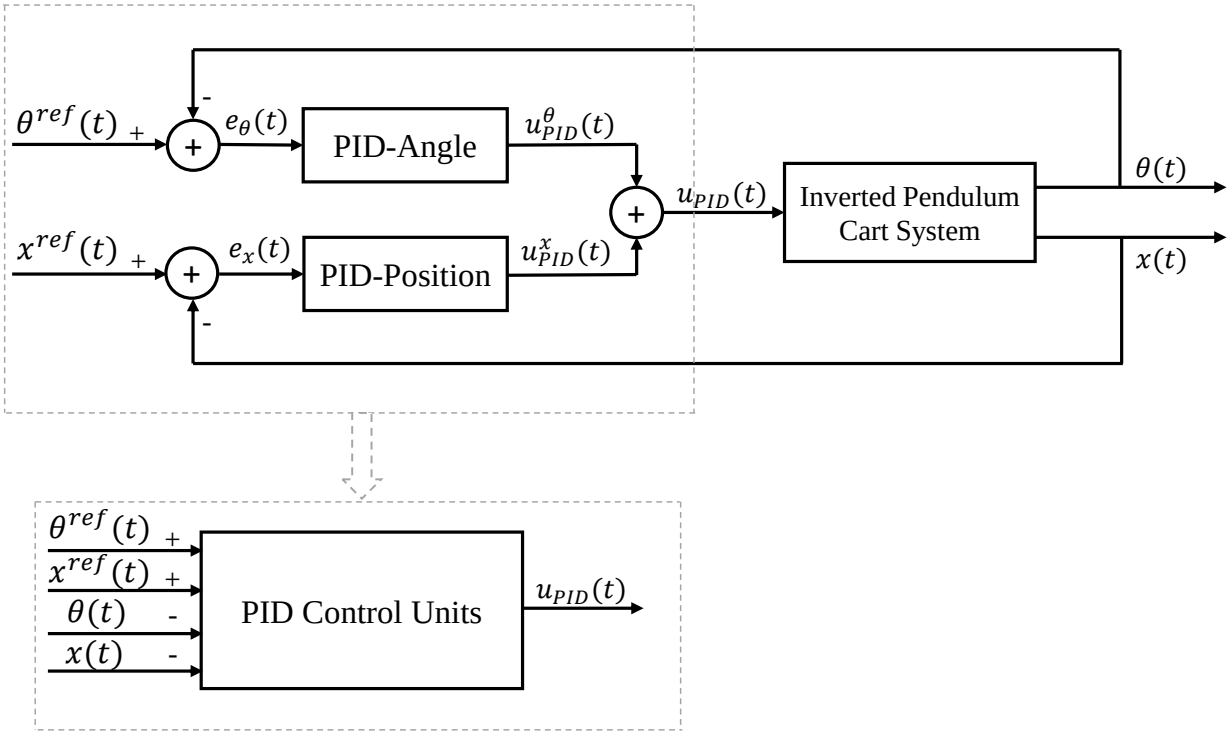


Figure 5.2: Inverted pendulum-angle and cart-position PID-control loops.

5.2.2 NLTA-Based PID Gain Tuning

The dynamics of the pendulum and the cart are coupled. As is evident from Equations (5.1) and (5.2), the cart acceleration affects that of the pendumum and vice versa. Hence, the PID control loops are also coupled. That is, the gains of the PID controller of the cart position influences the pendulum position and vice versa. As with NLTA-based controller of Chapter 5, the gains of both PID controllers are tuned using the NLTA heuritic. Due to the coupling effect of the two PID control loops, the gains of the two PID controllers cannot be independently tuned by separately executing Algorithm 2.4 on each of them. Instead, both controllers are treated by the NLTA Algorithm as a single controller with six parameters to tune. In other words, the algorithm in this case will search for an optimized

tuple $(K_p^x, K_i^x, K_d^x, K_p^\theta, K_i^\theta, K_d^\theta)$ in a 6-dimensional search space. A Generalized NLTA heuristic for the tuning of two sets of PID gains is given in Algorithm 5.1.

Algorithm 5.1 Generalized NLTA: Offline Tuning of two sets of PID Gains

Input:

$(K^{\theta}_{p_{min}}, K^{\theta}_{p_{max}})$: Search range of K^{θ}_{p}.
$(K^{\theta}_{i_{min}}, K^{\theta}_{i_{max}})$: Search range of K^{θ}_{i}.
$(K^{\theta}_{d_{min}}, K^{\theta}_{d_{max}})$: Search range of K^{θ}_{d}.
$(K^{x}_{p_{min}}, K^{x}_{p_{max}})$: Search range of K^{x}_{p}.
$(K^{x}_{i_{min}}, K^{x}_{i_{max}})$: Search range of K^{x}_{i}.
$(K^{x}_{d_{min}}, K^{x}_{d_{max}})$: Search range of K^{x}_{d}.
$\omega_0 > 0$: a nominal frequency for the low-pass filter.
$\Delta\omega > 0$: a frequency discount value.
$\omega_1 > 0$: an initial value for ω.
N_ITERATIONS: number of iterations per episode.
N_EPISODES: number of episodes.

Output:

Optimized PID gains $(K^{\theta}_{p}, K^{\theta}_{i}, K^{\theta}_{d})$ and $(K^{x}_{p}, K^{x}_{i}, K^{x}_{d})$.

1: **for** $p = 1$ to N_EPISODES **do** ▷ Beginning of an episode
2: Randomly initialize all the PID gains K^{θ}_{p}, K^{θ}_{i}, K^{θ}_{d}, K^{x}_{p}, K^{x}_{i}, $K^{x}_{d}$to some stable values ▷ Stability may be verified by simulating the system in Figure 5.2
3: Simulate the system in Figure 5.2 and calculate the ISE using (5.6) ▷ or any of its variants, e.g., (5.8)
4: ISE0 ← ISE
5: Smallest_ISE_p ← ISE ▷ Smallest ISE in the episode
6: $\omega \leftarrow \omega_1$
7: **for** $i = 1$ to N_ITERATIONS **do** ▷ Beginning of an iteration within an episode
8: Randomly select one PID gain candidate $K^{\theta'}_{p}$, $K^{\theta'}_{i}$, $K^{\theta'}_{d}$,$K^{x'}_{p}$, $K^{x'}_{i}$ or $K^{x'}_{d}$ from its respective range
9: Simulate the system in Figure 5.2 with the PID gain candidate selected in the previous step (line 8), and calculate the ISE using (5.6) ▷ or any of its variants
10: ISEV ← ISE
11: **if** $\frac{\text{ISEV}}{\text{ISE0}} \leq 1$ **or** $\frac{\text{ISEV}}{\text{ISE0}} \leq \frac{1}{\|H(j\omega)\|}$ **then** ▷ The second condition allows for exploration to avoid local minima
12: Replace the PID gain with its candidate value ▷ The one selected in line 8
13: Smallest_ISE_p ← ISEV
14: **if** $(\omega - \Delta\omega) > 0$ **then**
15: $\omega \leftarrow \omega - \Delta\omega$ ▷ To control the convergence speed of the search process
16: **end if**
17: **end if**
18: **end for**
19: tuple(p) ← $(K^{\theta}_{p}, K^{\theta}_{i}, K^{\theta}_{d}, K^{x}_{p}, K^{x}_{i}, K^{x}_{d})$ ▷ Optimized PID gains for episode p
20: Smallest_ISE(p) ← Smallest_ISE_p
21: **end for**
22: $q \leftarrow \arg\min_{p}$ Smallest_ISE(p)
23: **return** tuple(q)

Since we have more than one tracking error to stabilize, the NLTA algorithm can no longer use the optimization criterion defined by Equation (4.7). This criterion can be generalized to include multiple tracking errors. As a matter of fact, one can adopt an objective function to influence the system's transient response and steady state characteristics to certain desired regions due their direct relationship with the PID gains. However, some of these objectives may be contradictory. For example, a shorter settling time may lead to a higher overshoot, for example. To that end, in the following, we will suggest a number cost functions to be adopted for the accepting function of the NLTA approach. Each cost function exploits a certain comporomise between some of the system's response characteristics.

The first optimization criterion we consider is a convex cost function that minimizes the Integrated Squared Errors (ISE).

$$\text{ISE} = \int_0^t \big(w_\theta \, e_\theta^2(\tau) + w_x \, e_x^2(\tau)\big) d\tau, \tag{5.6}$$

where w_θ and w_x are some weighting constants. In this case, we took $w_\theta = w_x = 0.5$.

Another objective function is considered to reduce the overshoot along with the ISE. We call it the "ISE and Absolute Error" criterion (ISE-AB). It is defined as

$$\text{ISE-AB} = \int_0^t \big(w_\theta e_\theta^2(\tau) + w_x e_x^2(\tau) + w'_\theta |e_\theta(\tau)| + w'_x |e_x(\tau)|\big) d\tau, \tag{5.7}$$

where w_θ, w_x, w'_θ, and w'_x, are weight constants, which in this work were initialized to $w_\theta = w_x = w'_\theta = w'_x = 0.25$.

The third objective function tackles the cart response settling time T_s along with the ISE. We call it the "ISE and Settling Time" criterion (ISE-TS) and define it by

$$\text{ISE-ST} = \int_0^t \big(w_\theta e_\theta^2(\tau) + w_x e_x^2(\tau)\big) \, d\tau \; + \; w_s \, T_s, \tag{5.8}$$

where w_s is the weight associated to the settling time. Here, the weights are set as $w_\theta = w_x = 0.5$ and $w_s = 0.1$.

The final objective function addresses the ISE and the cart response overshoot OS. It is referred to as the "ISE and OverShoot" criterion (ISE-OS) and is given by

$$\text{ISE-OS} = \int_0^t \left(w_\theta e_\theta^2(\tau) + w_x e_x^2(\tau) \right) d\tau \; + \; w_o \, \text{OS}, \tag{5.9}$$

where w_o is the weight associated to the overshoot. We fixed the weights to $w_\theta = w_x = 0.5$ and $w_o = 0.1$.

5.3 State Feedback Control Mechanism

The control interest is not only to control signals that are directly related tracking errors, such as the settling time and overshoot. In addition, it is desired to enable the control architecture to obtimize a broader objective function that may encompass other signals as well, such as the control effort, for instance. As in Chapter 4, we are implementing that here with a neural network-based state feedback mechanism. It is then compared to another state feedback architecture proposed in [65] which is based on the system's Algebraic Riccati Equation (ARE). Both techniques are presented in the following sections.

5.3.1 Neural Network Optimization Algorithm

A feedforward neural network is trained in order to optimize the overall dynamical performance of the balance system. The aggregate control system including the full state neural network optimization loop and the PID control loops is shown in Fig. 5.3. As can be seen from the figure, the aggregate control signal is $u = u_{PID} + u_{NN}$.

The neural network architecture and training mechanism follow those discussed in Section 4.4 where the objective criterion adopted in this case is similar to (4.8b) so that

$$F_3 = X_{k+1}^T \, Q^{NN} \, X_{k+1} + R^{NN} (u_{NN})^2, \tag{5.10}$$

where $Q^{NN} \geq 0 \in \mathbb{R}^{4\times4}$ and $R^{NN} > 0 \in \mathbb{R}$ are symmetric semi-positive and positive definite

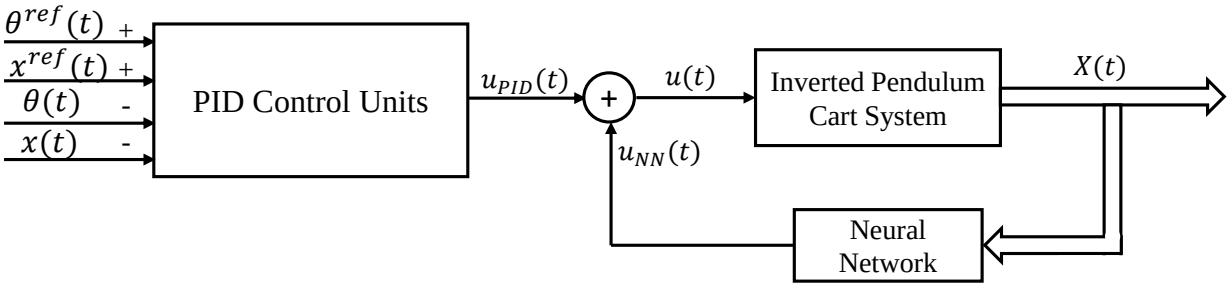


Figure 5.3: Overall PID-Neural Network control scheme.

weighting matrices, respectively. In this particular case, R^{NN} is a scalar since the control signal u_{NN} is a scalar. Note how cost function (5.10) takes into account minimizing the control actions as well as future states. The rest of the details are as listed in Algorithm 4.3.

5.3.2 Linear Quadratic Regulator

The linear quadratic regulator (LQR) [47] provides an optimal control solution to system (5.3) based on minimizing cost function

$$U(X,u) = \frac{1}{2}(X^T\,Q\,X + \,R\,u^2), \tag{5.11}$$

where $Q \geq 0$ and $R > 0$ are semi-positive and positive definite square matrices of proper dimensions. The control performance index is considered to be

$$J = \int_0^\infty U(X,u)\,dt. \tag{5.12}$$

The objective of the optimal controller is to find an optimal state feedback control gain diagonal matrix K such that the optimal control signal u^o is in the form $u^o = -K\,X$, where X is the feedback state vector. The feedback control gain K is given by

$$K = R^{-1}\,B^T\,P$$

where $P \geq 0 \in \mathbb{R}^{n \times n}$ is a square semi-positive definite matrix derived by solving the following ARE

$$A^T P + P A - P B R^{-1} B^T P + Q = 0. \tag{5.13}$$

As such, the closed-loop dynamics can be reformulated as

$$\dot{X} = A^c X,$$

where $A^c = (A - B R^{-1} B^T P)$. The LQR is integrated in the balance system's closed loop in the same manner as the neural network optimizer. The aggregare control architecture is shown in Fig. 5.4.

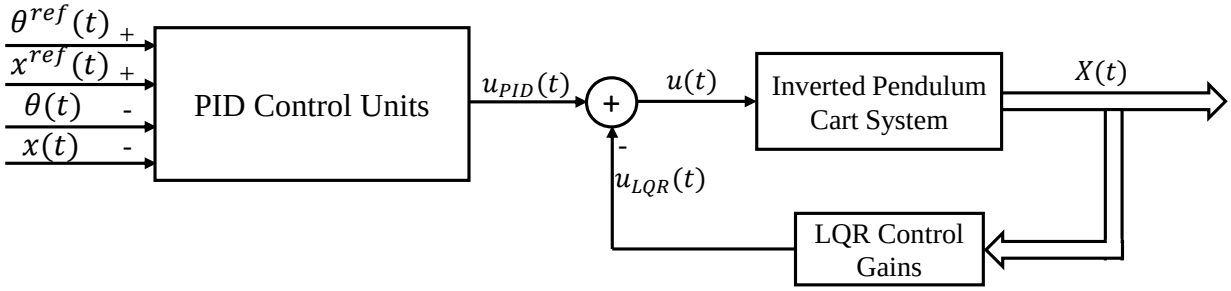


Figure 5.4: Combined PID-LQR control scheme.

5.4 Simulation Results

The proposed control schemes are applied on the balance system. The simulations are conducted using Matlab-Simulink environment. The system's physical parameters are listed in Table 5.1 [65]. The state space matrices are set to

Table 5.1: Parameters of the inverted-pendulum-cart system

Parameters	Values
M	2.4 kg
m	0.23 kg
l	0.36 m
g	9.8 m/s^2

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 29.8615 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.9401 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ -1.1574 \\ 0 \\ 0.4167 \end{bmatrix} \qquad X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The cart and pole reference positions are taken as $x^{ref}(t) = 0.1\,\text{m}$ and $\theta^{ref}(t) = 0$, $\forall t \geq 0$. This is to say that the controller is aimed to maintain the pole at an upright position while driving the cart 0.1 m to the right.

5.4.1 Performance Analysis of the Different PID Schemes

At first, the system is simulated with only the PID loops, as depicted in Figure 5.2. This is to study the performance of the tracker without the state feedback optimization loop. The generalized NLTA algorithm described in Algorithm 5.1 is applied offline to tune the gains of the two PID controllers. The algorithm parameters adopted for the simulations are listed in Table 5.2. In this case, the search range of the PID gains is selected around the values proposed in [65].

Table 5.2: NLTA parameters applied for the optimization of the PID gains

Parameters	Values
$(K^{\theta}_{p_{min}}, K^{\theta}_{p_{max}})$	(-44,-36)
$(K^{\theta}_{i_{min}}, K^{\theta}_{i_{max}})$	(-2,2)
$(K^{\theta}_{d_{min}}, K^{\theta}_{d_{max}})$	(-10,-6)
$(K^{x}_{p_{min}}, K^{x}_{p_{max}})$	(-3,1)
$(K^{x}_{i_{min}}, K^{x}_{i_{max}})$	(-2,2)
$(K^{x}_{d_{min}}, K^{x}_{d_{max}})$	(-5,-1)
ω_0 [rad/s]	200
$\Delta\omega$ [rad/s]	0.005
ω_1 [rad/s]	50
N_EPISODES	10
N_ITERATIONS	1000

The quality of the PID gains and hence the transient and steady state characteristics are affected by the choice of the objective cost function adopted by the NLTA algorithm.

The resultant response characteristics with each of the objective functions defined earlier are summarized in Table 5.3. It is interesting to observe that specialized objective functions are successful in minimizing their target characteristic. For example, the ISE-TS criterion led to the best settling time among the four tested objective functions. Likewise, the ISE-OS criterion led to the best overshoot. This emphasizes the flexibility of the proposed architecture to tailor the cost function according to the desired features of the system response.

Table 5.3: PID tuning outcomes with different cost functions

Cost function	Properties	Minimum values	Maximum values
ISE	Rise Time [s]	0.8122	0.9149
	Settling Time [s]	3.3351	5.0759
	Overshoot [%]	1.4904	6.7221
	ISE	0.4558	0.4658
ISE-AB	Rise Time [s]	0.9532	1.3752
	Settling Time [s]	1.8481	3.6533
	Overshoot [%]	0.3550	3.1481
	ISE	0.4660	0.5347
	ISE-AB	3.5746	4.0468
ISE-ST	Rise Time [s]	1.0120	1.6997
	Settling Time [s]	1.8747	2.8462
	Overshoot [%]	1.2320	1.9231
	ISE	0.4709	0.5979
	ISE-ST	0.6583	0.8825
ISE-OS	Rise Time [s]	1.1723	1.5896
	Settling Time [s]	3.5311	5.5525
	Overshoot [%]	0	0.1948
	ISE	0.4735	0.5724
	ISE-OS	0.4753	0.5735

The optimimized PID gains using the technique proposed in [65] and the proposed NLTA-based algorithm with the different objective functions are listed in Table 5.4. The cart and pendulum position responses as well as the control signal and the cumulative cart position error $\int_0^t |e_x(\tau)|\,\mathrm{d}\tau$ are shown in Figure 5.5. The dynamical cost, transient and steady state characteristics are listed in Table 5.5. The results confirm that every objective function successfully achieved its target goal. The best rise time, settling time and

Table 5.4: PID control gains

Objective Function	Angle			Position		
	K_p^θ	K_i^θ	K_d^θ	K_p^x	K_i^x	K_d^x
PID [65]	-40	0	-8	-1	0	-3
ISE	-43.9238	1.2625	-6.1163	-2.8623	-0.0017	-3.5402
ISE-ST	-43.6806	0.8948	-6.2171	-2.5071	-0.0279	-3.2817
ISE-OS	-42.3380	-1.2595	-6.1730	-1.8106	0	-2.6507
ISE-AB	-43.8129	0.2949	-6.0142	-2.3795	0	-3.1028

overshoot were attained through the the ISE, ISE-ST, and ISE-OS cost functions, respectively. It is worth noticing that the four variants of the PID controllers optimized with the NLTA heuristic outperformed the method proposed in [65] in all aspects of the transient response and the steady-state characteristics. This is particularly evident from Table 5.5 and Figure 5.5d. It is worth mentioning that the last two columns of Table 5.5 contain performance index measures related to the LQR and the ANN optimizers. Although none of these optimizers are enabled yet in the control scheme, their performance measures were used as additional objective measures to compare the results of the various PID tuning schemes.

Table 5.5: Performance of PID control units tuned using different cost criteria

Method	Rise time [s]	Settling time [s]	Overshoot [%]	ISE	$\int_0^t U(X(\tau), u(\tau))d\tau$	$\int_0^t F_3(X(\tau), u(\tau))d\tau$
PID [65]	4.8914	9.6098	**0**	0.72255	800.1996	79.5381
PID (ISE)	**0.8900**	3.3351	3.4130	**0.4558**	686.3625	67.2464
PID (ISE-ST)	1.0120	**1.8747**	1.7828	0.4709	674.2809	66.3176
PID (ISE-OS)	1.3283	3.8942	**0**	0.5093	**662.3581**	**65.4674**
PID (ISE-AB)	1.0578	1.9593	1.2077	0.4751	665.5093	65.5258

5.4.2 Combined Control Schemes

We will now study the performance of the full aggregate control scheme where the PID and the feedback optimization loops are both enabled. The weighting matrices which are used to find the Riccati solution and to establish the cost function of the neural network

are initialized to

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 250 \end{bmatrix} \qquad R = 1 \qquad Q^{NN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} \qquad R^{NN} = 0.016$$

The control gains calculated using the Riccati approach are [65]

$$K = [\,-137.7896 \;\; -25.9783 \;\; -22.3607 \;\; -27.5768\,]$$

The training parameters of the neural network are detailed in Table 5.6.

Table 5.6: Parameters of the Neural Network Optimizer

Parameters	Values
X_{min}, X_{max}	$\pm[0.175\,\text{red} \quad 0.35\,\text{rad/s} \quad 0.1\,\text{m} \quad 0.2\,\text{m/s}]$
u_{min}, u_{max}	$\pm 5\,\text{N}$
ΔX	$(X_{max} - X_{min})/20$
Δu^{NN}	$(u^{NN}_{max} - u^{NN}_{min})/20$
Number of hidden layers	1
Number of hidden neurons	13
Size of data set	194, 481 samples
Training data set	70%
Validation data set	15%
Testing data set	15%
Learning algorithm	Levenberg-Marquardt

The results are summarized in Figure 5.6 and Table 5.7. Once again, the proposed architecture along with the PID variants outperformed the combined PID-LQR controller suggested in [65]. The improvement in terms of index measures $\int_0^t U(X(\tau), u(\tau))\,\mathrm{d}\tau$ and $\int_0^t F_3(X(\tau), u(\tau))\,\mathrm{d}\tau$ reached up to 46% and 47%, respectively. The results also reveal how the ANN-based optimization loop was successful in reducing these two measures with respect to the PID control structure (comparing Tables 5.5 and 5.7).

Another simulation is carried out where the cart reference position $x^{ref}(t)$ follows a square wave command making the cart slides back and forth between 8 and 12 cm. The

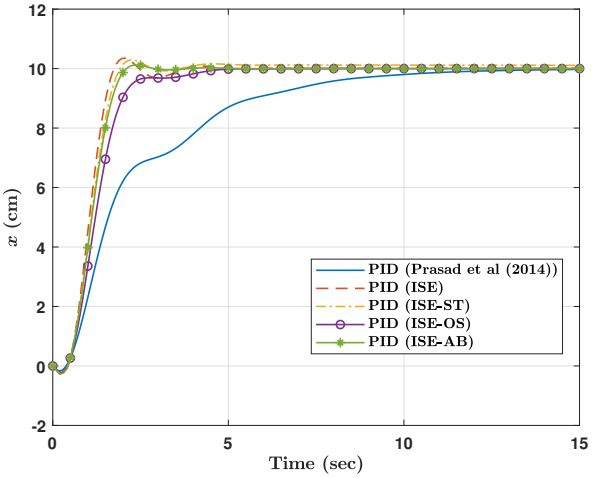
Table 5.7: Performance using combined control structures (PID+LQR versus PID+NN)

Method	Rise time (sec)	Settling time (sec)	Overshoot	ISE	$\int_0^t U(X(\tau), u(\tau))d\tau$	$\int_0^t F_3(X(\tau), u(\tau))d\tau$
PID + LQR [65]	3.2407	6.1969	**0**	1.1437	1207.6	120.5957
PID (ISE) + NN	0.9546	**3.3273**	0.6194	**0.4576**	672.7276	65.8731
PID (ISE-ST) + NN	1.1275	3.5240	0.1413	0.4733	661.1241	65.0051
PID (ISE-OS) + NN	2.0745	4.3102	**0**	0.5199	659.5133	65.1875
PID (ISE-AB) + NN	1.2055	3.4214	0.0103	0.4790	**654.4692**	**64.4231**

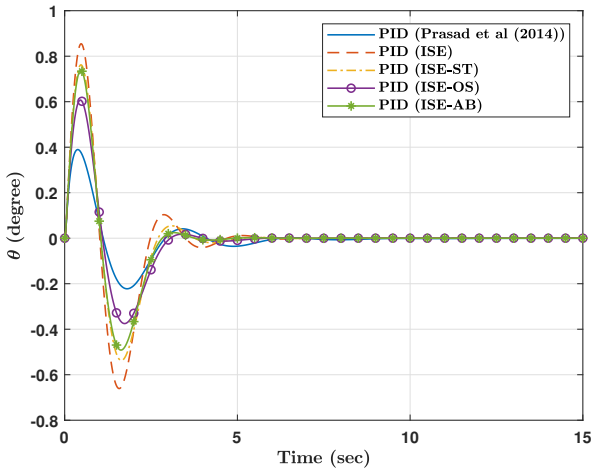
simulation is conducted with and without the control optimization loop. The results are displayed in Figures 5.7 and 5.8. Here too, the proposed control scheme showed to lead to better transient and steady-state characteristics and overall dynamic cost than the controller of [65].

5.5 Conclusion

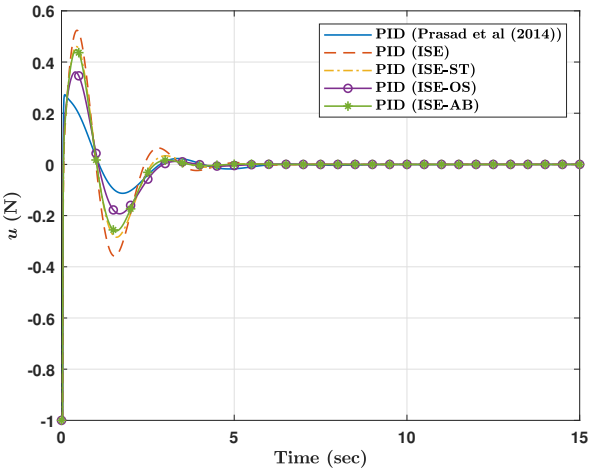
This chapter presents a generalization of the proposed NLTA-based PID along with the ANN optimization control structures to multi-output systems. The techniques were applied to a balance system with two outputs. A PID control unit was applied to track each output. We demonstrated how various objective functions can be integrated in the NLTA algorithm to search for the optimized PID gains to satisfy certain design criteria pertaining to the transient response and the steady-state characteristics, such as the settling time and the overshoot, for instance. The results were benchmarked against a control algorithm suggested in the literature. It was outperformed by the proposed control structures in all the conducted simulations with and without the ANN optimization loop. The study showed an improvement in the optimization cost measures of up to 47%. One of the salient features of the proposed control schemes is that they do not require a prior knowledge of the system dynamics. However, they do depend on a known region of stability for the control gains to be used as a search space by the NLTA algorithm.



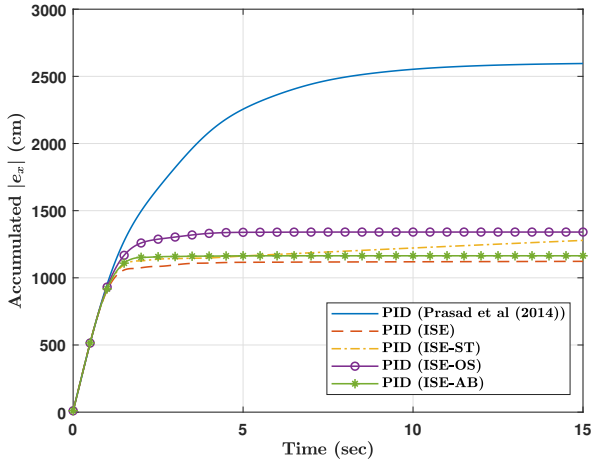
(a) Cart position x



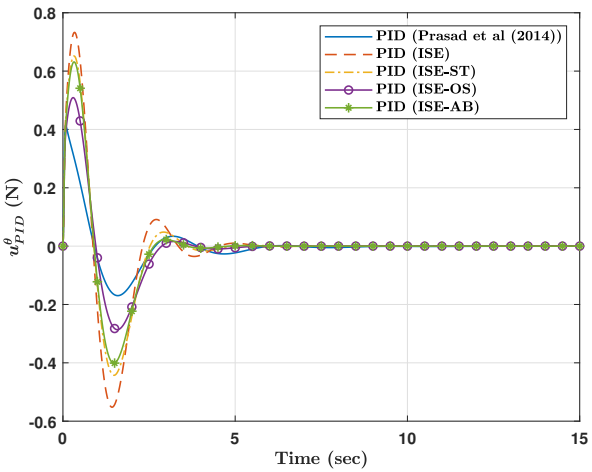
(b) Pendulum angle θ



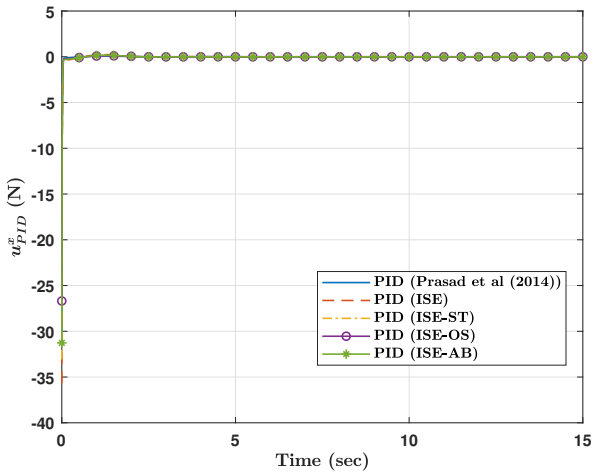
(c) Control signal u



(d) Cumulative car position error

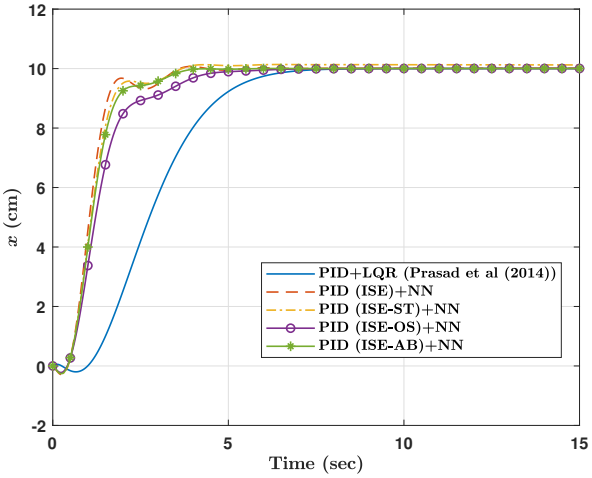


(e) Control signal u^{θ}_{PID}

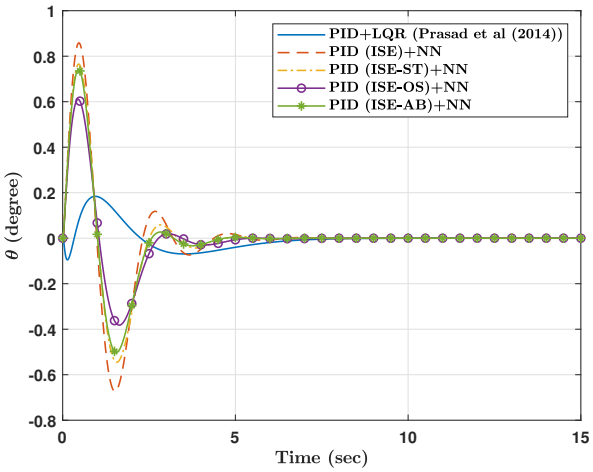


(f) Control signal u^{x}_{PID}

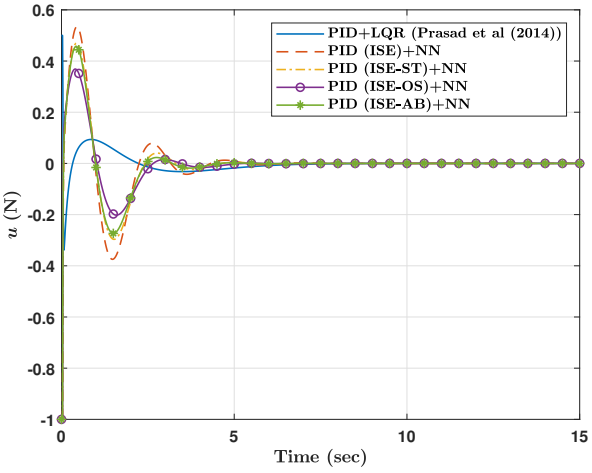
Figure 5.5: System performance under PID control structures



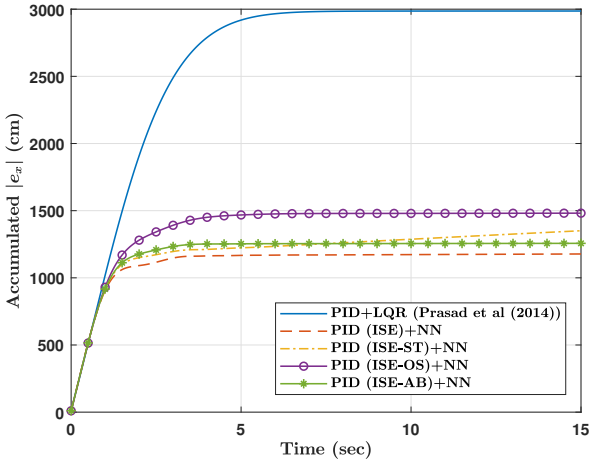
(a) Cart position x



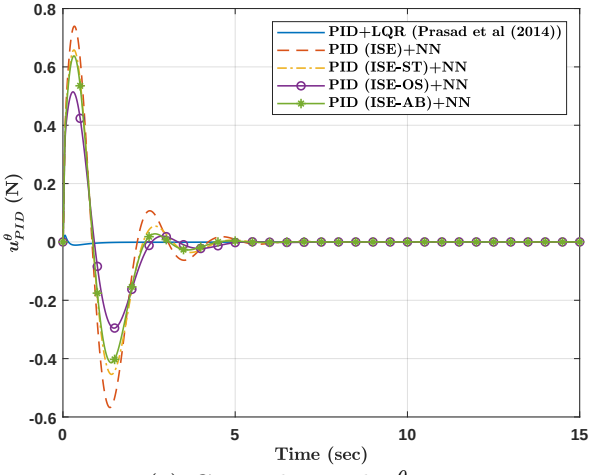
(b) Pendulum angle θ



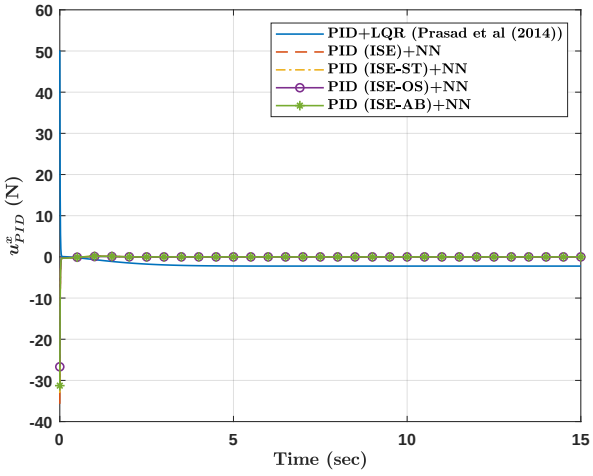
(c) Control signal u



(d) Cumulative car position error

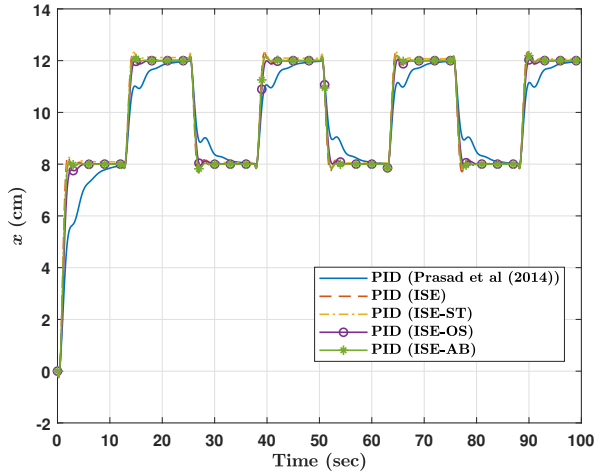


(e) Control signal u^{θ}_{PID}

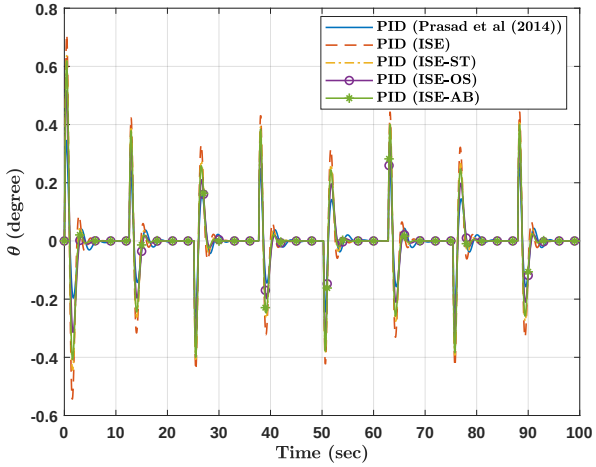


(f) Control signal u^{x}_{PID}

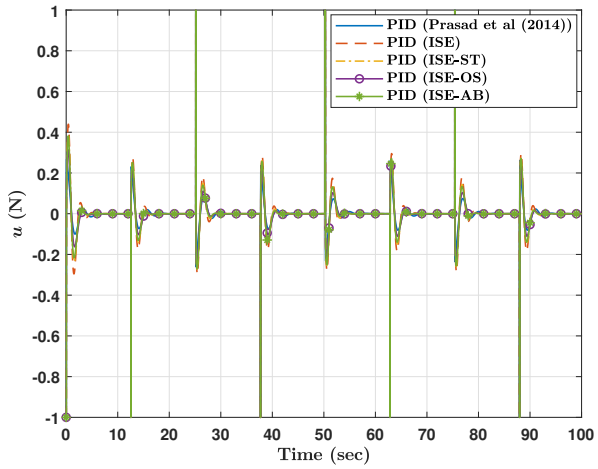
Figure 5.6: System performance under aggregate control structures



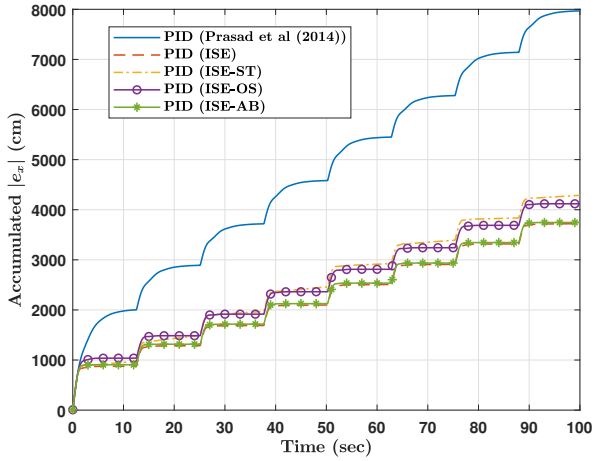
(a) Cart position x



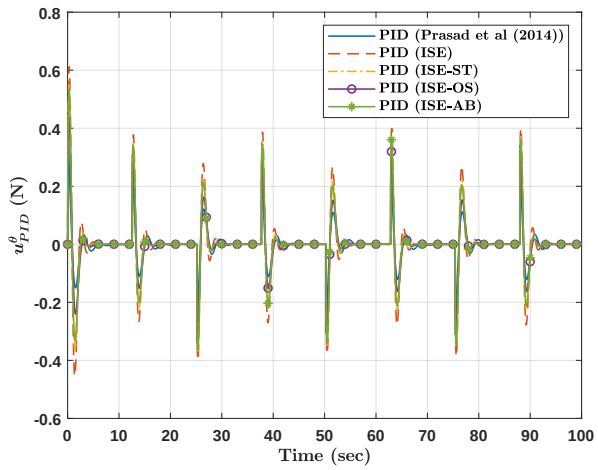
(b) Pendulum angle θ



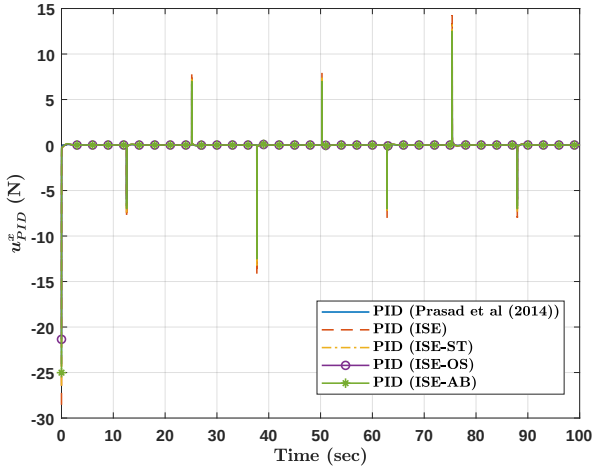
(c) Control signal u



(d) Cumulative car position error

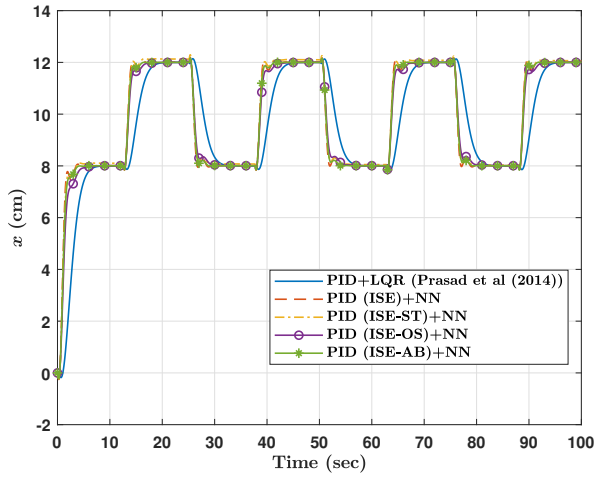


(e) Control signal u^{θ}_{PID}

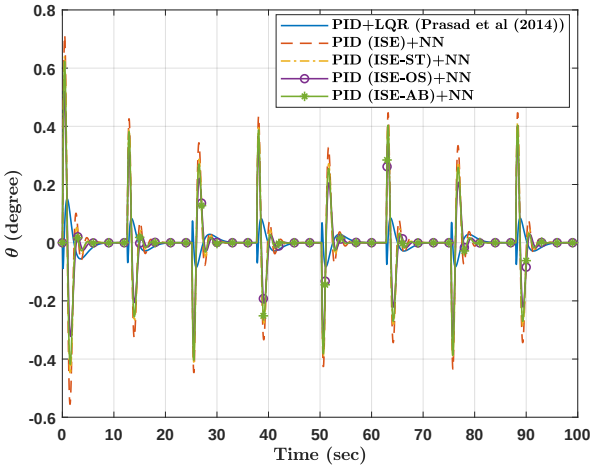


(f) Control signal u^{x}_{PID}

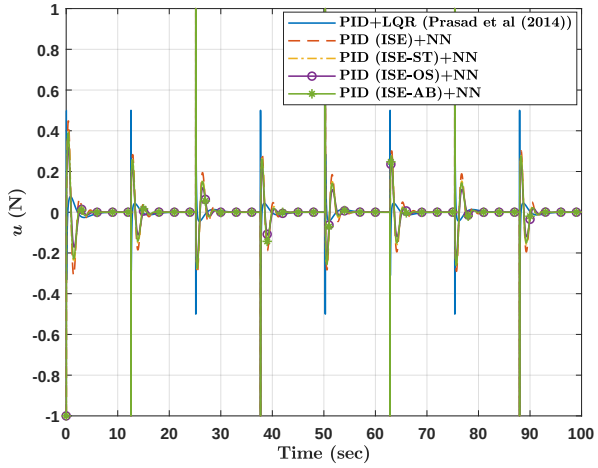
Figure 5.7: System performance with square wave reference under PID control structures



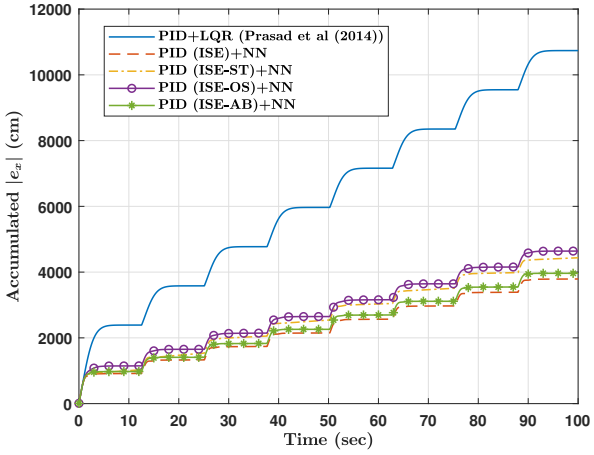
(a) Cart position x



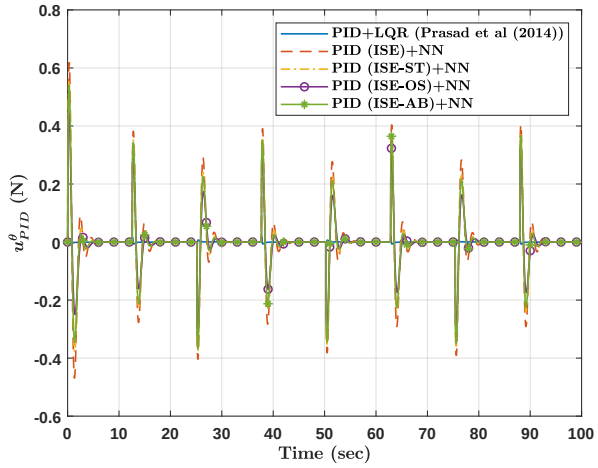
(b) Pendulum angle θ



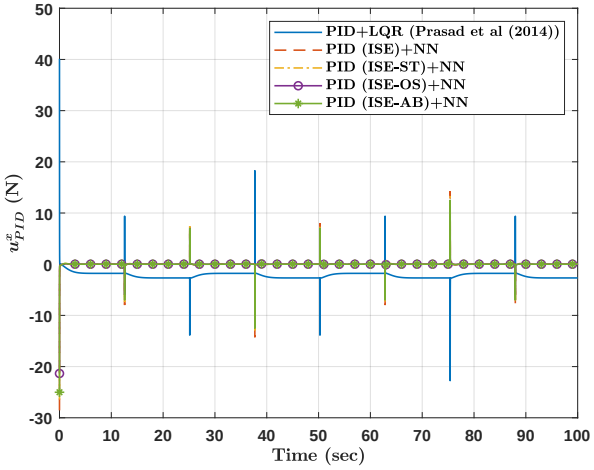
(c) Control signal u



(d) Cumulative car position error



(e) Control signal u^{θ}_{PID}



(f) Control signal u^{x}_{PID}

Figure 5.8: System performance with square wave reference under aggregate control structures

Chapter 6

Conclusion

The thesis investigates a model-free heuristic framework for optimized tracking control problems by incorporating the tracking error within a broader cost function in which more signals can be integrated. Such signals can include design-specific requirements, for example, such as the response overshoot, settling time, etc. The proposed heuristics do not explicitly depend on the prior knowledge of the plant's dynamic model, as they treat the latter as a black box. However, certain knowledge is still required about at least a subset of the stability region of the control gains to be optimized. That region is searched by the heuristic algorithms to determine an optimized set of gains. The developed algorithms make use of a tool chain of model-free optimizers; namely, Q-learning , artificial neural networks (ANN) and nonlinear threshold accepting (NLTA) technique. The proposed architecture is based on two control loops. The first loop only addresses the tracking error. It does so using either Q-learning or the NLTA approach. The Q-learning generates an optimized control policy based on tracking error combinations, while the NLTA approach is used to tune the optimized control gains of a conventional PID controller. Results proved that the latter can be more advantageous as it leads to smoother control signals. The second loop is the system optimization loop. This is where a system-wide cost function is optimized through a state feedback. This loop is implemented using a feedforward ANN. Results showed that it can be superior to the conventional LQR method.

At first, the control architecture was applied to a single-input single-output roll control

mechanism of a flexible wing aircraft with time-dependent dynamics. The tracking control loops were realized using Q-learning and NLTA-tuned PID controllers. It was shown that the latter generated smoother control signals and convergent tracking errors.

After that, the control architecture was generalized to multi-input multi-output systems, where a PID controller is assigned for the convergence of each tracking error. The idea was demonstrated on a two-input two-output balance system. Building on the conclusions of the comparison study between the Q-learning and NLTA-based PID controller, only the latter was applied in this demonstration. The ANN-based state feedback optimizer was benchmarked with an LQR technique suggested in the literature. Our results illustrated the superiority of our proposed architecture despite feeding the plant's dynamics as an input to the LQR approach. In this example, we also demonstrated how various objective functions can be integrated in the control architecture to satisfy different design objectives.

6.1 Future Work

A number of avenues can be taken to further extend the current work. The following are probably among the most prominent.

- The PID control gain tuning and the ANN training are performed offline. It might be beneficial to look into some ways to do that online instead.
- One of the major hurtles associated to Q-learning is its discrete nature. Q-learning implements optimized policies of transiting between a set of discrete states by applying actions from a discrete action space. This "jumping" between discrete states/actions leads to a chattering behavior when applied to a continuous-time system. A possible fruitful research direction would be to investigate methods to convert Q-learning actions from a discrete to a continuous space. Such a breakthrough would have a major impact on applying Q-learning to all continuous-time systems in general, and not only this work.

- Another limitation of the proposed heuristic is its dependency on a priori known stable regions of the PID gains so that they are used as a search space for the NLTA algorithm. Future efforts on determining such regions may take advantage of the recent literature in this field, such as the work in [16], for example.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[2] M. Abouheaf, N. Mailhot, and W. Gueaieb. An online reinforcement learning wing-tracking mechanism for flexible wing aircraft. In *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–7, June 2019.

[3] M. I. Abouheaf and F. L. Lewis. Multi-agent differential graphical games: Nash online adaptive learning solutions. In *52nd IEEE Conference on Decision and Control*, pages 5803–5809, 12 2013.

[4] M. I. Abouheaf, F. L. Lewis, and M. S. Mahmoud. Differential graphical games: Policy iteration solutions and coupled riccati formulation. In *2014 European Control Conference (ECC)*, pages 1594–1599, 2014.

[5] M. I. Abouheaf, F. L. Lewis, and M. S. Mahmoud. Model-free adaptive learning solutions for discrete-time dynamic graphical games. In *53rd IEEE Conference on Decision and Control*, pages 3578–3583, 12 2014.

[6] Mohammed Abouheaf and Wail Gueaieb. Neurofuzzy Reinforcement Learning Control Schemes for Optimized Dynamical Performance. In *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–7, June 2019.

[7] Mohammed Abouheaf, Frank Lewis, Kyriakos Vamvoudakis, Sofie Haesaert, and Robert Babuska. Multi-agent discrete-time graphical games and reinforcement learning solutions. *Automatica*, 50(12):3038–3053, 2014.

[8] Mohammed I. Abouheaf, Sofie Haesaert, W. Lee, and Frank L. Lewis. Q-learning with eligibility traces to solve non-convex economic dispatch problems. *Intern Jour of Electr Scien and Eng*, 7(7):1390–1396, 2012.

[9] Mohammed I Abouheaf and Frank L Lewis. Dynamic graphical games: Online adaptive learning solutions using approximate dynamic programming. In *Frontiers of Intelligent Control and Information Processing*, pages 1–48. World Scientific, 2015.

[10] R. Anushree and B.K. Swathi Prasad. Design and development of novel control strategy for trajectory tracking of mobile robot: Featured with tracking error minimization. In *2016 IEEE Annual India Conference (INDICON)*, pages 1–6, December 2016. ISSN: 2325-9418.

[11] Karl Johan Åström, Tore Hägglund, Chang C Hang, and Weng K Ho. Automatic tuning and adaptation for pid controllers-a survey. *Control Engineering Practice*, 1(4):699–714, 1993.

[12] J. Andrew Bagnell and Jeff G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.

[13] Reza Beigzadeh, Masoud Rahimi, and Seyed Reza Shabanian. Developing a feed forward neural network multilayer model for prediction of binary diffusion coefficient in liquids. *Fluid Phase Equilibria*, 331:48–57, 2012.

[14] Stuart Bennett. A brief history of automatic control. *IEEE Control Systems Magazine*, 16(3):17–25, 1996.

[15] Ilse Cervantes and Jose Alvarez-Ramirez. On the PID tracking control of robot manipulators. *Systems & Control Letters*, 42(1):37–46, 2001.

[16] Ankush Chakrabarty, Claus Danielson, Stefano Di Cairano, and Arvind Raghunathan. Active learning for estimating reachable sets for systems with unknown dynamics. *IEEE Transactions on Cybernetics*, pages 1–12, 2020.

[17] Chien-Chern Cheah, Chao Liu, and Jean-Jacques E. Slotine. Adaptive tracking control for robots with unknown kinematic and dynamic properties. *The International Journal of Robotics Research*, 25(3):283–296, 2006.

[18] Bing Chen, Xiaoping Liu, and Shaocheng Tong. Adaptive fuzzy output tracking control of mimo nonlinear uncertain systems. *IEEE Transactions on Fuzzy Systems*, 15(2):287–300, 2007.

[19] Fu-Chuang Chen and Hassan K. Khalil. Adaptive control of nonlinear systems using neural networks. *International journal of control*, 55(6):1299–1317, 1992.

[20] Zhenzhong Chu, Daqi Zhu, and Simon X. Yang. Observer-Based Adaptive Neural Network Trajectory Tracking Control for Remotely Operated Vehicle. *IEEE Transactions on Neural Networks and Learning Systems*, 28(7):1633–1645, July 2017. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[21] M. Cook and M. Spottiswoode. Modelling The Flight Dynamics of The Hang Glider. *The Aeronautical Journal*, 109(1102):I–XX, 2006.

[22] J. Craig, Ping Hsu, and Shankar Sastry. Adaptive control of mechanical manipulators. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 190–195. IEEE, April 1986.

[23] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in neural information processing systems*, pages 1017–1023, 1996.

[24] Robert H. Crites and Andrew G. Barto. Elevator Group Control Using Multiple Reinforcement Learning Agents. *Machine Learning*, 33(2):235–262, November 1998.

[25] Guido De Matteis. Response of Hang Gliders to Control. *The Aeronautical Journal*, 94(938):289–294, 1990.

[26] Guido de Matteis. Dynamics of hang gliders. *Journal of Guidance Control and Dynamics*, 14(6):1145–1152, 1991.

[27] Rafael Fierro and Frank L. Lewis. Control of a nonholomic mobile robot: Backstepping kinematics into dynamics. *Journal of robotic systems*, 14(3):149–163, 1997.

[28] Rafael Figueroa, Aleksandra Faust, Patricio Cruz, Lydia Tapia, and Rafael Fierro. Reinforcement learning for balancing a flying inverted pendulum. In *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pages 1787–1793. IEEE, June 2014.

[29] Takanori Fukao, Hisashi Nakagawa, and Norihiko Adachi. Adaptive tracking control of a nonholonomic mobile robot. *IEEE transactions on Robotics and Automation*, 16(5):609–615, 2000.

[30] Meng-Hock Fun and Martin T. Hagan. Levenberg-marquardt training for modular networks. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 468–473. IEEE, 1996.

[31] Beatrice A. Golomb, David T. Lawrence, and Terrence J. Sejnowski. Sexnet: A neural network identifies sex from human faces. In *NIPS*, volume 1, pages 572–579, 1990.

[32] Martin T. Hagan and Mohammad B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993, 1994.

[33] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, 1990.

[34] A. K. Jain, Jianchang Mao, and K. M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, March 1996.

[35] Jiangdagger, Zhong-Ping, and Henk Nijmeijer. Tracking control of mobile robots: a case study in backstepping. *Automatica*, 33(7):1393–1399, 1997.

[36] J. Stephen Judd. *Neural Network Design and the Complexity of Learning*. The MIT Press, 1990.

[37] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[38] Yutaka Kanayama, Yoshihiko Kimura, Fumio Miyazaki, and Tetsuo Noguchi. A stable tracking control method for a non-holonomic mobile robot. In *Proceedings IROS'91: IEEE/RSJ International Workshop on Intelligent Robots and Systems' 91*, pages 1236–1241. IEEE, 1991.

[39] Andreas Kaplan and Michael Haenlein. Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*, 62(1):15–25, January 2019.

[40] Byoung S. Kim and Anthony J. Calise. Nonlinear flight control using neural networks. *Journal of Guidance, Control, and Dynamics*, 20(1):26–33, 1997.

[41] 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*, 29(6):2042–2062, 2018.

[42] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[43] Birol Kocaturk. Motion Control of Wheeled Mobile Robots. *UPRAVLJANJE MOBILNIM ROBOTOM S KOTAČIMA.*, 13(1):41–47, January 2015. Publisher: Croatian Interdisciplinary Society.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6):84–90, June 2017. Publisher: Association for Computing Machinery.

[45] Ching-Hung Lee. A survey of pid controller design based on gain and phase margins. *International Journal of Computational Cognition*, 2(3):63–100, 2004.

[46] F. L. Lewis. Neural network control of robot manipulators. *IEEE Expert*, 11(3):64–75, June 1996.

[47] Frank L. Lewis, Draguna Vrabie, and Vassilis L. Syrmos. *Optimal Control*. John Wiley & Sons, March 2012. Google-Books-ID: U3Gtlot_hYEC.

[48] P. Liang and N.K. Bose. Neural network fundamentals with graphs, algorithms and applications. *Mac Graw-Hill*, 1996.

[49] Chuan-Kai Lin. h_∞ reinforcement learning control of robot manipulators using fuzzy wavelet networks. *Fuzzy Sets and Systems*, 160(12):1765–1786, 2009.

[50] D. M. Liu, G. Naadimuthu, and E. S. Lee. Trajectory tracking in aircraft landing operations management using the adaptive neural fuzzy inference system. *Computers & Mathematics with Applications*, 56(5):1322–1327, 2008.

[51] Nguyen Tan Luy. Reinforecement learning-based optimal tracking control for wheeled mobile robot. In *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 371–376. IEEE, May 2012.

[52] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence*, 55(2-3):311–365, 1992.

[53] Toshifumi Minemoto, Teijiro Isokawa, Haruhiko Nishimura, and Nobuyuki Matsui. Feed forward neural network with random quaternionic neurons. *Signal Processing*, 136:59–68, 2017. Hypercomplex Signal Processing.

[54] Nabil Nahas, Mohammed Abouheaf, Adel Sharaf, and Wail Gueaieb. A Self-Adjusting Adaptive AVR-LFC Scheme for Synchronous Generators. *IEEE Transactions on Power Systems*, pages 5073–5075, 2019.

[55] Nabil Nahas, Mohamed Noomane Darghouth, and Mohammed Abouheaf. A Non-Linear-Threshold-Accepting Function Based Algorithm for the Solution of Economic Dispatch Problem. *RAIRO - Operations Research*, April 2019.

[56] Nabil Nahas and Mustapha Nourelfath. Nonlinear threshold accepting meta-heuristic for combinatorial optimisation problems. *International Journal of Metaheuristics*, 3(4):265–290, 2014.

[57] Nahas Nabil. Non-linear hreshold algorithm based solution for the redundancy allocation problem considering multiple redundancy strategies. *Journal of Quality in Maintenance Engineering*, 25(3):397–411, January 2019.

[58] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. Springer, 2006.

[59] Julio E. Normey-Rico, Ismael Alcalá, Juan Gómez-Ortega, and Eduardo F. Camacho. Mobile robot path tracking using a robust pid controller. *Control Engineering Practice*, 9(11):1209–1214, 2001.

[60] Maciej Oczak, Stefano Viazzi, Gunel Ismayilova, Lilia T Sonoda, Nancy Roulston, Michaela Fels, Claudia Bahr, Jörg Hartung, Marcella Guarino, Daniel Berckmans, et al. Classification of aggressive behaviour in pigs by activity index and multilayer feed forward neural network. *Biosystems Engineering*, 119:89–97, 2014.

[61] Aidan O'Dwyer. *Handbook of PI and PID controller tuning rules*. Imperial college press, 2009.

[62] Dong C. Park, M. A. El-Sharkawi, R. J. Marks, L. E. Atlas, and M. J. Damborg. Electric load forecasting using an artificial neural network. *IEEE transactions on Power Systems*, 6(2):442–449, 1991.

[63] Vicente Parra-Vega, Suguru Arimoto, Yun-Hui Liu, Gerd Hirzinger, and Prasad Akella. Dynamic sliding pid control for tracking of robot manipulators: Theory and experiments. *IEEE Transactions on Robotics and Automation*, 19(6):967–976, 2003.

[64] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

[65] Lal Bahadur Prasad, Barjeev Tyagi, and Hari Om Gupta. Optimal Control of Nonlinear Inverted Pendulum System Using PID Controller and LQR: Performance Analysis Without and With Disturbance Input. *International Journal of Automation and Computing*, 11(6):661–670, December 2014.

[66] D. Psaltis, A. Sideris, and A.A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, 8(2):17–21, April 1988. Conference Name: IEEE Control Systems Magazine.

[67] M. F. Rahmat. Application of selftuning fuzzy pid controller on industrial hydraulic actuator using system identification approach. In *International Journal on Smart Sensing and Intelligent Systems*. Citeseer, 2009.

[68] Filoktimon Repoulias and Evangelos Papadopoulos. Planar trajectory planning and tracking control design for underactuated auvs. *Ocean Engineering*, 34(11-12):1650–1667, 2007.

[69] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

[70] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38, 1998.

[71] Atheer L. Salih, M. Moghavvemi, Haider A. F. Mohamed, and Khalaf Sallom Gaeid. Flight pid controller design for a uav quadrotor. *Scientific research and essays*, 5(23):3660–3667, 2010.

[72] Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[73] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[74] J. Shan, H.-T. Liu, and S. Nowotny. Synchronised trajectory-tracking control of multiple 3-dof experimental helicopters. *IEE Proceedings-Control Theory and Applications*, 152(6):683–692, 2005.

[75] Wai Wai Shein, Yasuo Tan, and Azman Osman Lim. Pid controller for temperature control with multiple actuators in cyber-physical home system. In *2012 15th International Conference on Network-Based Information Systems*, pages 423–428. IEEE, September 2012.

[76] Samarth Singh and Ranajit Mitra. Comparative analysis of robustness of optimally offline tuned pid controller and fuzzy supervised pid controller. In *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, pages 1–6, 3 2014.

[77] Jean-Jacques Slotine and S Shankar Sastry. Tracking control of non-linear systems using sliding surfaces, with application to robot manipulators. *International journal of control*, 38(2):465–492, 1983.

[78] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, September 2005.

[79] Richard S. Sutton, Andrew G. Barto, et al. *Introduction to Reinforcement Learning*, volume 135. MIT press Cambridge, 1998.

[80] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[81] Grzegorz Szafranski and Roman Czyba. Different approaches of pid control uav type quadrotor. In *The International Micro Air Vehicles Conference*, pages 70–75, January 2011.

[82] M. Osman Tokhi and Abul K. M. Azad. *Flexible Robot Manipulators: Modelling, Simulation and Control*. IET, 2008.

[83] Patrizio Tomei. Adaptive pd controller for robot manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):565–570, 1991.

[84] Rong-Jong Wai. Tracking control based on neural network strategy for robot manipulator. *Neurocomputing*, 51:425–445, April 2003.

[85] Danshi Wang, Min Zhang, Jin Li, Ze Li, Jianqiang Li, Chuang Song, and Xue Chen. Intelligent constellation diagram analyzer using convolutional neural network-based deep learning. *Optics Express*, 25(15):17150, July 2017.

[86] Ning Wang, Mohammed Abouheaf, and Wail Gueaieb. Data-driven optimized tracking control heuristic for mimo structures: A balance system case study. In *IEEE International Conference on Systems, Man, and Cybernetics*, Toronto, Canada, October 2020. IEEE.

[87] Ning Wang, Mohammed Abouheaf, Wail Gueaieb, and Nabil Nahas. Model-free optimized tracking control heuristic. *Robotics*, 9(3):1–24, June 2020.

[88] Ying Wang and Clarence W. De Silva. Multi-robot box-pushing: Single-agent qqlearning vs. team q-learning. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3694–3699. IEEE, 2006.

[89] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[90] Christopher John Cornish Hellaby Watkins. *Learning From Delayed Rewards*. phdthesis, King's College, Halifax, Nova Scotia, Canada, 1989.

[91] Dong Xu, Dongbin Zhao, Jianqiang Yi, and Xiangmin Tan. Trajectory Tracking Control of Omnidirectional Wheeled Mobile Manipulators: Robust Neural Network-Based Sliding Mode Approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part*

B (Cybernetics), 39(3):788–799, June 2009. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics).

[92] Jinyi Yao, Jiang Chen, and Zengqi Sun. An application in RoboCup combining Q-learning with adversarial planning. In *Proceedings of the 4th World Congress on Intelligent Control and Automation (Cat. No.02EX527)*, volume 1, pages 496–500 vol.1, June 2002.

[93] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[94] Huaguang Zhang, Qinglai Wei, and Yanhong Luo. A novel infinite-time optimal tracking control scheme for a class of discrete-time nonlinear systems via the greedy hdp iteration algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):937–942, 2008.