University of Alberta

Dynamic Tuning of PI-Controllers based on Model-Free Reinforcement Learning Methods

by

Lena Abbasi Brujeni

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

in

Process Control

Department of Chemical and Materials Engineering

Spring 2010
Edmonton, Alberta

Examining Committee

Dr. Jong Min Lee, Chemical and Materials Engineering

Dr. Sirish L. Shah, Chemical and Materials Engineering

Dr. Vinay Prasad, Chemical and Materials Engineering

Dr. Richard Sutton, Computing Science

To My Parents
You are my everything.

Abstract

In this thesis, a Reinforcement Learning (RL) method called Sarsa is used to dynamically tune a PI-controller for a Continuous Stirred Tank Heater (CSTH) experimental setup. The proposed approach uses an approximate model to train the RL agent in the simulation environment before implementation on the real plant. This is done in order to help the RL agent initially start from a reasonably stable policy. Learning without any information about the dynamics of the process is not practically feasible due to the great amount of data (time) that the RL algorithm requires and safety issues.

The process in this thesis is modeled with a First Order Plus Time Delay (FOPTD) transfer function, because almost all of the chemical processes can be sufficiently represented by this class of transfer functions. The presence of a delay term in this type of transfer functions makes them inherently more complicated models for RL methods.

RL methods should be combined with generalization techniques to handle the continuous state space. Here, parameterized quadratic function approximation compounded with k-nearest neighborhood function approximation is used for the regions close and far from the origin, respectively. Applying each of these generalization methods separately has some disadvantages, hence their combination is used to overcome these flaws.

The proposed RL-based PI-controller is initially trained in the simulation environment. Thereafter, the policy of the simulation-based RL agent is used as the starting policy of the RL agent during implementation on the experimental setup. As a result of the existing plant-model mismatch, the performance of the RL-based PI-controller using this primary policy is not as good as the simulation

results; however, training on the real plant results in a significant improvement in this performance. On the other hand, the IMC-tuned PI-controllers, which are the most commonly used feedback controllers are also compared and they also degrade because of the inevitable plant-model mismatch. To improve the performance of these IMC-tuned PI-controllers, re-tuning of these controllers based on a more precise model of the process is necessary.

The experimental tests are carried out for the cases of set-point tracking and disturbance rejection. In both cases, the successful adaptability of the RL-based PI-controller is clearly evident.

Finally, in the case of a disturbance entering the process, the performance of the proposed model-free self-tuning PI-controller degrades more, when compared to the existing IMC controllers. However, the adaptability of the RL-based PI-controller provides a good solution to this problem. After being trained to handle disturbances in the process, an improved control policy is obtained, which is able to successfully return the output to the set-point.

Acknowledgements

I would like to acknowledge my supervisors, Dr. Jong Min Lee and Dr. Sirish L. Shah, for their continuous support, and patience throughout my studies.

I should also thank Dr. Csaba Szepesvari for being always open to my questions, and for his helpful comments on my research project.

Special thanks for my friends for the time they occasionally spent with me discussing my research, and editing my thesis.

Finally, I would like to thank the examining committee for the time and effort they are spending on this defense.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations and Objectives

PI-controllers are the most practically used controllers in industry [Astrom and Hagglund, 1988]. The tuning of these controllers is of great importance as it significantly affects their control performance. There are various methods documented in the literature for the tuning of PI-controllers, which may not necessarily result in the optimum gains of the controller due to either lack of any optimization or the existing plant-model mismatch. Model based tuning methods are generally related to the development of some form of the process model and their performance is highly dependent on its accuracy. Yet, obtaining a precise process model is a non-trivial task. In addition, if the dynamics of the process change through time, considerable amount of time, effort, and money needs be spent on re-tuning or re-designing of the in-use controllers.

These shortcomings motivate the application of reinforcement learning (RL) methods in the dynamic tuning of PI-controllers. Model-free RL methods do not require a model of the system (though any prior knowledge about the system, such as a rough model, or a structure of the cost function, can be incorporated into them to increase their performance). In addition, RL methods continuously receive a feedback signal from their environment, so they sense any changes in the dynamics of the process and adapt to those changes to keep the control performance satisfactory.

Furthermore, RL methods have a simpler optimization step, compared to Model

Predictive Control, or MPC, which is a popular control technique for industrial multivariate control problems. MPC deals with a complex multi-step optimization problem that is computationally expensive to solve online (specially for non-linear systems). On the other hand, RL solves a single-step optimization problem, which is an easier task to perform in real time.

The application of the RL methods to chemical process control is a harder task compared to robotics, because chemical processes are involved with continuous state and continuous action spaces. In addition, the key component of model-free RL methods, exploration, can be troublesome in chemical processes because of safety issues. Therefore, model-free RL methods should be adjusted for application on a chemical process. To have a safer exploration, we have decided to keep the structure of our controller fixed as a PI-controller, and use RL methods to set the gains of this controller, instead of working with the control signal directly.

The proposed approach in this thesis builds a cost-to-go function from the data obtained from closed-loop simulations and experiments. This cost-to-go ensures that the cost-to-go-function-based RL methods do not need the model of the system. Through the optimization of the cost-to-go, at each step of the simulation or experiment, the best gains for the PI-controller are calculated, and then the estimate of the cost-to-go function is corrected based on the feedback earned from the process. Both simulation and experimental results are provided for set-point tracking, while disturbance rejection is also tested on the experimental plant.

1.2 Overview

This thesis is organized as follows: Chapter 2 provides the theoretical background, which is prerequisite for understanding the proposed approach in this thesis, including an introduction to different RL methods and a brief description of several tuning methods. Chapter 3, presents a literature review of some published papers and theses, related to this topic. Chapter 4 explains the author's research work and contains the supporting simulation results, along with their experimental validation. This is followed by the concluding remarks and future work in Chapter 5.

1.3 Contribution of this thesis

In this thesis, the application of model-free reinforcement learning methods is examined on a practical case of study in process control. This case of study is the dynamic tuning of a PI-controller for a Continuous Stirred Tank Heater (CSTH) experimental setup. The RL approach used here does not require a dynamic model; however any existing model, accurate or even otherwise, can be used to enhance its performance. In fact, to prevent safety problems that may occur due to the implementation of pure RL on the pilot plant, a rough model of the process is assumed to be at hand. This rough model is used to built a simulation environment. The RL agent is initially trained in this simulation environment. Thereafter, The policy of the simulation-based RL agent is used as the initial policy of the RL agent in experiment to begin training in experimental environment from a better first policy. The process in the simulation environment is modeled as an FOPTD transfer function. This type of transfer functions are more complicated for the application of RL methods, as a result of their delay terms. In addition, because of the continuity of the state space defined in this thesis, parameterized quadratic function approximation in combination with k-nearest neighborhood is used as a generalization technique. This combination is used as a means to overcome the flaws of applying each of the mentioned generalization methods separately.

Finally, the successful implementation of the proposed approach is provided on the CSTH pilot plant. To our best knowledge, we are the first to provided experimental results for the application of RL methods on a chemical plant. Through this experimental evaluations, significant adaptation is observed for the RL-based PI-controller, which makes it an appealing approach in the case of process model change.

Chapter 2

Theoretical Background

2.1 Introduction to PI-Controllers

PI-controllers are the most common single-input-single-output (SISO) feedback controllers in-use in industry [Astrom and Hagglund, 1988]. This is mainly because they are reliable and easy to implement by field engineers. The standard format of the controller that is used in this thesis is shown in Equation (2.1):

$$u_t = K_c e_t + K_i \int_0^t e_\tau d\tau \tag{2.1}$$

in which e_t represents the feedback error of the system at time t and is shown in Equation (2.2).

$$e_t = r_t - y_t \tag{2.2}$$

Here r_t is the set-point signal, while y_t is the output value. In the discrete format, the summation of the error takes the place of the integral term and Equation (2.1) changes to Equation (2.3):

$$u_k = K_c e_k + K_i T_s \sum_{j=0}^{k} e_j \tag{2.3}$$

In Equation (2.3), T_s is the sample time used for discretizing. Here, the sensor noise and disturbances to the system are assumed to be unknown. As shown in Equation (2.1), the control signal consists of two different parts: the proportional and integral terms. Increasing the gain of the proportional part will reduce the sensitivity of the output of the system to the load disturbances, while the integral term drives the process output towards the set-point and eventually eliminates

the steady-state error [Seborg et al., 1989]. However, because the integral term responds to the summation of the errors from the past, it may result in the overshoot, undershoot or even produce oscillations in the output. To prevent the side effects of the integral term, such as increasing the settling time and the overshoot (undershoot), a derivative term can be added to the controller law [Seborg et al., 1989]. This will speed up the rate at which the output changes. One disadvantage of including the derivative action is its sensitivity to measurement noise, which in the case of large noise and derivative gain, can result in an unstable system [Seborg et al., 1989].

2.1.1 Tuning of PI-Controllers

The performance of a PI controller is highly dependent on its gains. If these gains are chosen improperly, the output can show a big overshoot, become oscillatory, or even diverge. The adjustment of these controller gains to the optimum values for a desired control response is called *tuning of the control loop*. There are different methods documented in the literature for the tuning of PI-controllers, which may not necessarily result in the optimum gains of the controller. In addition, the most effective tuning methods are model based. This means that they generally require the availability of some form of the process model. There are also manual tuning methods which can be relatively inefficient. In the following paragraphs some common tuning methods are described and compared.

Ziegler-Nichols (Z-N) Method [Ziegler and Nichols, 1993]

The Ziegler-Nichols tuning method is a tuning technique presented both in online and off-line modes. The online version of this method is summarized in the following steps [Seborg et al., 1989]:

1. When the process reaches the steady state, set the integral (and derivative) gains equal to zero, so in this way the controller just includes the proportional term.

2. Put the controller in automatic mode with a small value of K_c.

Table 2.1: Controller settings based on the Ziegler-Nichols method

Ziegler-Nichols	K_c	K_i	K_d
P	$0.5K_{cu}$	—	—
PI	$0.45K_{cu}$	$\frac{1.2K_c}{P_u}$	—
PID	$0.6K_{cu}$	$\frac{2K_c}{P_u}$	$\frac{K_cP_u}{8}$

3. Change the set-point by a small amount. This will make the output move away from the set-point. Gradually increase the control gain till the output continuously oscillates with constant amplitude. The value of K_c that has resulted in this continuous cycling output is named the *ultimate gain*, and is denoted by K_{cu}. The period of the corresponding continuous cycling output is called the *ultimate period* and is denoted by P_u.

4. Applying the Z-N tuning rules mentioned in Table 2.1, calculate the PI-controller settings.

5. Evaluate the current settings and fine-tune.

For the off-line version, the ultimate gain and period introduced in Ziegler-Nichols tuning method are calculated based on the parameters of the model of the process. In this thesis, our main process characteristic of interest is First Order Plus Time Delay (FOPTD) transfer functions as the available model of the system. The general format for FOPTD models is shown in Equation (2.4) where K_p, θ, and τ are the gain, the time delay, and the time constant of the system, respectively. The reason for choosing an FOPTD model is that, most of the industrial chemical processes can be approximated by this type of transfer functions [Seborg et al., 1989].

$$G(s) = \frac{K_p}{\tau s + 1} \exp(-\theta s) \tag{2.4}$$

For FOPTD transfer functions, the ultimate gain and period can be derived from Equations (2.5) and (2.6).

$$\tau = \frac{P_u}{2\pi} \tan\left[\frac{\pi(P_u - 2\theta)}{P_u}\right] \tag{2.5}$$

$$P_u = \frac{2\pi\tau}{\sqrt{(K_pK_{cu})^2 - 1}} \tag{2.6}$$

As it is obvious from these two equations, the off-line Ziegler-Nichols tuning method is model-based which means that it requires the dynamics in the form of FOPTD.

This technique has several disadvantages. First of all, it can be time-consuming to find the best possible gains based on this tuning method, when the process is slow, or several trials are necessary. Secondly, this tuning rule is not applicable to integrating or open-loop unstable plants because these processes are usually unstable for big and small values of the gain [Seborg et al., 1989]. Finally, for the first order models without time delays, K_{cu} does not exist [Seborg et al., 1989].

IMC Method [Garcia and Morari, 1982]

Internal Model Control, or IMC, is a model-based design method. This method is effective when a reasonably precise model of the plant is available. One of the advantages of the IMC method is that it has flexibility for model uncertainty and tradeoffs between performance and robustness [Seborg et al., 1989]. The IMC controller is designed in two steps:

1. As it is shown in Equation (2.7), the model of the process shown by $\tilde{G}$ is divided into two parts: $\tilde{G_+}$ and $\tilde{G_-}$.

$$\tilde{G} = \tilde{G_+}\tilde{G_-} \tag{2.7}$$

   where $\tilde{G_+}$ includes any time delays and zeros in the right-half plane while it has a steady-state gain of one. This gain should be equal to one to ensure that $\tilde{G_+}$ and $\tilde{G_-}$ are two unique terms. Based on the definition of $\tilde{G_+}$, $\tilde{G_-}$ can be considered as the invertible part of the model of the system.

2. Design the controller based on Equation (2.8).

$$G_c^* = \frac{1}{\tilde{G_-}} f \tag{2.8}$$

   where f is a low-pass filter with a steady-state gain of one and is defined in Equation (2.9).

$$f = \frac{1}{(\tau_c s + 1)^r} \tag{2.9}$$

where τ_c is the desired closed-loop time constant. Also r is a positive integer which usually is set to be one.

As Equation (2.8) shows, the IMC controller is designed based on the invertible part of the dynamics of the plant. Hence, this controller is proved to be physically realizable and stable [Seborg et al., 1989]. The non-invertible $\tilde{G_+}$ limits the performance of any controller applied to this process. In the case of open-loop unstable processes, IMC should be modified, because the standard IMC design method is rooted in the pole-zero cancelation [Seborg et al., 1989]. τ_c is the key parameter in the IMC design method. As it can be concluded from Table 2.2, a higher τ_c leads to a lower K_c and a lower K_i. This results in a more conservative controller. The common guidelines on choosing the closed-loop time constant in the IMC methods for FOPTD models are shown in Equations (2.10) to (2.12):

$$\frac{\tau_c}{\theta} > 0.8 \quad \text{and} \quad \tau_c > 0.1\tau \tag{2.10}$$

$$\tau > \tau_c > \theta \tag{2.11}$$

$$\tau_c = \theta \tag{2.12}$$

Where τ and θ are the time constant and the time delay, respectively. For processes with a dominant time constant, τ_{dom}, guideline (2.11) can be used by substituting τ by τ_{dom} [Seborg et al., 1989].

Any IMC controller is shown to be equivalent to a standard feedback controller [Seborg et al., 1989]. Hence, the IMC method can be used to derive PI or PID controller settings for systems with different models. These tuning relations depend on the low-pass filter introduced in Equation 2.9, and the way the time-delay of the model is approximated.

The IMC-based tuning relations for FOPTD transfer functions are shown in Table 2.2, for a PI and a PID controller. Here r in 2.9 is considered to be one. In this thesis, IMC tuning rules are used to define six different IMC controllers for comparative studies. This idea is explained with more detail later in Section 4.1.2.

Table 2.2: Controller settings based on the IMC method

Model	K_cK_p	K_i	K_d
$\frac{Ke^{-\theta s}}{\tau s+1}$	$\frac{\tau}{\tau_c+\theta}$	$\frac{K_c}{\tau}$	—
$\frac{Ke^{-\theta s}}{\tau s+1}$	$\frac{\tau+\frac{\theta}{2}}{\tau_c+\frac{\theta}{2}}$	$\frac{K_c}{\tau+\frac{\theta}{2}}$	$\frac{K_c\tau\theta}{2\tau_c+\tau}$

2.2 Introduction to Reinforcement Learning

In the following sections the basic theories of Reinforcement Learning are described. The structure and the explanations are mostly based on "Reinforcement Learning: An Introduction", a book by Richard, S. Sutton, and Andrew G. Barto [Sutton and Barto, 1998], as the author believes this book is one of the best resources one can find in the philosophy and basics of Reinforcement Learning.

The amount and type of the information that is available from the environment distinguishes various learning schemes. In this way, three main categories are usually defined for learning methods: supervised learning, reinforcement learning (RL), and unsupervised learning.

Supervised learning is mainly about function approximation and the generalized mapping of inputs to outputs in a way that results in the best possible approximation of the output for the entire input space. Here, in addition to the input/output pairs of data, the desired behavior is explicitly mentioned.

In contrast to this, in unsupervised learning the only information provided by the environment is the data set. Here, no evaluation of the desired output is available. In this scheme a model is built by analyzing the data and is applied to future tasks.

The final category, which is of interest to us, sits between these two methods. In RL, not only the data is provided, but also, after the agent's interaction, an evaluative feedback from the environment is sent to the learning agent stating how good the output is. This evaluative feedback is different from the one in supervised learning because it does not mention explicitly the desired input/output pairs.

RL is inspired by psychological and animal learning theories [Fu and Anderson, 2006]. All people, more or less, apply RL in their lives, while they are discovering, for the first time, how to walk, ride a bike, ice skate, or train a pet. This Artificial

Intelligence method deals with the big picture of a goal-directed problem; therefore, it is particularly suitable for problems which include a long-term versus short-term cost trade-off; to have the optimal solution to this type of problems, it may sometimes be crucial to accept short-term costs in order to gain smaller costs in the future.

RL has been applied successfully to different problems, such as robot control and navigation [Connell and Mahadevan, 1993], elevator scheduling [Crites and Barto, 1996], telecommunications [Singh and Bertsekas, 1997], and logic games like poker, backgammon, and chess [Tesauro, 1995, 1994, Fogel et al., 2004].

In the next few sections, some common terms in Reinforcement Learning will be defined. This will be necessary to proceed to the description of different reinforcement learning methods.

Policy

The policy determines the behavior of the learner. For example, it can be a mapping from a state to an action the learner ought to take in that state. As Sutton and Barto say, "Policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior " [Sutton and Barto, 1998]. Policy in general can be either stochastic or deterministic.

Immediate Cost Function (reward)

This term in the reinforcement learning related literature has different names, such as immediate cost (or immediate reward), or reinforcement. In this thesis, the term immediate cost will be used for this definition.

In RL, at each state, each action that the agent may take results in an immediate reaction from the environment. This immediate reaction shows the immediate desirability of that action and is represented by a single number. The objective of reinforcement learning is to minimize the summation of the immediate costs it receives through time. Just like the policy, the immediate cost can be either stochastic or deterministic. Sutton and Barto compare this term to pleasure and pain for biological systems [Sutton and Barto, 1998].

In this thesis, immediate cost is defined as the squared error at each time step.

Cost-to-go Function

RL actions may affect, not only the immediate cost, but also the next state and all the consequent immediate costs. While the immediate cost shows the cost at the current stage, cost-to-go function indicates the long-term cost. The cost-to-go of a state is the total amount of immediate costs that is expected to be received by the learner in the future, if the process starts from that state. This term takes into account the states that are likely to follow and their costs. Sutton and Barto compare the cost-to-go function to the farsighted judgment of how happy or unhappy humans are with their environment being in its specific state [Sutton and Barto, 1998].

The aim of defining the cost-to-go function is to find a way to receive less cumulative immediate costs. In decision making, actions that will result in states with the lowest cost-to-go are preferred to the ones that lead to the lowest immediate cost.

The estimation of this cost-to-go function from the observations of the learner is the most important part in most of the RL methods. Many studies have been conducted on how to efficiently estimate the cost-to-go function [Samuel, 1959, Bellman and Dreyfus, 1959, Holland, 1986, Anderson, 1987, Christensen and Korf, 1986, Chapman and Kaelbling, 1991, Tan, 1991, Yee et al., 1990, Dietterich and Flann, 1995].
It should be noted that this term is also known as Value Function in the literature.

Model

A model is the estimate of the dynamics of the environment. Not all reinforcement learning methods require the model of the environment. In fact, RL methods can be divided into two groups: Model-based methods and model-free ones.

Markov Decision Process [Thompson, 1933, 1934, Robbins, 1952]

A Markov Decision Process, or MDP, is a tool to model a sequential decision-making problem [Szepesvri]. Most process control problems can be represented as

MDPs [Kumar, 1985]. MDPs are characterized by possessing the Markov property. Possessing the Markov property basically means that the current state of the system is sufficient for predicting its upcoming state. In this way, the current state of the system can be considered as a compressed history of its past.

The Markov property is important in reinforcement learning methods because decisions and cost-to-go functions are presumed to depend just on the current state of the process. In this way, satisfying the Markov property for a process enables us to predict the next state and the expected immediate cost by just knowing the current state and RL action [Sutton and Barto, 1998, Kaelbling et al., 1996]. Accordingly, in reinforcement learning methods, the state of the process should be chosen sufficiently informative to result in an MDP [Sutton and Barto, 1998].

In many real world environments, It is not possible for the RL agent to have a complete observation of the state of the environment, though complete observability is vital for learning methods based on MDPs. In the case of noisy or incomplete observations of the state, the process is called a *Partially Observable Markov Decision Process*, or POMDP. There are different methods that deal with POMDPs of which the simplest one is just to ignore the incompleteness of the observations, accept them as the states of the system, and implement the same learning procedures as the MDP case. This approach, called *state-free deterministic policies* in RL literature, may yield credible results in some cases, but there are no guarantees for that [Kaelbling et al., 1996]. There are some other methods that deal with POMDPs, such as *state-free stochastic policies* and *policies with internal states*; the former, considers some mappings from observations to probability distribution over actions. The latter, uses the history of the previous observations and actions to disambiguate the current state.

Standard Structure of Reinforcement Learning

The standard structure of reinforcement learning is depicted in Figure 2.1. As this figure shows, at each step of interaction, the reinforcement learning agent receives two signals from the environment. The first signal is represented by x, which contains information about the current state of the environment; the second is the

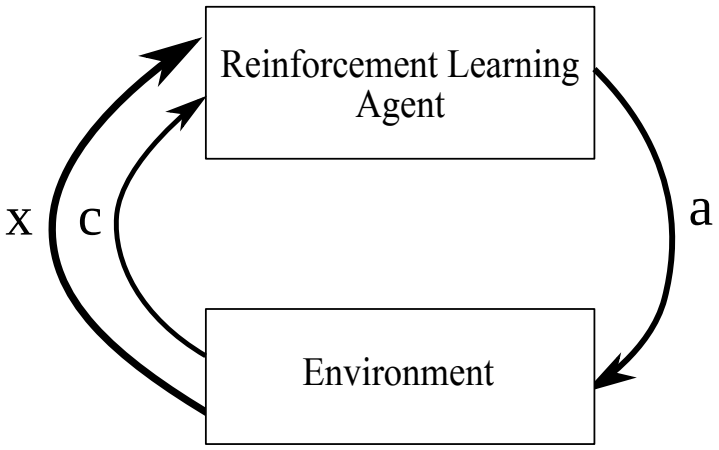

Figure 2.1: Reinforcement learning standard structure

feedback of the plant to the previous action of the agent. This feedback is called the immediate cost and is shown by c. Based on the current state of the system, the RL agent chooses an action a, and implements it on the environment; as a response to the action of the agent, the state of the environment changes and a new immediate cost signal gets generated. The sole purpose of the agent during this process is to find a policy π, that maps states to actions in a way that minimizes some long-run measurement of the immediate cost signal [Kaelbling et al., 1996]. Accordingly, the standard structure of the reinforcement learning consists of:

a set of states shown by $\mathbf{S}$

a set of RL actions shown by $\mathbf{A}$

a set of scalar immediate cost signal shown by $\mathbf{R}$

The objective function that the RL agent tries to minimize can have different formats. The most common objective functions are as follows:

The *finite horizon* objective function which optimizes at each step of time the expected summation of the immediate costs for the next h step:

$$J(x_k) = E\Big\{ \sum_{i=0}^{h} c_{k+i+1} \Big\} \tag{2.13}$$

The *infinite horizon discounted* objective function takes into account the long-term immediate costs received, but the later the immediate cost is gained, the lower its

importance is. Here discount factor γ has the role of geometrically discounting the importance of the future immediate costs and bounding the infinite sum.

$$J(x_k) = E\Big\{\sum_{i=0}^{\infty} \gamma^k c_{k+i+1}\Big\} \qquad \text{Where } 0 \leq \gamma < 1 \tag{2.14}$$

Finally, the *average reward* objective function is defined to minimize the long-term average of the summation of the immediate costs:

$$J(x_k) = \lim_{h\to\infty} E\Big\{\frac{1}{h}\sum_{i=0}^{h} c_{k+i+1}\Big\} \tag{2.15}$$

Process control tasks are usually continuous (and not episodic), as a result of this characteristics, objective function introduced in Equations (2.13) is considered improper for our proposed RL-based PI-controller. In addition, the average of reward objective function introduced in Equation (2.15) gives the same weighting factors to the immediate costs received through time. This is while error at the beginning of set-point change is bigger and hence is more important to us. Also, using Equation (2.15) may result in a time-consuming learning process. To give a higher weighting factor to error at the beginning of set-point change, the discounted infinite horizon objective function introduced in Equation (2.14) is used as the objective function of the proposed approach in this thesis.

Almost all reinforcement learning methods are mainly constructed of two parts: prediction and control. In the prediction part, the aim is to estimate the cost-to-go of a specific policy as precisely as possible. This part is usually considered as the most important step in reinforcement learning. The control part is to improve that specific policy locally with respect to the current cost-to-go function. This idea is shown in Figure 2.2.

2.2.1 Dynamic Programming [Bellman, 1957]

Dynamic Programming, or DP, is one of the main approaches in solving MDP problems. DP was introduced by Bellman [Bellman, 1957]. In this section, this concept is explained for finite MDP Problems; However, DP methods can also be applied to problems with continuous state and action sets, being modified with the

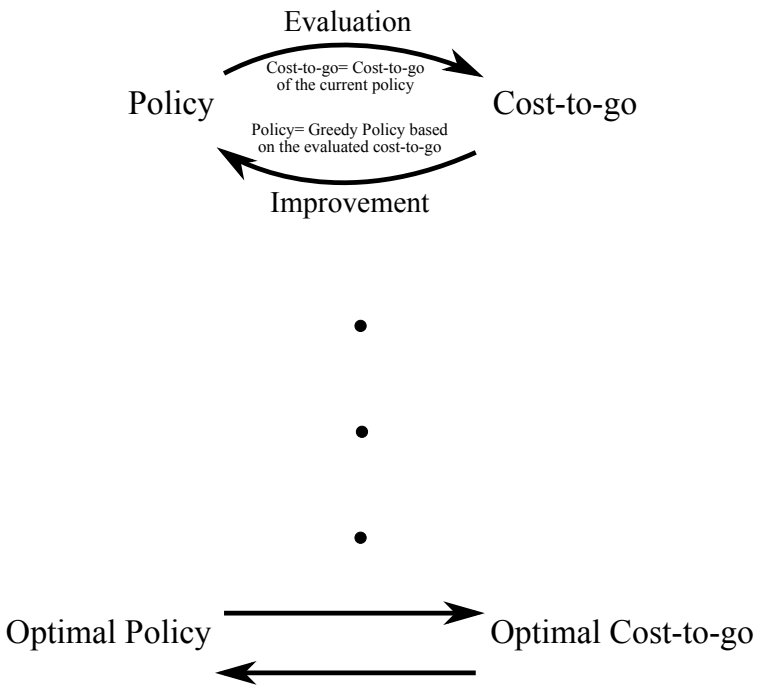

Figure 2.2: The interaction of the prediction and the control part until they result in the optimal cost-to-go and policy

generalization methods described in section 2.2.3. Here, the state and action sets are shown by $\mathbf{S}$ and $\mathbf{A}(\mathbf{x})$ and the model of the environment is represented by the transition probability and the expected immediate cost defined in Equations (2.16) and (2.17), respectively. These terms are assumed to be precisely known for DP methods.

$$\mathfrak{P}^{a}_{xx'} = Pr\{x_{k+1} = x^{'} | x_k = x, a_k = a\} \tag{2.16}$$

$$\mathfrak{C}^{a}_{xx'} = E\{c_{k+1} | a_k = a, x_k = x, x_{k+1} = x^{'}\} \tag{2.17}$$

Dynamic programming is a model based method that is used to calculate the optimal solution of MDP problems. Given the perfect dynamics of the environment, DP computes the solution of the Bellman optimality equation shown in Equation (2.18), for the optimal cost-to-go, and Equation (2.19), for the optimal action-cost-to-go.

$$\begin{aligned} J^*(x) &= \min_a E\{c_{k+1} + \gamma J^*(x_{k+1}) | x_k = x, a_k = a\} \\ &= \min_a \sum_{x^{'}} \mathfrak{P}^{a}_{xx'} [\mathfrak{C}^{a}_{xx'} + \gamma J^*(x^{'})] \end{aligned} \tag{2.18}$$

$$
\begin{aligned}
Q^*(x,a) &= E\{c_{k+1} + \gamma \min_{a'} Q^*(x_{k+1}, a') | x_k = x, a_k = a\} \\
&= \sum_{x'} \mathfrak{P}^a_{xx'} [\mathfrak{C}^a_{xx'} + \gamma \min_{a'} Q^*(x', a')] \qquad (2.19)
\end{aligned}
$$

All of these Equations hold for all $x \in \mathbf{S}$, $a \in \mathbf{A}(\mathbf{x})$, and $x' \in \mathbf{S}^+$; where $\mathbf{S}^+$ is $\mathbf{S}$ plus a terminal state for episodic cases. The assumption of a precise model and the great amount of computation required, limits the application of DP methods.

Policy Evaluation

Policy Evaluation deals with the estimation of the cost-to-go or the action-cost-to-go of an existing policy, π. The goal here is to have an approximation of the cost-to-go function as precise as possible, but no change will be applied to the policy in order to improve it. The cost-to-go function is shown in Equation (2.20)

$$
J^\pi(x) = \sum_a \pi(x,a) \sum_{x'} \mathfrak{P}^a_{xx'} [\mathfrak{C}^a_{xx'} + \gamma J^\pi(x')] \qquad (2.20)
$$

where $\pi(x,a)$ is the probability at which action a, at state x, under policy π is chosen.

In dynamic programming methods, the transition probability and the expected immediate cost are assumed to be known. In this way, Equation (2.20) can be considered as a set of $|\mathbf{S}|$ simultaneous linear algebraic equations in which $J^\pi(x)$, $x \in \mathbf{S}$ are their $|\mathbf{S}|$ unknown variables. Solving Equation (2.20) as a system of equations is simple but time consuming or even practically impossible in large state sets. So to find the solution of Equation (2.20), iterative methods are preferred. In *iterative policy evaluation*, firstly a random value will be chosen as the initial value of the cost-to-go, J_0; then, the Bellman equation is applied to update the cost-to-go. The Bellman equation as an update rule is shown in Equation (2.21).

$$
\begin{aligned}
J_{i+1}(x) &= E_\pi\{c_{k+1} + \gamma J_i(x_{k+1}) | x_k = x\} \\
&= \sum_a \pi(x,a) \sum_{x'} \mathfrak{P}^a_{xx'} [\mathfrak{C}^a_{xx'} + \gamma J_i(x')] \qquad (2.21)
\end{aligned}
$$

Here i shows the iteration number. As $i \to \infty$, J_i converges to J^π for all $x \in \mathbf{S}$. To find J^π for the entire state set, the whole state space should be visited repeatedly. Iterative policy evaluation is summarized in Algorithm 1.

Algorithm 1 Policy Evaluation [Sutton and Barto, 1998]

1: Define the policy which is going to be evaluated
2: **for** each $x \in \mathbf{S}$ **do**
3: Initialize $J(x)$
4: **end for**
5: $\Delta \leftarrow \infty$
6: **repeat**
7: **for** each $x \in \mathbf{S}$ **do**
8: $\nu \longleftarrow J^{\pi}(x)$
9: $J^{\pi}(x) \longleftarrow \sum_a \pi(x,a) \sum_{x'} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J^{\pi}(x^{'})]$
10: $\mathbf{\Delta} \leftarrow \max(\mathbf{\Delta}, |\nu - J(x)|)$
11: **end for**
12: **until** Δ is smaller than a positive number

Policy Improvement [Bellman, 1957, Howard, 1960, Watkins, 1989]

Knowing the cost-to-go function of a policy π, at state x, determines how good is to follow that policy from that state; however it does not determine if π is the best possible policy to follow from x. If Equation (2.22) holds for policies π and $\pi^{'}$, $\pi^{'}$ is either as good as, or better than π.

$$J^{\pi^{'}}(x) \leq J^{\pi}(x) \tag{2.22}$$

To further improve an existing policy, the possible changes to the policy (to the action suggested by the policy), should be evaluated. This is easily possible based on policy evaluation scheme explained in Section 2.2.1. Evaluating all the changes, at all the possible states and choosing at each state the action that has a lower action-cost-to-go under the existing policy is called *policy improvement*. In fact, this is where action-cost-to-go functions play an important role. This idea is summed up in Equation (2.23):

$$\begin{aligned} \pi^{'} &= arg\min_a Q^{\pi}(x,a) \\ &= arg\min_a E\{c_{k+1} + \gamma J^{\pi}(x_{k+1}) | x_k = x, a_k = a\} \\ &= arg\min_a \sum_{x^{'}} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J^{\pi}(x^{'})] \end{aligned} \tag{2.23}$$

Policy improvement will result in a strictly better policy except when the existing policy is the optimal one [Sutton and Barto, 1998].

Policy Iteration [Bellman, 1957, Howard, 1960]

Policy iteration is nothing but applying policy evaluation and policy improvement repeatedly after each other to yield the optimal policy. This idea is illustrated below:

$$\pi_0 \xrightarrow{\text{Evaluation}} J^{\pi_0} \xrightarrow{\text{Improvement}} \pi_1 \xrightarrow{\text{Evaluation}} \cdots \xrightarrow{\text{Improvement}} \pi^* \xrightarrow{\text{Evaluation}} J^*$$

Policy iteration is shown in Algorithm 2.

Algorithm 2 Policy Iteration [Sutton and Barto, 1998]

for each $x \in \mathbf{S}$ **do**
  Initialize $J(x)$ and the policy
end for
$\Delta \leftarrow \infty$ {Policy Evaluation}
repeat
  for each $x \in \mathbf{S}$ **do**
    $\nu \longleftarrow J^{\pi}(x)$
    $J^{\pi}(x) \longleftarrow \sum_a \pi(x,a) \sum_{x'} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J^{\pi}(x^{'})]$
    $\mathbf{\Delta} \leftarrow \max(\mathbf{\Delta}, |\nu - J(x)|)$
  end for
until Δ is smaller than a positive number
$Stop \leftarrow true$ {Policy Improvement}
for each $x \in \mathbf{S}$ **do**
  $b \leftarrow \pi(x)$
  $\pi(x) \leftarrow arg\min_a \sum_{x'} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J(x^{'})]$
  if b and $\pi(x)$ are not the same, then set Stop as False
end for
if Stop is True **then**
  Stop
else
  Go back to Policy Evaluation
end if

Value Iteration

To overcome the computational cost of the policy evaluation in policy iteration, *Value Iteration* method is introduced. Policy evaluation can be computationally expensive because it requires multiple sweeps of the state space. This is while value iteration truncates the evaluation step right after the first sweep. Although the policy evaluation process is not performed completely, the convergence to the

optimal policy is still guaranteed. Value iteration combines the policy improvement and the trimmed policy evaluation as Equation (2.24):

$$\begin{aligned} J_{i+1}(x) &= \min_a E\{c_{k+1} + \gamma J_i(x_{k+1}) | x_k = x, a_k = a\} \\ &= \min_a \sum_{x'} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J_i(x')] \end{aligned} \tag{2.24}$$

In fact, value iteration is Bellman's optimality Equation (2.18) modified to a back up rule. The value iteration is shown in Algorithm 3. Here $\pi(x)$ is the learned policy.

Algorithm 3 Value Iteration [Sutton and Barto, 1998]

1: **for** each $x \in \mathbf{S}$ **do**
2: Initialize $J(x)$
3: **end for**
4: **repeat**
5: **for** each $x \in \mathbf{S}$ **do**
6: J(x) $\longleftarrow \min_a \sum_{x'} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J(x')]$
7: **end for**
8: **until** policy is good enough
9: $\pi(x) = \arg\min_a \sum_{x'} \mathfrak{P}^a_{xx'}[\mathfrak{C}^a_{xx'} + \gamma J(x')]$

In reinforcement learning literature, DP methods are also known as model-based RL methods. In the few next sections some model-free RL methods are introduced.

2.2.2 Temporal-Difference Learning [Samuel, 1959, Klopf, 1972]

Temporal-difference or TD methods are used to solve the prediction problem. The most important advantage of TD methods is that they do not require the exact dynamics of the system; instead, they are based on experiences from the environment. In these methods, different states are visited, different actions are implemented, and their immediate cost is observed; then, based on these interactions, the prediction of the cost-to-go and the policy are updated. Here, the update rule for the cost-to-go is as follows:

$$J(x_k) \longleftarrow J(x_k) + \alpha[c_{k+1} + \gamma J(x_{k+1}) - J(x_k)] \tag{2.25}$$

As the target value in Equation (2.25) itself is an estimation, TD is considered as a *bootstrapping* method. Equation (2.26) indicates how this target has been derived:

$$\begin{aligned} J^{\pi}(x_k) &= E_{\pi}\Big\{\sum_{j=0}^{\infty}\gamma^j c_{k+j+1}|x_k = x\Big\} \\ &= E_{\pi}\Big\{c_{k+1} + \gamma\sum_{j=0}^{\infty}\gamma^j c_{k+j+2}|x_k = x\Big\} \\ &= E_{\pi}\Big\{c_{k+1} + \gamma J^{\pi}(x_{k+1})|x_k = x\Big\} \end{aligned} \tag{2.26}$$

Based on Equation (2.25), the one step TD error is introduced:

$$\delta_k = c_{k+1} + \gamma J(x_{k+1}) - J(x_k) \tag{2.27}$$

TD methods update their estimation of cost-to-go at each step, which makes them suitable for online implementations.

Sarsa [Rummery and Niranjan, 1994]

TD methods are just used to predict the cost-to-go of a fixed policy. In other words they are applied in the case of the evaluation of a specific policy. On the other hand, control concepts are applied for further policy improvement which is usually of interest. For the case of policy improvement, action-cost-to-go function, $Q(x_k, a_k)$, is learned rather than cost-to-go function itself, $J(x_k)$. In this way the update rule will be changed from Equation (2.25) to (2.28):

$$Q(x_k, a_k) \longleftarrow Q(x_k, a_k) + \alpha[c_{k+1} + \gamma Q(x_{k+1}, a_{k+1}) - Q(x_k, a_k)] \tag{2.28}$$

Here, $Q(x_k, a_k)$ is the action-cost-to-go which is defined in Equation (2.29). As this equation shows, action-cost-to-go is the expected weighted summation of the immediate cost received if action a is implemented at state x.

$$Q(x_k, a_k) = E_{\pi}\Big\{\sum_{j=0}^{\infty}\gamma^j c_{k+j+1}|x_k = x\&a_k = a\Big\} \tag{2.29}$$

This algorithm is called *Sarsa*, by Sutton [Sutton, 1996]. Sarsa stands for the sequence of State, Action, Reward, State, and Action happening consequently in this algorithm. In this way, based on the current state, the current action is chosen

and implemented. After this, the reward (immediate cost) is received from the environment and the next state is observed. Finally the next action is chosen based on the observed next state. This algorithm is described next as Algorithm 4.

When the policy is ϵ-greedy (ϵ-greedy policies are defined later in Equation 2.47),

Algorithm 4 Sarsa [Sutton and Barto, 1998]

for each $x \in \mathbf{S}$ and $a \in \mathbf{A}$ **do**
  Initialize Action-cost-to-go function
end for
for each run **do**
  Initialize the first state, x
  At state x choose action a that minimizes the action-cost-to-go function
  With a small probability, ϵ, choose a random action and overwrite a with that (Policy: ϵ-greedy)
  for each step up to the end of run **do**
    Implement action a and observe the consequent immediate cost c, and the next state $x^{'}$
    At state $x^{'}$ choose action $a^{'}$ that minimizes the action-cost-to-go function
    With a small probability ϵ, choose a random action and overwrite $a^{'}$ with that (Policy: ϵ-greedy)
    $Q(x_k, a_k) \longleftarrow Q(x_k, a_k) + \alpha[c_{k+1} + \gamma Q(x_{k+1}, a_{k+1}) - Q(x_k, a_k)]$
    $x \longleftarrow x^{'}$
    $a \longleftarrow a^{'}$
  end for
end for

as long as all the state-action pairs are swiped for an infinite number of times, and ϵ converges in the limit to zero, Sarsa is shown to converge to the optimal policy and action-cost-to-go with probability of 1 [Sutton and Barto, 1998]. The very common format for ϵ, to satisfy the convergence to zero in the limit, is $\epsilon = 1/k$, where k is the time index for discrete systems [Sutton and Barto, 1998].

Sarsa is chosen as the RL method of choice in this thesis because of three reasons: First, we are interested in testing a model-free RL method. Second, value-function-based RL methods are rarely used in the tuning of PI-controllers. Third, Sarsa is easy to implement and understand and provided satisfactory results in the preliminary stages of this research project.

2.2.3 Generalization

In the case that the state or the action set is not finite, visiting all the possible state-action pairs is not possible; so it does not make sense to store a look-up table of cost-to-go over the states (or the state-action pairs). Consequently, it is necessary to generalize from the states that are learned (or state-action pairs), to the regions that have not been visited. Also, in discrete cases, where the state or action has many components, or the state or action set is huge, generalization can be used to overcome the curse of dimensionality associated with the huge amount of memory, data, and computation that is required.

The problem of learning in large spaces can be handled by generalization techniques; which efficiently store the information and transfer it among similar states and actions. In these techniques, approximated cost-to-go function at each time step is represented as a parameterized function with the vector of parameters usually shown by $\overrightarrow{\theta_t}$. It is this vector of parameters that is updated after each interaction. The number of parameters normally will be smaller than the number of states (state-action pairs); therefore, changing one parameter affects the estimated cost-to-go of many states [Sutton and Barto, 1998].

Most of the generalization schemes try to minimize the mean-squared error (MSE) over a distribution of the inputs, which is shown by P. In our case, the MSE for an estimation of the cost-to-go J_k, with the parameter vector of $\overrightarrow{\theta}_k$ results in Equation (2.30) as shown below.

$$MSE(\overrightarrow{\theta_k}) = \sum P(x)[J^{\pi}(x_k) - J_k(x_k)]^2 \tag{2.30}$$

The distribution P in Equation (2.30) is to weight the errors of the cost-to-go of different states. This is mainly because it is usually impossible to decrease the error to zero for all the states [Sutton and Barto, 1998]. In other words, better estimation of cost-to-go at some states will be achieved at the price of worse approximation at others. If an identical error over the entire state (state-action) space is preferred, then the function approximator should be trained with a uniform distribution of samples over the entire state (state-action). The goal of a reinforcement learning method in terms of MSE is to find the global optimum parameter vector, shown by

$\overrightarrow{\theta}^*$ for which Equation (2.31) holds for all $\overrightarrow{\theta}^*$:

$$MSE(\overrightarrow{\theta}^*) \leq MSE(\overrightarrow{\theta}_k) \tag{2.31}$$

Satisfying this equation is sometimes possible for simple function approximators like the linear ones; but it does not hold for complex function approximators like neural networks or decision trees. In the case of complicated function approximation, instead of a global optimum, a local optimum may be achieved [Sutton and Barto, 1998].

Gradient-Descent Methods [Sutton, 1988]

Gradient-descent methods are considered as the most common function approximation methods in Reinforcement Learning [Sutton and Barto, 1998]. Here $\overrightarrow{\theta}_k$ has a fixed number of components as shown in Equation (2.32)

$$\overrightarrow{\theta_k} = [\theta_k(1), \theta_k(2), \theta_k(3), \ldots, \theta_k(n)] \tag{2.32}$$

The objective here is to minimize Equation (2.30). For this to happen, after each interaction, the parameter vector is shifted by a small amount in the opposite direction of the gradient of the error. This direction is chosen because along it the error would reduce the most at that sample. Equations (2.33) and (2.34) show this idea.

$$\overrightarrow{\theta}_{k+1} = \overrightarrow{\theta}_k - \frac{1}{2}\alpha\nabla_{\overrightarrow{\theta}_k}[J^\pi(x_k) - J_k(x_k)]^2 \tag{2.33}$$

$$\overrightarrow{\theta}_{k+1} = \overrightarrow{\theta}_k + \alpha[J^\pi(x_k) - J_k(x_k)]\nabla_{\overrightarrow{\theta}_k} J_k(x_k) \tag{2.34}$$

Where $\nabla_{\overrightarrow{\theta}_k} J_k(x_k)$ is defined in Equation (2.35)

$$\nabla_{\overrightarrow{\theta}_k} J_t(x_k) = \left[\frac{\partial J_k(x_k)}{\partial\theta_k(1)}, \frac{\partial J_k(x_k)}{\partial\theta_k(2)}, \ldots, \frac{\partial J_k(x_k)}{\partial\theta_k(n)}\right] \tag{2.35}$$

Here, α is known as the learning rate and should be positive, less than one, and decreasing over time. This is a necessary condition for the convergence of the gradient-descent method. To be more precise, if the learning rate is reduced in a way that it meets the standard stochastic approximation condition in Equation (2.36),

then the convergence of this method to a local optimum is certain [Sutton and Barto, 1998].

$$\sum_{k=1}^{\infty} \alpha_k = \infty \qquad and \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty \tag{2.36}$$

In Equation (2.36), the first condition is to assure that the learning steps are big enough to eventually overcome initial conditions or random fluctuations [Sutton and Barto, 1998]. The second condition is to guarantee convergence [Sutton and Barto, 1998].

The reason for not eliminating the error totally on each interaction is that our main objective is not to find a cost-to-go function for which its estimation error on all the visited states is zero. instead, we are just seeking an estimation that balances the error over the state space. If at each step, we change the parameters to eliminate the error completely, our procedure may not result in that balance [Sutton and Barto, 1998].

In Equation (2.34), the true value of $J^{\pi}(x_k)$ is usually unknown; instead, an estimation of this value, ν_k, can be used to approximate Equation (2.34) by Equation (2.37).

In addition to Equation (2.36), Equation (2.38) should hold for the convergence of $\overrightarrow{\theta}_k$ to a local optimum [Sutton and Barto, 1998]. This equation basically means that ν_k should be an unbiased estimation of $J^{\pi}(x_k)$.

$$\overrightarrow{\theta}_{k+1} = \overrightarrow{\theta}_k + \alpha[\nu_k - J_k(x_k)]\nabla_{\overrightarrow{\theta}_k} J_k(x_k) \tag{2.37}$$

$$E\{\nu_k\} = J^{\pi}(x_k) \quad \text{for each } k \tag{2.38}$$

Where E is the Expectation operator.

Linear Methods

As explained in Section 2.2.3, linear function approximators have the advantage of resulting in the global optimum parameter vector. In this section the combination of gradient descent methods with linear function approximation is described. Here the approximate cost-to-go function, $J(x_k)$, is a linear function of $\overrightarrow{\theta}_k$, the vector of parameters. The dependency of the cost-to-go to the state is shown by a column

vector of features with the same number of elements as the parameter vector. This feature column vector is shown in Equation (2.39):

$$\overrightarrow{\phi}_x = [\phi_x(1), \phi_x(2), \ldots, \phi_x(n)]^T \tag{2.39}$$

Where n is the number of the parameters in $\overrightarrow{\theta}_k$. As shown by a subscript in this equation, features are a function of the state of the system.

In this way, the cost-to-go function will be approximated by the following equation:

$$J_k(x) = \overrightarrow{\theta}_k \overrightarrow{\phi}_{x_k} = \sum_{i=1}^{n} \theta_k(i)\phi_{x_k}(i) \tag{2.40}$$

Substituting Equation (2.40) in Equation (2.37) results in Equation (2.41):

$$\overrightarrow{\theta}_{k+1} = \overrightarrow{\theta}_k + \alpha[\nu_k - J_t(x_k)] \overrightarrow{\phi}^T_{x_k} \tag{2.41}$$

This equation is much simpler than the previous forms, mainly because, in this case the gradient of cost-to-function with respect to the parameter vector will be as simple as Equation (2.42):

$$\nabla_{\overrightarrow{\theta}_k} J_k(x_k) = \overrightarrow{\phi}^T_{x_k} \tag{2.42}$$

Different functions can be used to make features from the states. Coarse Coding, Tile Coding, and Radial Basis Functions, (RBF), are among the most popular ones [Sutton and Barto, 1998]. The representation of states in terms of the features has a great impact on the efficiency of the learning procedure in terms of the required amount of data and computation. So, selecting the proper features for learning is a vital issue [Sutton and Barto, 1998].

Quadratic Features

Quadratic approximation of the cost-to-go is also a linear function approximation method. One should note that the linear term in "linear function approximation" refers to the linearity in terms of the parameters vector, and not the features one. Here, there exist a linear vector of parameters, and another vector of features, which is the quadratic format of the state. This concept is shown in 2.43:

$$\phi(x) = x_k \otimes x_k \tag{2.43}$$

where $\otimes$ is the element-wise crossing operator. In this way, the estimated cost-to-go is defined as Equation 2.44:

$$J(x) = \overrightarrow{\theta}\,\phi_x \tag{2.44}$$

where θ is the vector of parameters. In this thesis, quadratic function approximation is used around the origin as a means to provide stability around origin. This idea is explained in detail in Function Approximation part of Section 4.1.3.

k-nearest neighborhood function approximation [Smart and Kaelbling, 2000]

k-nearest neighborhood, or $k - NN$ is a type of lazy-learning method in which the function for a specific point is approximated locally based on the values of its k closest neighbors. k is a positive number and its value increases as the number of the data points go higher. Usually a contribution factor is added to the function approximation, so that the nearest neighbor has a more significant impact on the value of the function, than the further ones. The most common contribution factor is $\frac{1}{d}$, in which d represents the distance between the current point, and the neighbor. $k - NN$ can be considered as the generalized linear interpolation.
$k - NN$ can be computationally intensive, as the size of the stored values grows by time. To overcome this problem, different nearest neighbor search algorithms have been proposed, which decrease the number of required distance evaluations.

Function Approximation for control problems

To apply function approximation to model-free control tasks, it is the action-cost-to-go, $Q(x, a)$ defined in Equation (2.29) that should be approximated by generalization. For the linear function approximation case, the action-cost-to-go is estimated through Equation (2.45). For discrete actions, it is the vector of parameters in this equation that shows the dependency of the action-cost-to-go on the implemented action. In this way, to each possible action, one vector of parameters is assigned.

$$Q(x, a) = \overrightarrow{\theta_a}\phi_x \tag{2.45}$$

The general gradient-descent update rule for action-cost-to-go function is shown in Equation (2.46):

$$\overrightarrow{\theta}_{a_k,k+1} = \overrightarrow{\theta}_{a_k,k} + \alpha[\nu_k - Q(x_k, a_k)]\nabla_{\overrightarrow{\theta}_{a_k,k}} Q(x_k, a_k) \tag{2.46}$$

The target (ν_k) here is an approximation of $Q^{\pi}(x_k, a_k)$, which can be any of the back up values mentioned previously for action-cost-to-go [Sutton and Barto, 1998].

In this thesis, the state space is continuous. Hence, function approximation is used for generalization of the action-cost-to-go over the state space. Quadratic function approximation is used for a region close to origin in which the process is assumed to be linear. On the other hand, k-nearest neighborhood is applied for the rest of the state space. This procedure is explained with more details in Function Approximation part of Section 4.1.3.

2.2.4 Exploration versus Exploitation

As mentioned in previous sections, the evaluative feedback of reinforcement learning specifies how good the action taken is, but it does not give any information about the best possible actions. At each point of time, the greedy action suggested by the policy of the RL agent is the best known action so far, based on the learning, but this does not mean that there is no better action that may result in a much better performance. As a result of this characteristic, specially for a non-stationary environment, continuous exploration by a trial and error search in the action set is crucial to find the best actions. The exploratory actions may decrease the performance of the system, as they may result in a higher cost-to-go rather than a smaller one. Because of this, the rate of exploration should be adjusted to keep the overall performance of the task satisfactory. For this, most of the time the greedy action should be implemented. This challenge is known as the trade-off between exploration and exploitation [Sutton and Barto, 1998]. The ϵ-greedy methods are one of the most common schemes in this case [Sutton and Barto, 1998]. These methods choose a random action at a small fraction of time, while the rest of the time the greedy action is implemented. This is shown in Equation (2.47)

$$\Pi(x, a) = \begin{cases} argmin_a Q(x, a) & \text{With probability of } 1 - \epsilon \\ a^{random} \in A(x) & \text{With probability of } \epsilon \end{cases} \tag{2.47}$$

where $\Pi(x, a)$ is the policy at state x.

An ϵ-greedy policy is used here as the policy that Sarsa is aiming to learn.

2.2.5 Summary

In this section, the prerequisite background of this research thesis is provided. IMC tuning rules introduced in Section 2.1.1 are used to define IMC-based-tuned PI-controllers to compare our proposed approach with. Sarsa defined in Section 2.2.2 in combination with quadratic and k-nearest neighborhood function approximation is used as the main RL method applied in this thesis. The policy of the RL agent here is chosen to be ϵ-greedy. The immediate cost obtained as reinforcement from the process is considered to be the squared output error, while the objective function of the RL agent is set as the discounted infinite horizon summation of the immediate costs received through time.

The following chapter provides a literature review on the tuning of PI-controllers based on different RL methods and the gain scheduling of PI-controllers. The literature review chapter is concluded with a paragraph on the comparison of our proposed approach with the reviewed work from literature.

Chapter 3

Literature Review

One of the reinforcement learning methods documented in the literature for the tuning of PI or PID-controllers is the Continuous Action Reinforcement Learning Automata, or CARLA, which was first introduced by Howell *et.al.* [Howell et al., 1997]. This learning scheme has a set of probability densities used to define the action set. CARLA chooses the RL actions in a stochastic trial and error process, aiming to minimize a performance objective function, which usually is a function of error over time. Howell *et.al* applies CARLA to the control of a semi-active suspension system [Howell et al., 1997]. Here, the control goal is defined as minimization of the mean squared acceleration of the vehicle body. The advantages of CARLA are mentioned in its ability to operate over a bounded continuous action sets, being robust to high levels of noise, and being suited to function in a parallel computing environment. The cost-to-go function in this paper is defined as the integral of the time-weighted squared error over a period of time (T=5s) which is shown in Equation (3.1).

$$J = \int_{t=0}^{t=T} \tau(e_t)^2 dt \tag{3.1}$$

CARLA is also applied to the control of engine idle-speed, both in a simulation environment, and in practice [Howell and Best, 2000, Howell et al., 2000]. These papers report a significant improvement in control performance, compared to that of Ziegler-Nichols tuning method.

CARLA is used as an optimization approach for optimum tuning of PID controllers in an automatic voltage regulator of a synchronous generator [Kashki et al., 2008]. This paper explains CARLA in detail and compares its performance

with that of Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). The simulation results provided in this paper show a better efficiency and performance for CARLA as opposed to the other two methods.

Gain scheduling is another concept that is comparable with what we perform in this thesis. In gain scheduling, a global nonlinear controller is obtained by interpolation or selection of local linear controllers around various operating conditions [Shamma]. Here, the controllers are chosen based on some scheduling variables, which are usually process outputs or inputs [Lee and Lee, 2008]. Static gain scheduling schemes just use the current value of the scheduling variables to make decisions. This may result in slow adaptation when the operating condition is changed [Shamma and Athans, 1992]. Dynamic scheduling algorithms are not as common as the static ones. These methods are integrated with a sort of a dynamic scheduling variable, such as the history of the feed-back error [Jutan, 1989]. The dynamic gain scheduling methods are either controller-specific or hard to implement on non-minimum phase processes [Lee and Lee, 2008]. In addition, gain scheduling techniques have fundamental limitations, such as the requirement of varying slowly for the scheduling variable [Shamma and Athans, 1992].

Value function (cost-to-go) based approaches are another way of applying reinforcement learning in the tuning of PI-controllers. Lee and Lee use the Approximate Dynamic Programming, or ADP, for some gain scheduling case of studies [Lee and Lee, 2008]. Their method at each state, based on the evaluation of the cost-to-go function of the next state, chooses the best control action among the control signals suggested by the available controllers. The most important advantage of this technique is that it overcomes the curse of dimensionality by the followings: limiting the action space to the control parameters of some specific controllers, bounding the state space to those visited during a closed-loop simulation, and solving the Bellman optimality equation in an approximate manner. The function approximator used by Lee and Lee [Lee and Lee, 2008] is the k-nearest neighbor interpolation. In this paper both model-based and model-free approaches are studied by simulating three different case studies.

Lee and lee [Lee and Lee, 2006] explain approximate dynamic programming in

detail. In this paper, ADP is applied to process control and scheduling as a tool to find the solution of the MDPs. Here, as a result of dealing with a large state space, dynamic programming is approximated to decrease the amount of computation and data storage and make this procedure more practical for industrial scenarios. Lee and Lee [Lee and Lee, 2006] also studies the modification of the cost-to-go function in a way that the system avoids visiting unknown areas of the state space. This is done by adding a reverse function of the density of the data over the state space as a penalty term to the cost-to-go function. While modifying the cost-to-go function in this way is helpful in preventing the system from encountering safety issues, it also may forbid the system from finding the optimal solution as a result of bounding the regions of the state space which is going to be visited regularly. This disadvantage of the penalty term may not be of much importance because using approximate dynamic programming in the first place will result in a non-optimal solution. Lee and Lee [Lee and Lee, 2006] study a paper machine headbox control problem. In this study, the regulation performance of some linear model predictive controllers with different input weights is compared to that of an ADP method. Providing the state trajectory Lee and Lee conclude that the ADP method outperforms the other linear model predictive controllers by switching between their control policies through time. Here the numerical evaluations of the same case are also presented.

Anderson *et.al.* studies the combination of reinforcement learning with feedback controllers by simulating the heating coil of the heating system of building [Anderson et al., 1997]. In this paper, three different cases are studied. In the first case, a feed-forward neural network is trained to reduce the error between the temperature of the output air and the desired set-point, defining various objective function. In the second case a neural network is used to predict the steady state output of the PI-controller for the heating coil model. The performance of this neural network on its own is mentioned to be poor compared to the PI-controller because of not having the proportional term. Combining the trained neural network with a proportional controller with optimum gain for that specific case, results in a far better control performance in comparison to that of the PI-controller. Thirdly, a table look-up Q-learning algorithm is used to train the reinforcement learning agent

in combination with a PI-controller. In the third case, the output of the RL agent is added to the output of the PI-controller. The immediate cost is also defined as the squared error plus a term proportional to the square of the change in the action from the previous time step to the current one. Anderson *et.al* include the action change term in the immediate cost in order to reduce fluctuations in the control signal and minimize the stress on the control mechanism. The performance of this case is compared to that of the PI-controller in terms of RMS and it is shown that the combination of reinforcement learning agent and the PI-controller leads to a lower RMS by 8.5%. Also, in this paper the effect of the learning parameters, α and γ is studied.

Anderson *et.al.* uses a neural network to reset the PI control loop when a set-point change or an inlet disturbance occurs [Anderson et al., 1997]. This paper reports a significant decrease in the rise time when their modification is in effect. More important of that, an experimental section is included in this paper. The HVAC apparatus and the control hardware used in this report is described in detail. The PI-controller is tuned while the system was set to operate at the highest expected gain, or, in other words, at low air flow and low water flow conditions. In this way, the PI-controller would remain stable at all the other possible gain states. A steady state model of the system is used for training the neural network. This model is developed via the effectiveness-NTU method. Another neural network is trained based on data collected directly from the experimental apparatus. The control performance and the rise time of these three cases are compared: a tuned PI-controller, the modified PI-controller with a neural network trained based on a model, and a modified PI-controller with a neural network trained based on real data. This comparison demonstrates a better control output performance for the neural network plus PI-controller when real data is used to train the network. The model based neural network also performs better than the PI-controller. Also it is shown that when the set-point changes (which results in the change of the gain state of the system from a high value to a low one) the rise time of the PI-controller increases noticeably while it remains almost fixed for the neural network plus the PI-controller. In addition, the response of the PI-controller is found to be slower

than the other cases for disturbance rejection. Based on the explained simulations and experiments, Anderson *et.al.* conclude that in an HVAC system, the addition of a neural network to a PI-controller lowers the coil response time while it also eliminates the sluggish control when the gain state changes from the one that the PI-controller is tuned at [Anderson et al., 2001]. Delnero, in his master thesis, [Delnero, 2001] explainers the same concept with more detail.

Tu provides another master thesis that approaches the modification of feedback control systems with reinforcement learning methods [Tu, 2001]. In this thesis, a discussion is provided on why a specific structure of reinforcement learning, which is previously introduced by Kretchmar [Kretchmar, 2000], does not learn the optimal policy. The reason mentioned for this problem is the absence of the Markov property. As a solution, Tu suggests a new learning architecture based on the state space analysis of the feedback control system. In the second part of this thesis, two continuous reinforcement learning methods are applied to feedback control which are said to outperform the previously applied discrete reinforcement learning techniques. The discussion in this thesis is based on the simulation of a simple first order plant. More complex dynamics are suggested for future work.

Hwang *et.al* introduce a technique in which reinforcement learning is used to build artificial neural networks in order to linearize a non-linear system based on feedback linearization theory [Hwang et al., 2003]. Later, the linearized plant is controlled by a regular PI-controller. This model-free scheme is named Reinforcement Linearization Learning System (RLLS). Simulation results of a pendulum system is provided showing that the proposed RLLS method presents better control reliability and robustness than traditional artificial neural network techniques.

Ernst *et.al.* introduce interesting ideas about reinforcement learning in process control [Ernst et al., 2009]. The main objective of this paper is to compare Model Predictive Controllers, or MPCs, and reinforcement learning as alternative approaches of deterministic control, in a unified scheme. This paper provides some experimental results for designing a controller for an electrical power oscillation damping case which has non-linear deterministic dynamics. These results are

used to show the advantages and disadvantages of fitted Q-iteration reinforcement learning method, over model predictive control. Ernst *et.al.* conclude that RL is strong enough in the competition with MPC even when a good deterministic model of the system is at hand. They also conclude that the applied RL method is slightly more robust than the MPC approach while the MPC scheme is more accurate. When a good model of the system is available this paper suggests that these two schemes may be used together to prevent MPC from being trapped in a local optimum: fitted Q-iteration should be applied in the off-line mode with samples from a Monte Carlo simulation. On the other hand, MPC could start its optimization process from a better initial guess, exploiting the policies accumulated by RL online, in addition to the system's model. In the case of not having an exact model of the system, instead of running system identification techniques, the direct application of fitted Q-iteration to the observation data is suggested.

Doya proposes a reinforcement learning scheme for continuous time system without discretization of time, state, and action [Doya, 2000]. This paper uses function approximation to approximate the value (cost-to-go) function and to improve the policy of the reinforcement learning agent. The estimation of the value (cost-to-go) function is based on minimizing the temporal difference, or TD, error in a continuous-time format. To improve the policy, a continuous actor-critic technique and a value-gradient based greedy policy are defined. For the case of the greedy policy, a non-linear feedback control law using the gradient of the value function (cost-to-go) and the model of the input gain is derived. The performance of this scheme is examined on a nonlinear simulation case in which the aim is to swing a pendulum up while posing some limits on the applied torque. Based on these simulations, Doya reports that the continuous version of the actor-critic accomplishes the task in much fewer number of attempts compared to the conventional discrete actor-critic technique. In addition, the value-gradient based method with a previously known or learned dynamics, outperforms the actor-critic method. Thirdly, this paper compares the two different update rules for the approximation of the value (cost-to-go) function: i) using the Euler approximation and ii) applying the exponential eligibility traces. The conclusion of Doya for this

comparison is that the addition of the eligibility traces results in a more efficient and stable performance. Finally, the proposed algorithms are examined on a cart-pole swing-up.

Gordon claims that the success of reinforcement learning methods in practical tasks depends on the ability of combining function approximation and temporal different methods [Gordon, 1995]. He mentions that it is very difficult to reason about the function approximation in the regions that observation data does not exist. Gordon, in this paper, presents a proof of the convergence for a broad class of temporal difference schemes which use different function approximation techniques such as k-nearest neighbor, linear interpolation, some types of splines, and local weighted averaging. He also demonstrates the usefulness of these schemes experimentally. Also, this paper provides a new approach of approximate value iteration method.

To compare this thesis with the reviewed work in the literature, it should be mentioned that, this thesis deals with a more complicated process, which includes a delay term in the transfer function. To the best knowledge of the author, such a delay term has not been considered in other RL related work. This thesis also presents results on the model-free cost-to-go function based reinforcement learning methods, which do not require the model of the plant, are adaptive, and can be performed online. Moreover, the experimental validation section, can be mentioned as a completely new part in this thesis. To the best knowledge of this author, this is the first reported experimental evaluation of a model-free cost-to-go function based reinforcement learning methods on a real pilot plant in the field of process control.

Chapter 4

Dynamic Tuning of PI-Controllers with Reinforcement Learning methods

4.1 Methodology

The focus of study in this thesis is the tuning of PI-controllers. This is mainly because these controllers are the most commonly used controllers in industry [Astrom and Hagglund, 1988] and they are easy to understand and implement. It is estimated that over 90% of the controllers in industry are of this type. The performance of these controllers are highly related to their tuning parameters. If these parameters are not chosen properly, these controllers would degrade process performance, often significantly.

The operation of finding the best gains for PI-controllers is called tuning. There are various tuning methods in the literature. The one that is of interest in this thesis is known as the Internal model control (IMC) tuning method, which is described in detail in Section 2.1.1. The IMC is chosen for comparative studies in this thesis, mainly because it is one of the most common tuning methods in industry.

The IMC tuning approach has some shortcomings. For example, it requires a precise model of the process which can result in poor performance when the exact model is not available or the dynamics of the process change. In addition, the IMC tuning approach suggests a range of tuning parameters for each process and not an exact value. In this way, finding the best value of the tuning parameters,

in the suggested range by the IMC, is often a challenging task. In this thesis, reinforcement learning is used as a tool to overcome these shortcomings.

The following sections describe the steps taken to implement, evaluate and compare the RL-based tuning approach with the IMC-based controllers on a simulated and a computer-interface pilot scale process.

4.1.1 System Identification

One of the main characteristics of model-free RL methods is that they do not require the model of the process in their procedure, but this does not mean that any existing information about the model of the system should be ignored. In fact, prior knowledge about the process can be used to enhance these methods. In chemical processes, usually the model of the process is known to some degree. In some cases, finding the precise model of the plant is straightforward, while in others, system identification may not be easily performed. Nevertheless, if some information is available about the model of the process, even if it is not very precise, it is recommended to incorporate this knowledge into the model-free reinforcement learning procedure, mainly because this makes the learning process faster and easier. Therefore, in this thesis, a rough model of the process is assumed to be at hand either through physical analysis or a simple system identification.

The experimental settings used here is a heated tank pilot plant which exists in the computer process control laboratory at the Chemical and Materials Engineering Department of the University of Alberta. Here, the controlled variable is the temperature of the outlet water while the manipulated signal is the flow-rate of the steam. This experimental setup is explained in Appendix A in detail.

An approximate First Order Plus Time Delay, or FOPTD model of the mentioned plant can be obtained through a simple Random Binary Sequence (RBS) excitation of the process and subsequent process identification from the input-output data. Recall that the general format of FOPTD models are shown in Equation (2.4). In addition, Equation (4.1) gives a continuous time state-space

Table 4.1: Summary of the operating conditions used in the system identification of the experimental plant:

Variable	Value	Unit
Steam Flow-Rate	6.5	$\frac{kg}{hr}$
Cold Water Flow-Rate	4.3	$\frac{kg}{min}$
Hot Water Flow-Rate	0	$\frac{kg}{min}$
Level of Water in the tank	30	cm
First Valve Closing	70	○
Second Valve Closing	60	○
Thermocouple	1	–

representation of FOPTD models.

$$\begin{aligned} x_t^{'} &= -\tfrac{1}{\tau}x_t + \tfrac{K_p}{\tau}u_{t-\theta} \\ y_t &= x_t \end{aligned} \tag{4.1}$$

The identified model is presented in Equation (4.2). In this equation, the input and the output are the steam flow-rate and the output temperature, respectively. In addition, the unit for the former is $\frac{Kg}{hr}$ while the latter is measured in degrees of Celsius ($^{\circ}C$). Here time is measured in seconds (s).

$$G(s) = \frac{1.8}{110s+1}\exp(-20s) \tag{4.2}$$

Comparing Equation (2.4) and Equation (4.2) results in Equations (4.3-4.5) which define the process parameters of our plant.

$$K_p = 1.8 \tag{4.3}$$

$$\tau = 110 \tag{4.4}$$

$$\theta = 20 \tag{4.5}$$

These model parameters are obtained for the operating condition described in Table 4.1.

4.1.2 Defining the comparative IMC controllers

The identified model shown in Equation (4.2) is used to build a simulation environment. The RL agent is trained by interacting with it before being implemented on the real plant.

Based on the model of the plant, the domain suggested in Equations (2.10-2.12) is calculated for the closed-loop time constant used in the IMC method. Then, according to this domain, six different closed-loop time constants are chosen as shown in Equation (4.6).

$$\tau_c \in \{\tau_c^{min}, \quad \tau_c^{min} + \frac{\tau_c^{max} - \tau_c^{min}}{5}, \quad \tau_c^{min} + \frac{2(\tau_c^{max} - \tau_c^{min})}{5}, \quad \dots, \quad \tau_c^{max}\} \tag{4.6}$$

where τ_c^{min} and τ_c^{max} are the minimum and maximum of the closed-loop time constant, which are calculated as 16 and 110, respectively. Accordingly six different PI-controllers are defined with their proportional and integral gains being derived from the rules summarized in Table 2.2. The values for these closed-loop time constants, and the gains are shown in Equations (4.7) and (4.8), respectively.

$$\tau_c \in \{16.0 \quad 34.8 \quad 53.6 \quad 72.4 \quad 91.2 \quad 110.0\} \tag{4.7}$$

$$GainSet = \begin{pmatrix} 1.6975 & 0.0154 \\ 1.1152 & 0.0101 \\ 0.8303 & 0.0075 \\ 0.6614 & 0.0060 \\ 0.5496 & 0.0050 \\ 0.4701 & 0.0043 \end{pmatrix} \tag{4.8}$$

Here the first element in each row is the proportional gain, while the second term is the integral gain.

4.1.3 Training in Simulation

To train the RL agent in a simulation environment, an RL algorithm named Sarsa, which is described in detail in Section 2.2.2, is implemented with the objective of combining the existing tuning parameters in a way that leads to a better performance, and a model-free self-tuning PI-controller. A graphical representation of this idea is shown in Figure 4.1. As this figure shows, a regular control loop is modified by adding an RL agent parallel to the PI-controller. This RL agent, at each time step, observes the state of the plant and sets the gains of the PI-controller as the gains of one of the previously defined IMC controllers. The following paragraphs describe the main elements of our proposed RL method, the pseudo-code of the simulation part is also provided at the end of this section.

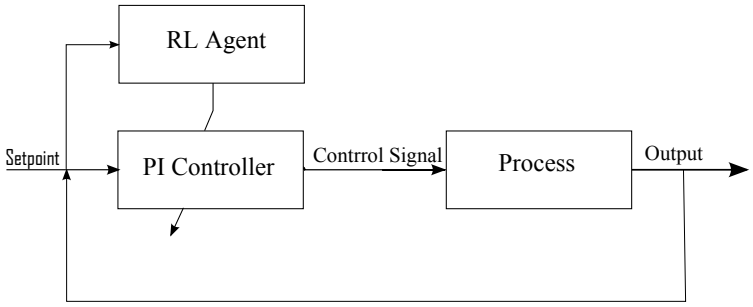

Figure 4.1: The PI-controller loop modified by RL

State

As discrete systems are easier and more practical to implement and control [Elali, 2004], Equation (4.1) is discretized to Equation (4.9) by choosing the sample time as T_s and assuming that time delay θ is the integer multiple of the sample time, T_s.

$$\begin{aligned} x_{k+1} &= \exp(-\tfrac{T_s}{\tau})x_k + \tfrac{K_p}{\tau}\int_0^{T_s}\exp(-\tfrac{t}{\tau})dt\; u_{k-\frac{\theta}{T_s}} \\ y_k &= x_k \end{aligned} \tag{4.9}$$

Applying the state space realization rules results in a state defined by Equation (4.10):

$$\widehat{x}_k = \begin{pmatrix} y_k \\ y_{k-1} \\ u_{k-1} \\ u_{k-2} \\ \vdots \\ u_{k-\frac{\theta}{T_s}} \end{pmatrix} \tag{4.10}$$

As Equation (4.10) shows, this state includes the output of the process, while the cost-to-go defined in Equation (4.22) contains the error of the system. To generalize this method for all the possible set-points, and make tracking possible, the state is modified from Equation (4.10) to Equation (4.11), by substituting the output with the feedback error, e_k and augmenting the control signal as shown in

Equation (4.12).

$$\widetilde{x}_k = \begin{pmatrix} e_k \\ e_{k-1} \\ \widetilde{u}_{k-1} \\ \widetilde{u}_{k-2} \\ \vdots \\ \widetilde{u}_{k-\frac{\theta}{T_s}} \end{pmatrix} \tag{4.11}$$

$$\widetilde{u}_k = \frac{SP}{K_p} - u_k \tag{4.12}$$

Here SP and K_p are the set-point or target, and the process gain, respectively. In this way, the set-point is included in the mapping of the state space to the cost-to-go.

Our case of interest is the application of model-free RL methods, which do not need the exact values of the process parameters. This makes the state introduced in Equation (4.11) improper, mainly because in order to calculate the augmented control signal introduced in Equation (4.12), the process gain should be known. This is in contradiction with the most important assumption of this thesis, which is not knowing the exact dynamics of the process. To get around this problem, Equation (4.11) is changed to Equation (4.13), by substituting the augmented control signal with the difference of the control signal as defined in Equation (4.14) and Equation (4.15), where $\Delta e_k = e_k - e_{k-1}$.

$$\overline{x}_k = \begin{pmatrix} e_k \\ e_{k-1} \\ \Delta u_{k-1} \\ \Delta u_{k-2} \\ \vdots \\ \Delta u_{k-\frac{\theta}{T_s}} \end{pmatrix} \tag{4.13}$$

$$\Delta u_k = u_k - u_{k-1} \tag{4.14}$$

$$\Delta u_k = K_{c,k}\Delta e_k + K_{i,k}T_s e_k \tag{4.15}$$

In this way, the set-point is incorporated in the state without the involvement of any process parameters.

Another concern that may arise here is the size of the history of the control signal that should be stored. As it can be concluded from the previously introduced equations for states, this size depends on the time delay of the process. To follow

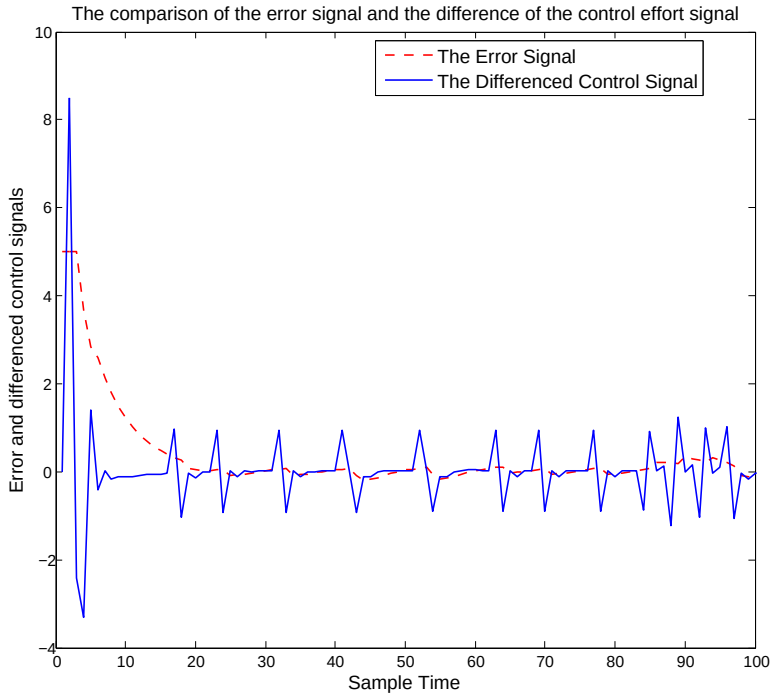

Figure 4.2: The comparison of the error and the Δu signals in an episode of learning

the assumption of not knowing the exact process parameters, it is suggested that the maximum possible time-delay is estimated and the size of the state space is determined based on that. In this way, one can be sure that the complete state of the system is stored, although the delay of the process is unknown.

Simulation results show that defining the state as shown in Equation (4.13) does not result in a learning process as good as it was expected. This can be because Δu varies in a wider domain compared to the control signal, u, or to the feedback error, e. This results in a harder function approximation problem when the difference of the control signal is used as a part of the state. In addition, there are some sudden changes for this term, which makes the function approximation problem even more troublesome. Both of these issues are shown in Figure 4.2, in which the error and the Δu signals are compared. As Figure 4.2 shows, the error signal has a smaller range of change and is much smoother, compared to the Δu signal. As a future work to this thesis, more studies are suggested to investigate the reason for this problem.

Therefore, Equation (4.13) is modified to Equation (4.16) by substituting error

terms in the place of the difference of control signal terms (The number of the components in the state vector is kept the same). This means that an observation of the state is used in the place of the complete state of the process. In this way, the process can be treated like a Partially Observable Markov Decision Process, or POMDP. As explained in Section 2.2, one way of dealing with POMDPs is to store the history of the measurements to disambiguate the state of the process. In the same manner, here, the history of the feedback error has been stored to get a better observation of the state.

$$x_k^{aug} = \begin{pmatrix} e_k \\ e_{k-1} \\ e_{k-2} \\ \vdots \\ e_{k-\frac{\theta}{T_s}-1} \end{pmatrix} \tag{4.16}$$

One should note that, the size of the state here is directly dependent on the time-delay and sample-time of the model. In fact, as the ratio of the time-delay to the sample-time goes higher, the size of the state increases, which results in a much more difficult function approximation problem. In general, a value of 0.1 to 0.2 for the ratio of the sample-time to the time-constant of the process is suggested.

As it can be concluded from Equation (4.16), the state set here is continuous.

Action

The reinforcement learning action here is the gains of the PI-controller shown in Equation (4.17).

$$a_k = \begin{pmatrix} K_{c,k} \\ K_{i,k} \end{pmatrix} \tag{4.17}$$

One should note that the same idea can be implemented for PID-controllers by including the derivative gain in the RL action, in addition to the proportional and the integral gains. For PID-controllers, the RL action will have an extra component, which, in the case of continuous action-space, will result in a more difficult optimization and learning process.

Action Set

The action set here is discrete and consists of the pairs of the gains for the PI-controller. This set is presented in Equation (4.8).

Learning Rate

The learning rate, which in this thesis is shown by α, is an important parameter which should meet the standard stochastic condition, shown in Equation (2.36), to result in a satisfactory learning performance [Sutton and Barto, 1998]. The learning rate, initially starts as 0.9 and is considered to reduce with time, so it is defined as Equation (4.18). If the learning rate is not chosen properly, the learning process may diverge. It is worth mentioning that this parameter should always be non-negative and less than unity. Equation (4.18) shows the function used as the learning rate in this thesis.

$$\alpha_k = \frac{0.9}{\text{Episode Number}^{0.5}} \tag{4.18}$$

Here episode is a set-point tracking simulation for a fixed set-point. The simulation episodes in our thesis start with the initial condition of $\overrightarrow{0}$ and finish after 105 steps.

Discount Factor

The discount factor denoted by γ should also be non-negative and less than unity. In this thesis, the discount factor is set to 0.95. A closer value to the unity for this parameter results in a slower learning process; however the horizon in which the cost-to-go is measured will be bigger. A bigger horizon leads to a bigger weighting factor for the immediate costs received through time, which may decrease the size of the overshoots and undershoots of the RL-based PI-controller. The number of the consequential immediate costs that contribute to the cost-to-go is calculated through Equation (4.19).

$$N = \frac{1}{1-\gamma} \tag{4.19}$$

Here γ is the discount factor. For $\gamma = 0.95$, N is calculated to be 20. A higher value for γ increases the value of N.

Exploration Rate

Exploration is another important concept that should be defined in reinforcement learning algorithms. Exploration helps the RL agent to find the best possible actions at each time step. The balance between exploration and exploitation is

a common issue in reinforcement learning [Sutton and Barto, 1998]. Usually, ϵ-greedy methods are used for exploration, which start with a high rate of exploration that reduces through time and the exploitation becomes more dominant. Our algorithm starts with 40% of exploration which reduces with the rate shown in Equation (4.20):

$$R_k = \frac{0.4}{\text{Episode Number}^{0.1}} \tag{4.20}$$

In this way, most of the time the greedy RL action (gains of the PI-controller) suggested by the current policy is implemented; while at other times, gains are picked randomly from the action set for this controller.

Immediate cost

The immediate cost signal is the squared feedback error, which is shown in Equation (4.21).

$$c_k = e_k^2 \tag{4.21}$$

Objective Function

The objective function here is defined as J in Equation (4.22):

$$J = \sum_{k=0}^{\infty} \gamma^k c_k \qquad \text{Where } \gamma = 0.95 \tag{4.22}$$

Function Approximation

The state space in this thesis is continuous which brings up the idea of using function approximation, for generalization. Here, the state space is divided into two different regions. First, an area which is close enough to the origin, in which the action-cost-to-go is sufficiently estimated with a quadratic function. Second, a far region for which the action-cost-to-go is approximated by k-nearest neighborhood explained in Section 2.2.3. Equation (4.23) presents this idea. (The way the appropriate region for quadratic function approximation is chosen is explained later

in this section)

$$Q(x_k^{aug}, a_k) = \begin{cases} Q(x_k^{aug}, a_k) \quad \text{read from the memory} & \text{for } |x_k^{aug}| \geq \delta \text{ \& existing point} \\ \sum_{i=1}^{n}(CO_k \otimes NCost_k)_i / \sum_{j=1}^{n}(CO_k)_j & \text{for } |x_k^{aug}| \geq \delta \text{ \& new point} \\ \overrightarrow{\theta_{a_k,k}} x_k^{aug} \otimes x_k^{aug} & \text{for } |x_k^{aug}| < \delta \end{cases} \tag{4.23}$$

where $\otimes$ is the element-wise crossing operator, x_k^{aug} is defined in Equation (4.11), θ is the row vector of parameters with the initial value of $\vec{0}$, δ is a positive number (in our thesis $\delta = 0.4$), which defines the boundary between the regions close to and far from the origin, CO is the contribution factor defined in Equation (4.24), $NCost$ is the vector of the cost of the neighbors, and $<$ and $\geq$ are element-wise operators.

$$CO = \exp(-d^2/\lambda^2) \tag{4.24}$$

Here, d is the distance between the neighbor and the new point, and λ is a positive tuning parameter (In our thesis $\lambda = 1$.

The update rules for these function approximators are also provided in Equation 4.25).

$$\begin{cases} Q(x_k^{aug}, a_k) = Q(x_k^{aug}, a_k) + \alpha[c_{k+1} + \gamma Q(x_{k+1}^{aug}, a_{k+1}) - Q(x_k^{aug}, a_k)] & \text{for } |x_k^{aug}| \geq \delta \\ NCost_k = NCost_k + CO_k \otimes (Q(x_k^{aug}, a_k) - NCost_k) / \sum_{j=1}^{n}(CO_k)_j & \text{for new points} \\ \overrightarrow{\theta}_{a_k,k+1} = \overrightarrow{\theta}_{a_k,k} + \alpha[Q(x_{k+1}^{aug}, a_{k+1}) - Q(x_k^{aug}, a_k)](x_k^{aug} \otimes x_k^{aug}) & \text{for } |x_k^{aug}| < \delta \end{cases} \tag{4.25}$$

In this thesis, for the k-nearest neighborhood function approximation, instead of defining a value for the number of the neighbors involved in the estimation of the cost of a new point, a sphere around the new point is determined as the neighborhood for that, and any existing point in this area is put into account in the approximation of the cost of it, later the update rule is applied to all the points involved in the approximation. The radius of this sphere ($r = 4$) is obtained from the comparison of the performance of different simulations with different values of it. In addition, it is important to note that the predictions of a function approximator in general are just valid for the regions of the state space that are covered by previously visited points, unless the structure of the approximated function is known [Smart and Kaelbling, 2000]. So, here, the dimensions of the new state is compared with the dimension of its neighbors to make sure that function

approximation is just used to interpolate among the neighbors, and extrapolation is avoided.

To have an initial set of data points to start with, the process is controlled with fixed-gain PI-controllers for the previously defined 40 different set-points. The gains of these controllers are set as the gains introduced in Action Set past of Section 4.1.3. The state-action points visited during these simulations and their approximate action-cost-to-go are stored and used as the initial set of data points for the k-nearest neighborhood function approximation. Here, the action-cost-to-go value is estimated by calculating the weighted summation of the immediate costs gained after visiting each pair of state and action. The weighted summation of the immediate cost is known as return, and is shown in Equation (4.26), where $\gamma = 0.95$, and $c_k = e_k^2$.

$$R = \sum_{k=0}^{\infty} \gamma^k c_k \tag{4.26}$$

The pseudo-code of the initialization of the memory for k-NN is shown in Algorithm 5.

Algorithm 5 Initialization of the state-action and cost memory for k-NN function approximation

for each $K \in GainSet$ **do**
  for Set-point=-5:0.25:5 **do**
    Use K to define a fixed-gain PI-controller
    Set x and x^{aug} to zero
    Control the process by the defined controller and store all the visited state-action points, and the earned immediate costs
  end for
end for
for all the visited state-action pairs **do**
  Approximately calculate the associated cost through $\sum_{k=0}^{\infty} \gamma^k c_k$, where c_k is the immediate cost at each step
  if the state-action pair is new **then**
    Add the state-action pair and its mapped cost to the memory
  end if
end for

Function approximation has a huge impact on the performance of the learning process. If the function approximator is chosen wisely, learning will approach its

optimum value faster, requiring less amount of data. As a result of this impact, it is suggested to study on the function approximation and pick out the best possible choice. For more information in this case refer to Section 2.2.3.

Defining the region around the origin for quadratic function approximation

To have stability near the origin, a quadratic function approximation is implemented around the origin. It should be noted that any other structure for the function approximation close to the origin may result in regulating problems. Also, it is known from the optimal control theories that the optimal cost-to-go function for an infinite-horizon state-feedback control problem for unconstrained linear systems is a quadratic function of state [Astrom and Wittenmark, 1996]. The mentioned reasons are used as a motivation of choosing a region close to the origin and implementing quadratic function approximation in that part. For the rest of the state-action space, k-NN approximation is used to estimate the cost-to-go function. The reason for not using the quadratic function approximation for the entire state-action space is that, this approximation method is really sensitive to the way the state-action space is sampled. Each visit results in an update of the parameter vector, which changes the cost-to-go of many other state-action pairs. Consequently, the vector of parameters may fluctuate and not converge to a value for big state-action spaces.

To find a region in which quadratic function approximation is precise enough, the initial data, obtained from the simulations of different fixed-gain PI-controllers is used. This data set is divided into different squares of data with varying dimension. Then, on each set of data, a quadratic function is fitted and the squared R is calculated for that. Comparing the squared R, the set of data that has the biggest size and an acceptable squared R is chosen as the region near the origin in which quadratic function approximation is implemented. This idea is shown in Algorithm 6.

The simulation procedure of this thesis consists of two parts that are implemented continuously one after the other. The first one is the learning or training part. In which the RL agent interacts with the simulated environment and improves

Algorithm 6 Defining the size of the region around the origin for quadratic function approximation

for counter=1:20 **do**
  Set $NeighborSet(counter) = x^{aug}$ for $|x^{aug}| < 0.1 * counter$.
  Set $CostSet(counter)$ as the set of the costs to which each member of $NeighborSet$ is mapped.
  Fit a quadratic function on $NeighborSet(counter)$ as X and $CostSet(counter)$ as Y.
  Calculate $R^2(counter)$ for this fitting.
end for
Set $r = 2$
Set $temp = R^2(20)$
for counter=19:1 **do**
  if $R^2(counter) > temp$ **then**
    Set $r = 0.1 * counter$
  end if
end for

its policy based on the reaction of the process. The learning simulation itself includes 40 episodes of learning, in which a random-binary-signal is fed to the system as the set-point, so as to cover different frequencies of change, and visit a wider region of the state-space. The levels of this random-binary signal is set as shown in Equation (4.27), in which b for each episode has a value between -5 to $+5$ with the step-size of 0.25.

$$Level = [-b \quad b] \tag{4.27}$$

Each episode lasts for 105 sample-time steps to make sure that the output has reached its steady-state value.

4.1.4 Evaluation of the RL policy

The second part of the simulation of this thesis is the evaluation part in which the performance of the RL-based PI-controller is measured. In this part of the simulation, the policy of the RL remains unchanged, and it is just used to set the gains of the PI-controller and the evolution of the output is observed. This part also includes 40 episodes of evaluation with 105 steps. In each episode the set-point is fixed and has a value between -5 and $+5$ with the step-size of 0.25.

To measure the control performance, the integral of the squared error, or ISE, defined in Equation (4.28), is calculated in each episode, and the average of its final value through the evaluative part, over the 40 episodes with different set-points, is compared to that of the IMC for the same case.

$$ISE = \sum_{k=0}^{\infty} e_k^2 \tag{4.28}$$

To show that learning is actually improving the policy of the RL Agent, learning performance is also measured by calculating the average of the return or the weighted summation of the immediate cost (which in this thesis is the squared error) over those 40 episodes of evaluation with 40 different set-points. This value is compared to that of the IMC, as well. The main reason for caring about return as an evaluation of the learning performance, in addition to the ISE, is that return is actually what the RL agent tries to learn and minimize, not the ISE. ISE is a well-known control performance measurement in industry (not the return). For these reasons, both return and ISE are evaluated and presented in this thesis. The difference between return and ISE is in the discount factor of return, which makes it bounded, unlike the ISE term.

The evaluation part is implemented after each 10 batches of learning episodes. Each batch starts with an episode with the set-point of -5, and finishes with an episode with the set-point of +5. The methodology of the simulation section of this thesis is summarized in Algorithm 7.

4.2 Simulation Results

In this section, the simulation results for our proposed RL-based PI-controller is provided for the case of set-point tracking. The experiments performed later in Section 4.3.1 are also implemented for set-point tracking, while for this case disturbance rejection studies are also carried out. The improvement resulted from the learning simulations in the performance of the RL-based PI-controller, and its comparison to the IMC controllers are shown in Figure 4.3. As it can be seen from this graph, both the ISE and the return for the RL-based PI-controller decrease

Algorithm 7 The algorithm of the simulation procedure

Identify the model of the plant roughly
Build a simulation environment based on the obtained model
Calculate the domain suggested for closed-loop time constants by IMC
Choose six different closed-loop time constants from the calculated domain
Calculate the IMC parameters based on the chosen closed-loop time constants
Build an initial set of data for function approximation with $k - nn$
Set $\delta = 0.4$
Set the initial vector parameter, $\overrightarrow{\theta}_0 = \overrightarrow{0}$
repeat {Training part}
  for BatchCounter=1:10 **do**
    for Set-point=-5:0.25:5 **do**
      Initialize the first state, $x^{aug} = \overrightarrow{0}$
      Calculate $Q(x^{aug}, a)$ from Equation (4.23) for all the RL actions
      If the visited state-action pair is a new point, add it to the memory and assign its cost-to-go , which is calculated in the previous step
      Comparing the calculated $Q(x^{aug}, a)$ for all the possible RL actions, choose the action that has the lowest $Q(x^{aug}, a)$ (greedy action)
      Set a = greedy action, with a small probability ϵ overwrite a with a random action from the action set
      for each x^{aug} visited through the episode **do**
        Implement action a, observe the immediate cost c, and the next state $x^{'}$
        At state $x^{'}$ set $a^{'}$ as the greedy action as described before
        With a small probability ϵ, choose a random action and overwrite $a^{'}$
        Implement the update rules introduced in Equation (4.25)
        $x^{aug} \longleftarrow x^{'}$
        $a \longleftarrow a^{'}$
      end for
    end for
  end for
  for Set-point=-5:0.25:5 **do** {Evaluation part}
    Initialize the first state, $x^{aug} = \overrightarrow{0}$
    Calculate the greedy action, a
    for each x^{aug} visited through the episode **do**
      Implement action a and observe the consequent immediate cost c, and the next state $x^{'}$
      At state $x^{'}$ set $a^{'}$ as the greedy action as described before
      $x^{aug} \longleftarrow x^{'}$
      $a \longleftarrow a^{'}$
      Calculate $Return = \sum_{k=0}^{\infty} \gamma^k c_k$
    end for
  end for
  Calculate the average of the calculated $Return$ over the set-points
until The improvement in the performance is negligible

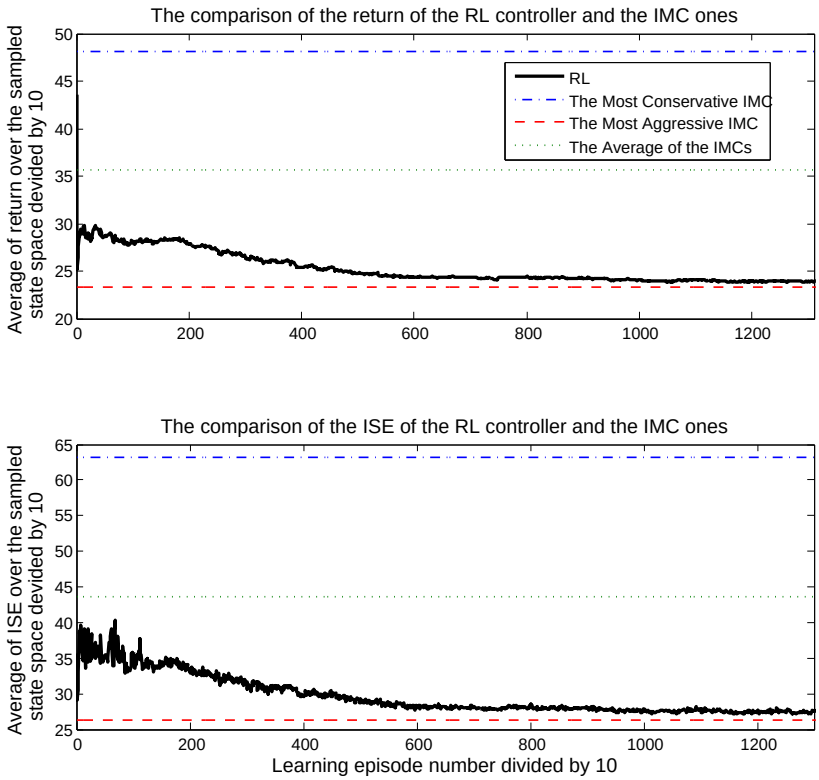

Figure 4.3: The performance improvement through the learning process, simulation results for the identified model

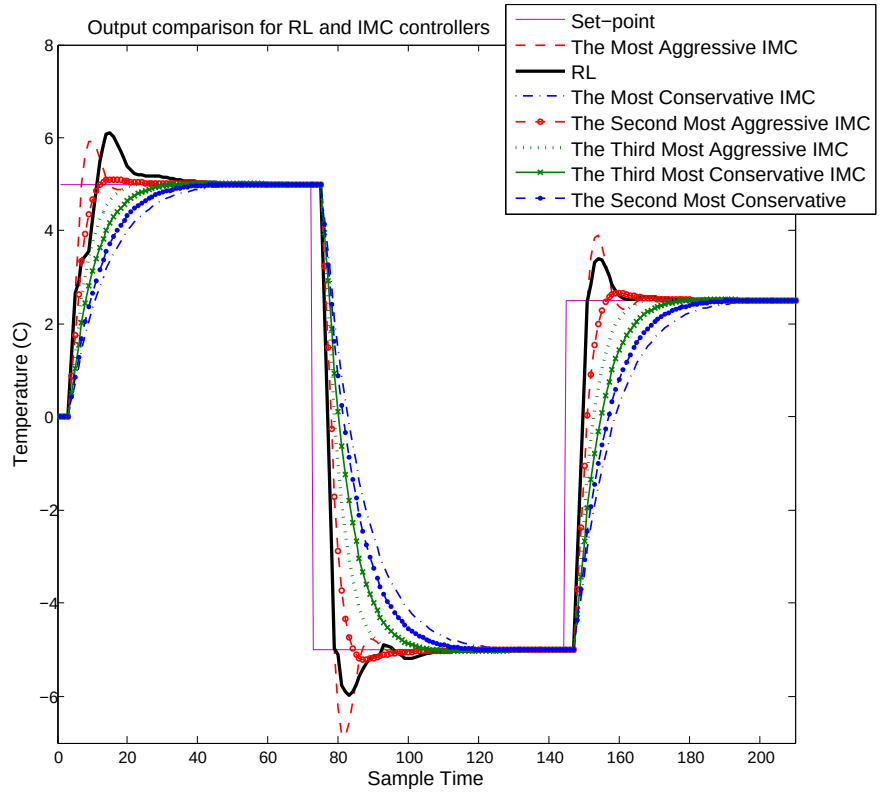

Figure 4.4: The comparison of the process output of RL and the IMC controllers in simulation environment after learning

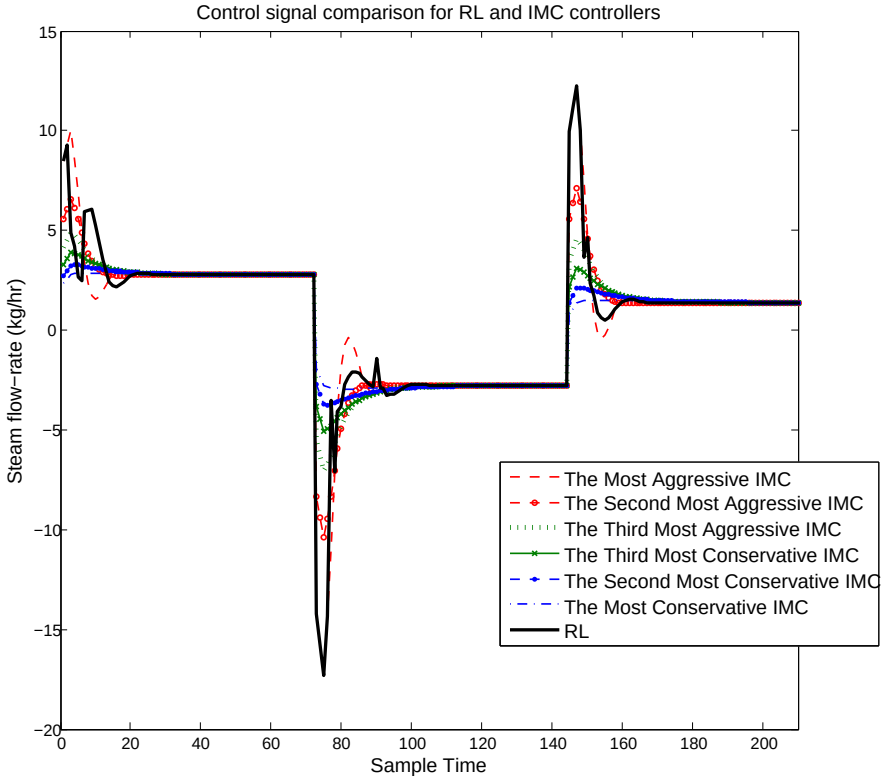

Figure 4.5: The comparison of the control signal of RL and the IMC controllers in simulation environment after learning

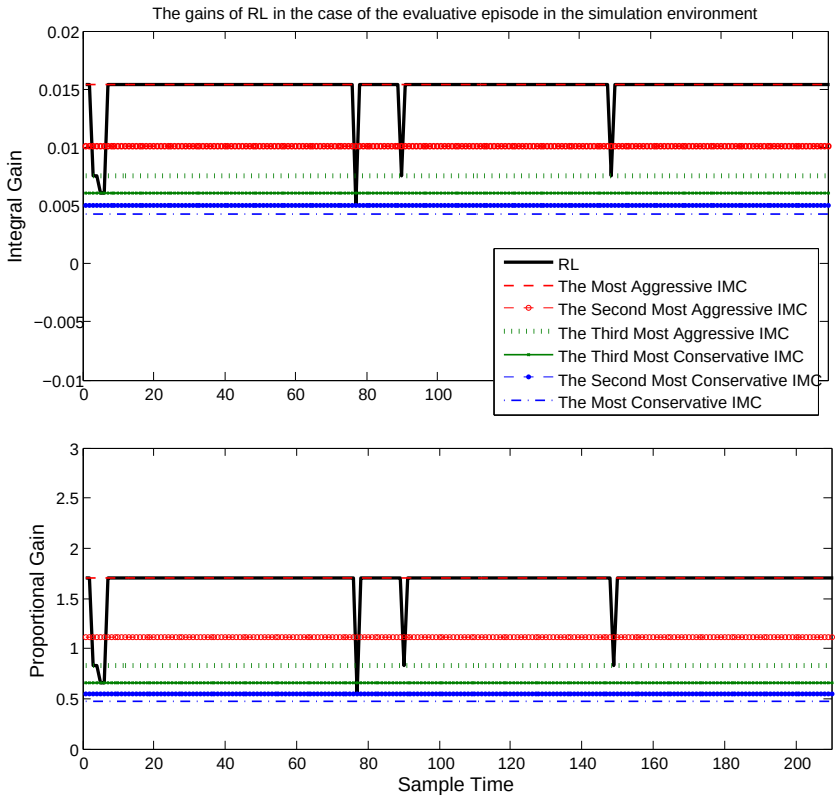

Figure 4.6: The change in the gains of the RL-based PI-controller for the process output shown in Figure 4.4

through the learning simulations and converge to a value which in the case of both ISE and return is very close to the most aggressive IMC controller.

Figure 4.4 demonstrates the comparison of the process output of the RL-based PI-controller and all the IMC controllers used in the action-set of the RL agent. As it can be seen from this figure, the output response of the RL-based PI-controller, most of the time, is as fast as the most aggressive IMC controller, while it shows a lower overshoot (or undershoot). The same concept is shown for the control signal in Figure 4.5. Figure 4.6 shows the gains used by the RL for the process output shown in Figure 4.4. As this figure displays, the RL-based PI-controllers chooses the highest gains for most of the time steps to result in a rapid convergence to the set-point. It is just some steps after the implementation of the step change in the set-point that RL reduces the gains in-use. As the former errors in the episode have a higher share in return, trying to have a fast rising process, while reducing the overshoot (or undershoot), seems reasonable to result in a low return, which is the real objective of the RL agent. As the evolution of the return through the learning period, and the value of the return for the most aggressive IMC controller shows, it can be expected that eventually the RL-based PI-controller behaves like the most aggressive IMC controller (fixed gain), which apparently has the optimum value of the return and ISE. It is known from optimal control theories that the optimum controller for an infinite-horizon state-feedback control problem for unconstrained linear systems is a fixed gain PI-controller [Astrom and Wittenmark, 1996]. The main reason for implementing the simulation part in this thesis is to obtain a wise initial policy for the RL-agent in the experimental part. Through learning in simulation environment, different state-action pairs are visited and their action-cost-to-go is approximated for implementation on the experimental setup.

4.3 Experimental Validation

After obtaining simulation results, the simulation-based RL is trained on the real plant and its performance is evaluated, both for set-point tracking and disturbance rejection. For this, firstly the gains of the PI-controller are obtained from the

simulation-based policy; then the control signal is calculated based on the measured feedback errors; afterwards, this control signal, which is the steam flow-rate is set on the plant; finally the new feedback error is measured.

As the first step, the performance of the imported policy is evaluated. As a result of the possible plant-model mismatch, at this stage the performance of the RL agent may not be as good as its performance in the simulation environment. Therefore, starting from the policy learned in the simulation environment, RL agent is re-trained interacting with the real process, so as it adapts to its new environment. It should be noted that this is a way of incorporating any prior knowledge in our model-free RL procedure. Learning from scratch on the real plant is not practical, because it can be time-consuming and sometimes not safe enough. The algorithm of this procedure is shown in Algorithm 8.

Algorithm 8 The algorithm of the experimental validation part

1: Start up the plant at its operating conditions mentioned in Table 4.1
2: Use the learned policy of the simulation-based RL agent as the initial policy of the experimental part
3: Evaluate this initial policy
4: **repeat**
5: Train the policy on the pilot plant using Sarsa as described before
6: Evaluate the modified policy
7: **until** the desired performance is achieved

4.3.1 Set-point Tracking

Figure 4.7 presents the comparison of the process output for the RL-based PI-controller, the most aggressive, and the most conservative PI-controllers. As it can be concluded from this figure, the process output for both the RL, and the most aggressive IMC controller shows some oscillations due to the existing model-mismatch. This is while the most conservative IMC controller is very slow in tracking the set-point. The control signal for the same evaluative episode is shown in Figure 4.8. One should note that the control signal for the case of performing experiments on the real plant, unlike the simulation part, is bounded due to the physical constraints governing the real plant. Here the control signal is the flow-

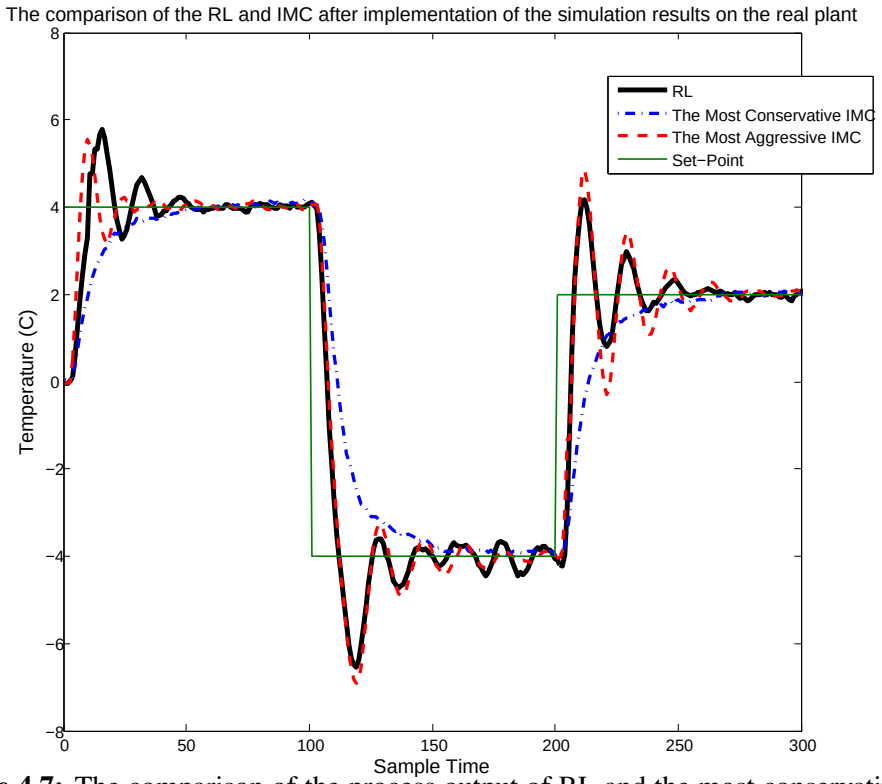

Figure 4.7: The comparison of the process output of RL and the most conservative and the most aggressive IMC controllers on the real plant after being trained in simulation environment.

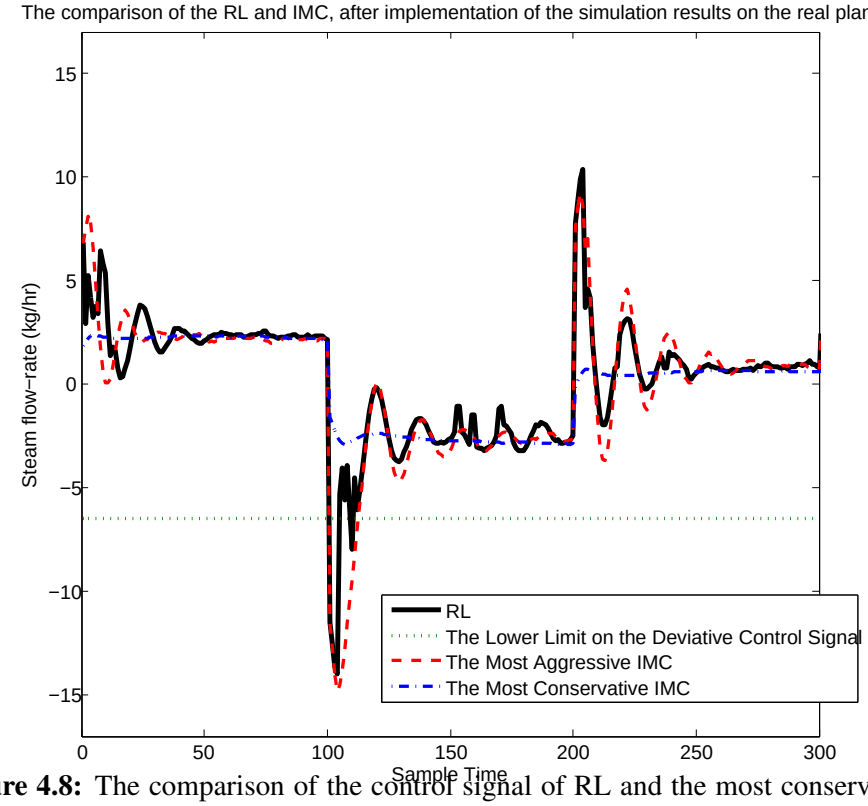

Figure 4.8: The comparison of the control signal of RL and the most conservative and the most aggressive IMC controllers on the real plant after being trained in simulation environment.

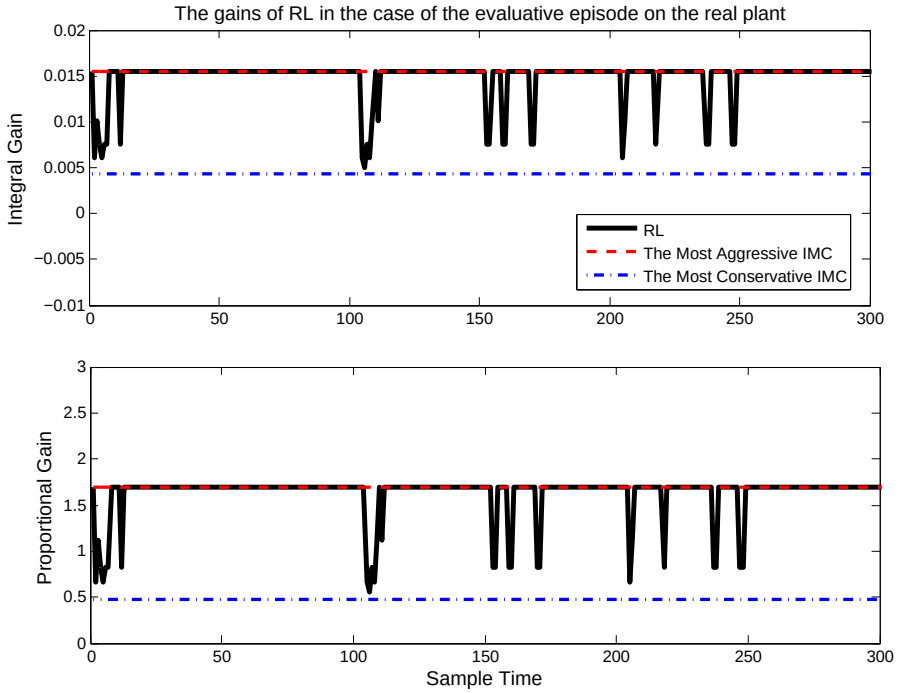

Figure 4.9: The change in the gains of the RL-based PI-controller for the process output shown in Figure 4.7

rate of the steam, whose absolute value is bounded between zero and an upper limit. In process control, variables are usually used in deviation format. Therefore, the limits of the control signal shown in our results are shifted towards the negative axis for $6.5\frac{kg}{hr}$, which is the operating value for the steam flow-rate. Whenever the RL agent, (trained in the simulation environment) suggests a value beyond the governing limits, that value is overwritten by the closest bound to it.

Figure 4.9 displays the gains applied by the RL agent during the evaluative experimental episode. This figure shows a significantly higher amount of gain change compared to Figure 4.6. This is mainly because of the existing model-mismatch, and therefore, the wrong mapping from the states to the suitable gains.

To compare the performance of the RL-based PI-controllers against the IMC controllers numerically, the ISE and the return are calculated during the same evaluative episode, both for the RL and the IMC controllers. Figures 4.10 and 4.11 respectively present the progress of these two values during the evaluative experimental episode. Table 4.2 also summarizes the final value of the ISE and

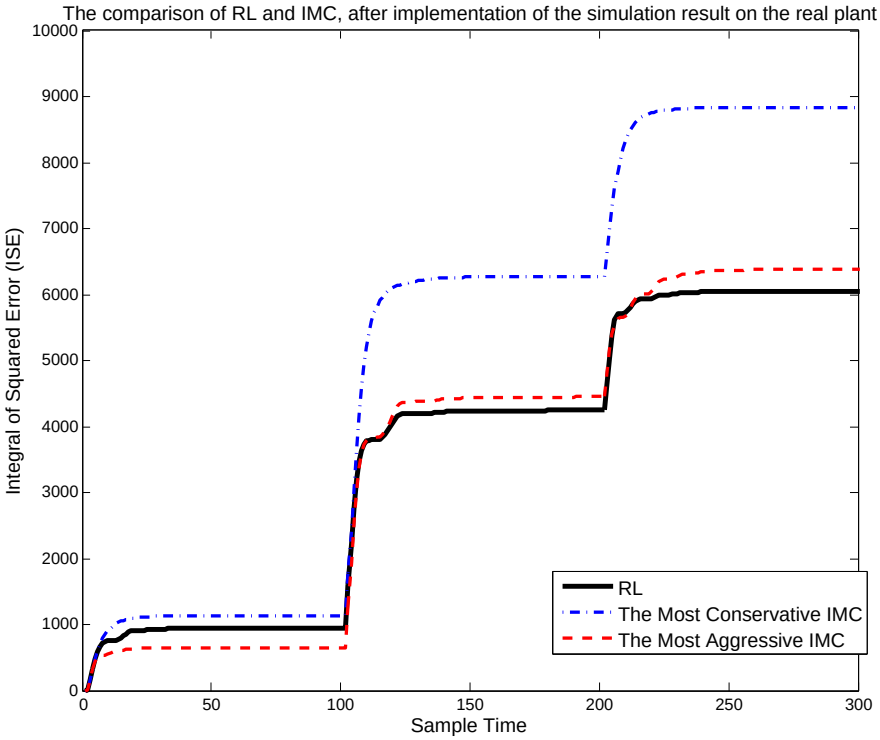

Figure 4.10: The comparison of the ISE for RL and the most conservative and the most aggressive IMC controllers through the initial episode of evaluation on the real plant, after being trained in the simulation environment.

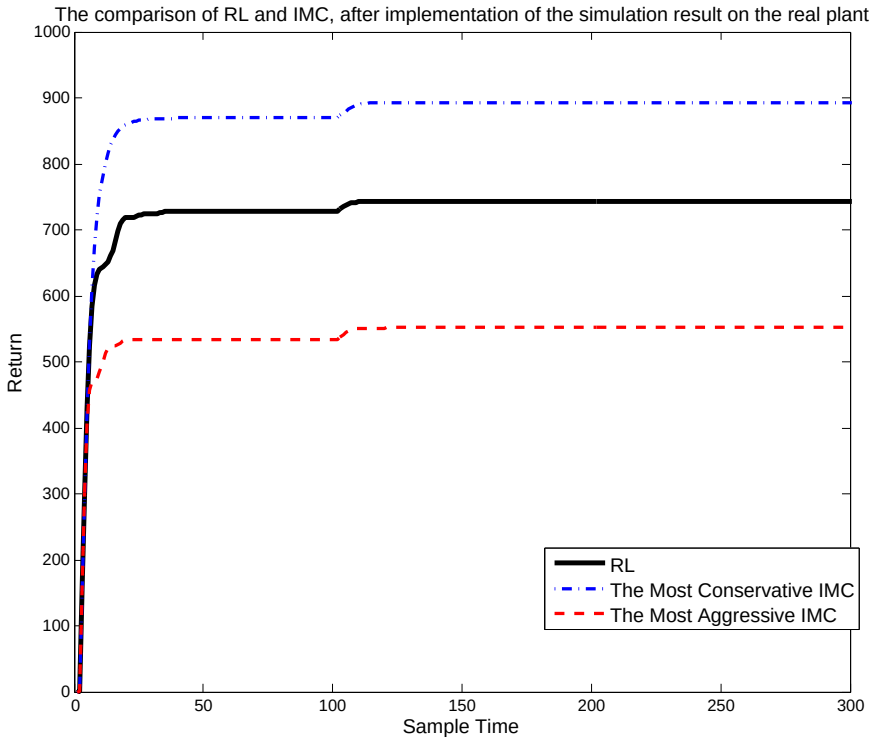

Figure 4.11: The comparison of the return for RL and the most conservative and the most aggressive IMC controllers through the initial episode of evaluation on the real plant, after being trained in the simulation environment.

Table 4.2: The comparison of the ISE and the return for the RL-based PI-controller, the most aggressive IMC, and the most conservative IMC, in an evaluative episode on the real plant, after being trained in the simulation.

	RL	The Most Aggressive IMC	The Most Conservative IMC
Return	743	552	893
ISE	6046	6380	8838

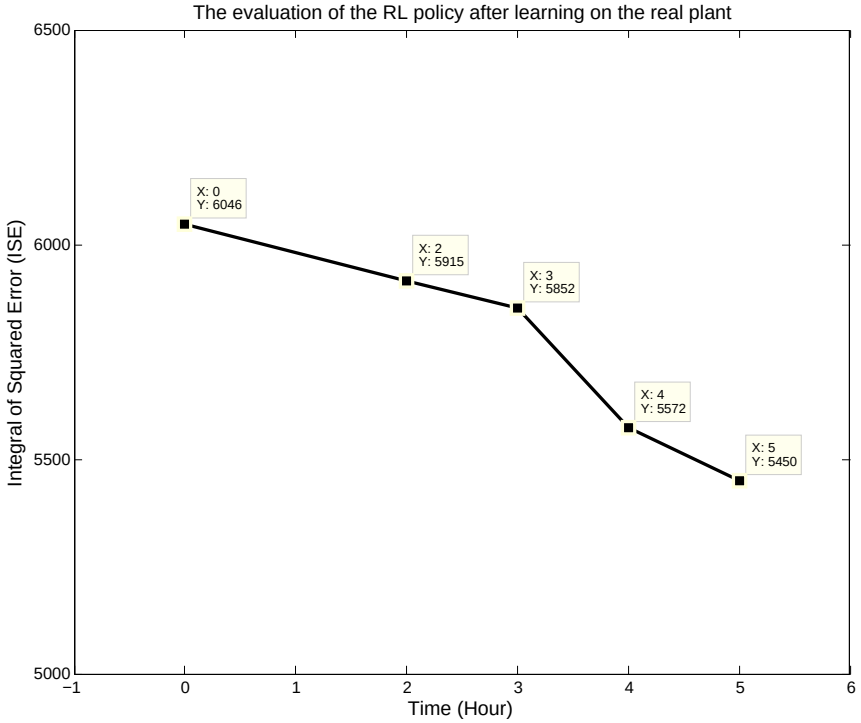

Figure 4.12: The improvement of ISE for the RL-based PI-controller through the learning process on the real plant

return for the RL, the most aggressive, and the most conservative IMC controllers. These values lead us to conclude that the RL-based PI-controller has a better ISE compared to the most aggressive and the most conservative IMC controllers, while it stands between these two controllers in terms of return.

Adaptation of the RL-based PI-controller in the case of set-point tracking

Next, the adaptability of the RL agent to model change is studied. For this aspect, the RL agent is set to the training mode on the real plant, and once a while its control and learning performance are re-evaluated. In this way, the first evaluation is happened after two hours of training, while after that, training is stopped hourly,

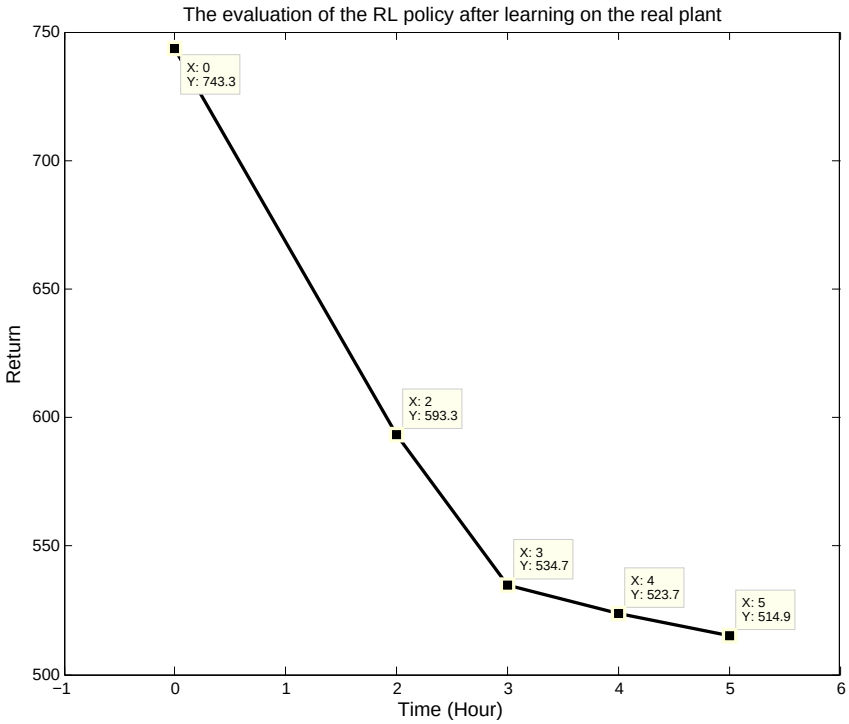

Figure 4.13: The improvement of return for the RL-based PI-controller through the learning process on the real plant

Table 4.3: The improvement of the ISE and the return through the learning process on the real plant.

	Return	ISE
The Initial Value	743	6046
2 Hours of Learning	593	5915
3 Hours of Learning	535	5852
4 Hours of Learning	524	5719
5 Hours of Learning	515	5450

and the policy of the RL agent is evaluated. Table 4.3 summarizes the obtained values for the ISE and return during these re-evaluations. Also, Figures 4.12 and 4.13 are plotted based on the data provided in this table. As it shown by these graphs, the performance of the RL-based PI-controller improves significantly after being trained on the real plant. Therefore, it can be concluded that the RL agent shows an indicative adaptability to the change in the process. Further improvement in the performance of the RL-based PI-controller can be expected if more training is performed.

In addition, Figure 4.14 compares the process output of the RL-based PI-

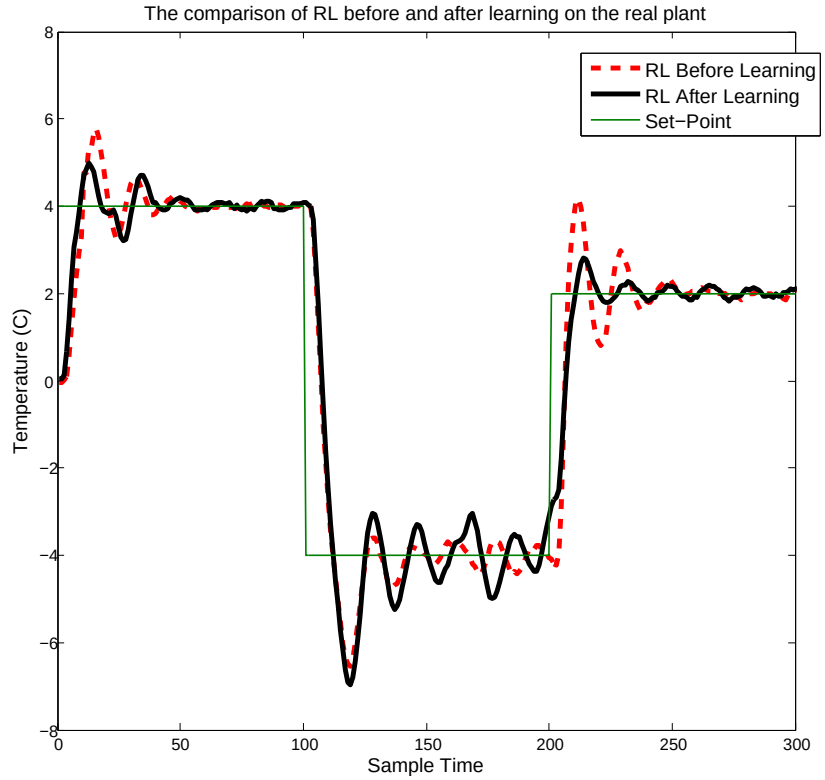

Figure 4.14: The comparison of the process output for RL before and after learning on the real plant.

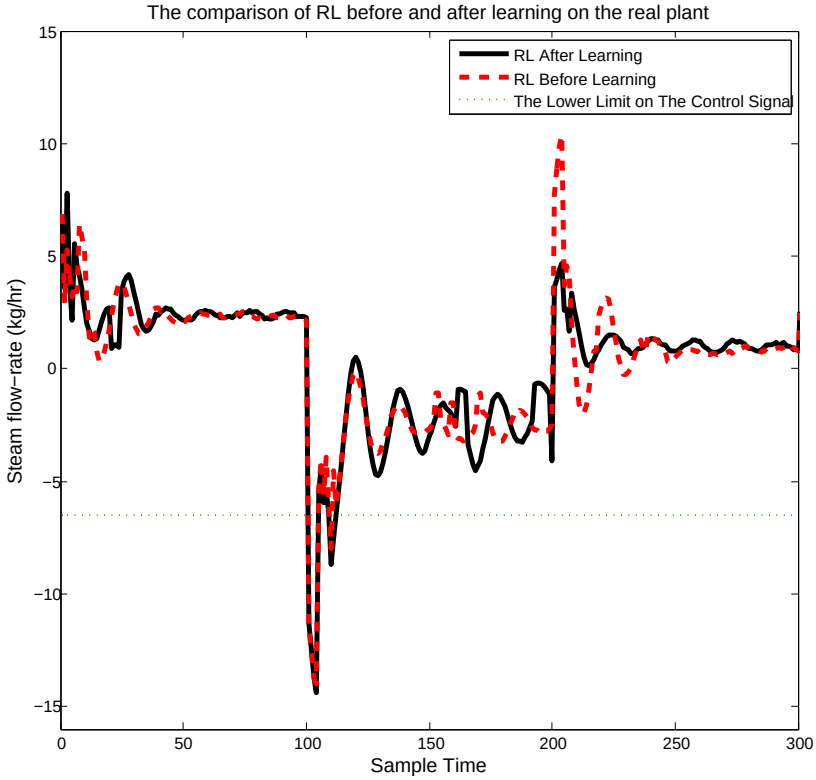

Figure 4.15: The comparison of the control signal for RL before and after learning On the real plant.

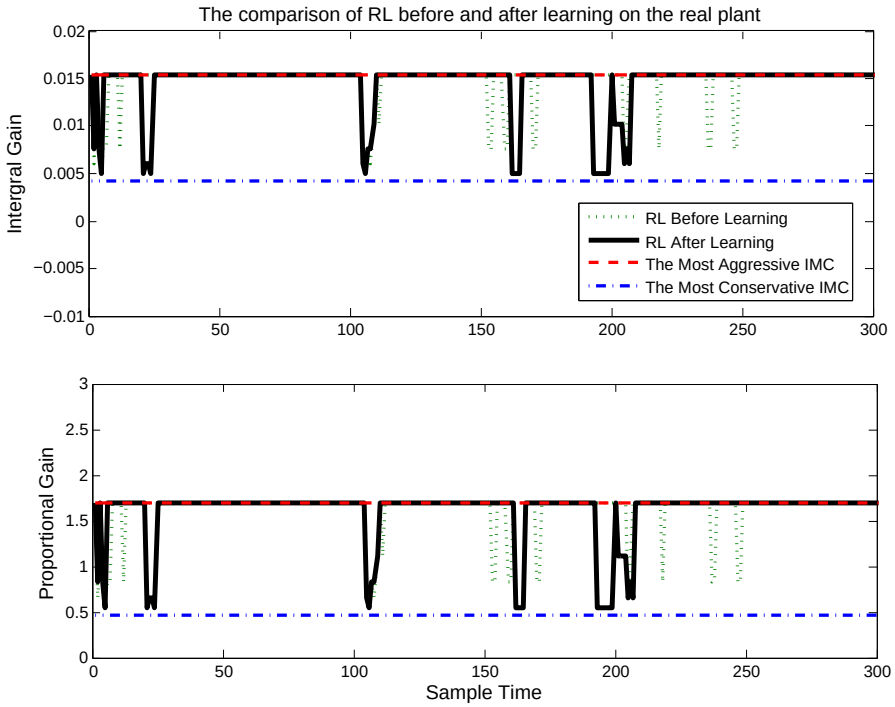

Figure 4.16: The comparison of the gains for RL before and after learning on the real plant.

controller before and after learning on the experimental plant. As one can see in this figure, the process output for the positive step changes in the set-point, shows a significantly lower overshoot, and less oscillation. In the case of the negative step change in the set-point, after the mentioned amount of learning, the RL agent does not show any noticeable improvement. It should be mentioned that in the case of the negative step-change in the set-point, the values suggested by the RL agent violates the existing physical constrains, so for some parts of the evaluative episode, the control signal is over-written by the constraint, and the suggestion of the RL agent is not feasible. In addition, it is observed that the heating and cooling dynamics of the plant are very different, due to the difference in the heating and cooling surfaces, and the hysteresis of the control valve on the steam line. The fitting results obtained for the identified model shows that this model has poorer predictions for the negative steps rather than the positive ones. Based on all that was mentioned, it can be concluded that the RL agent is dealing with a bigger change between the simulated model and the real plant, when the step-change in the set-

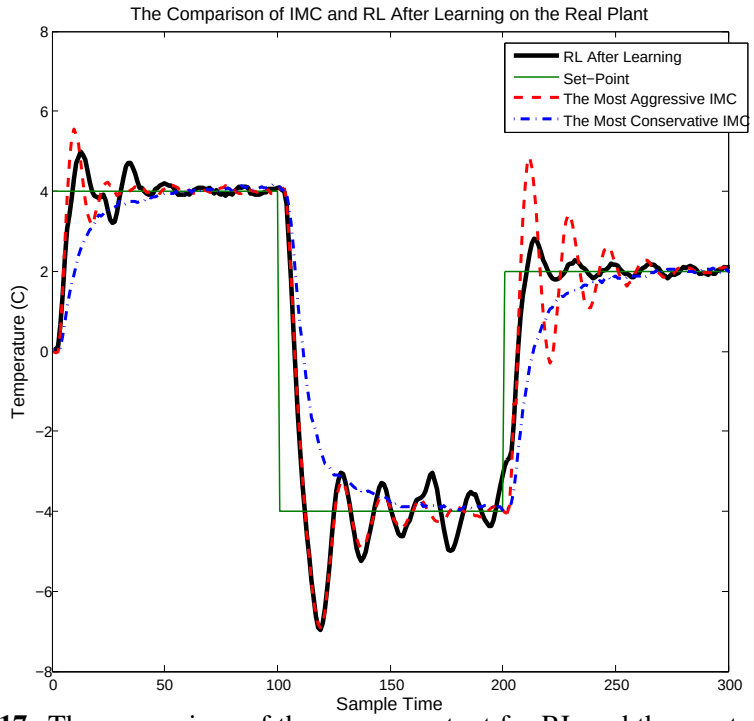

Figure 4.17: The comparison of the process output for RL and the most aggressive and the most conservative IMC controllers after learning on the real plant.

point is negative. This results in an inherently harder learning problem, which may require more time to show improvement in the performance, compared to the case of the positive step-change.

Figure 4.15 shows the control signal for the process output shown in Figure 4.14, while Figure 4.16 presents the applied gains by the RL agent for the same case. As it can be seen in Figure 4.15, the control signal of the RL-based PI-controller shows some high overshoots or undershoots. This is mainly a result of the way the objective function and immediate cost is determined in this thesis. Here the former is defined as Equation (4.22), while the latter is determined in Equation (4.21). As these equations shows, what we care about is the amount of the error through time, not the applied control signal or its change. If the amount of the applied control effort matters, a specific term, which is a function of the magnitude of the control signal (for example the absolute value of the control signal) should be added to the cost-to-go and the immediate cost.

The process output and the control signal of the RL-based PI-controller after learning on the real plant is also compared to the process output and the control signal of the most aggressive and the most conservative IMC controllers. The

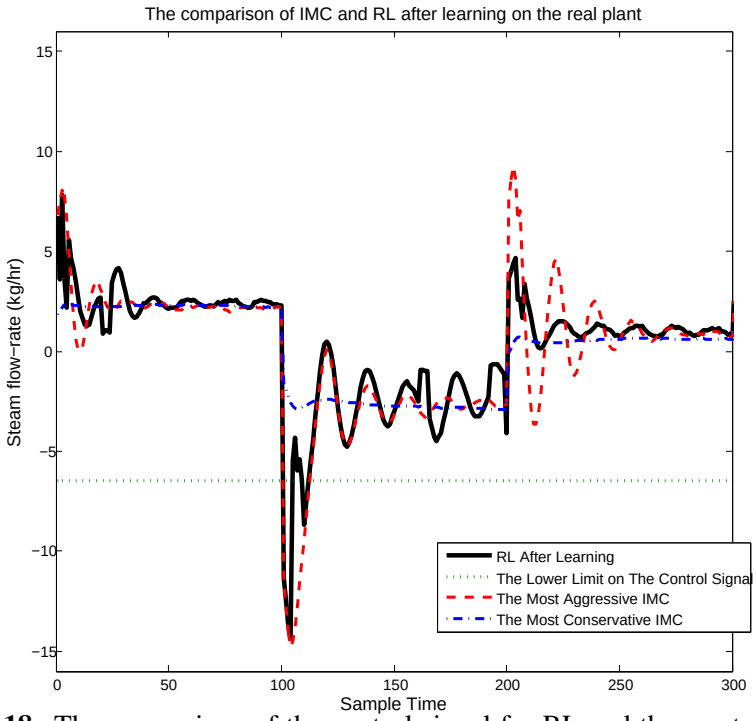

Figure 4.18: The comparison of the control signal for RL and the most aggressive and the most conservative IMC controllers after learning on the real plant.

former is shown in Figure 4.17, while the latter is depicted in Figure 4.18. As these figures present, RL-based PI-controllers performs significantly better than the IMC controllers, after being trained on the experimental plant. This can be numerically verified by comparing the ISE, which is 6380.6, 8838.3, and 5450.4 for the most aggressive IMC, the most conservative IMC, and the RL-based PI-controller after being trained on the real plant, respectively.

4.3.2 Disturbance Rejection

Next, the ability of the RL-based PI-controller is tested in rejecting disturbances. For this test, first the test plant is started up at the operating condition and is set to reach the steady-state. Then, at the steady-state, first a positive disturbance is introduced to the plant, by opening the hot water valve, and setting the flow-rate of the hot water at $0.3\frac{kg}{min}$. Then, after sufficient time for the output to reach steady-state, a bigger positive disturbance is applied to the system, by opening the hot water valve more, and setting the flow-rate of the hot water at $0.9\frac{kg}{min}$. Finally, after enough time is passed for the process to reach steady-state, a negative disturbance is introduced to the plant by closing the hot water valve, so the flow-

Table 4.4: The disturbance rejection settings

	Sign	The Flow-Rate of the Hot Water $\frac{kg}{min}$
First Disturbance	Positive	$0.3\frac{kg}{min}$
Second Disturbance	Positive	$0.9\frac{kg}{min}$
Third Disturbance	Negative	$0.6\frac{kg}{min}$

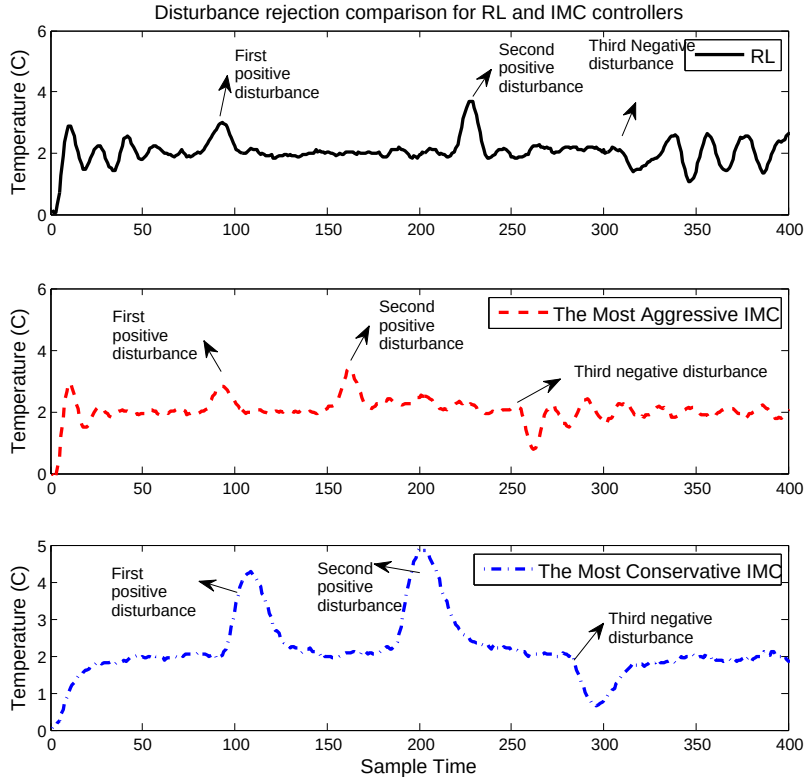

Figure 4.19: The comparison of the process output of RL and the most aggressive, and the most conservative IMC in the case of disturbance rejection.

rate of the hot water is set to $0.6\frac{kg}{min}$. The settings of the disturbance rejection steps are summarized in Table 4.4. The same set of experiments are carried out for the most aggressive and the most conservative IMC controllers, as a truth-ground for comparison with the RL-based PI-controller.

Figure 4.19 displays the output of the RL-based PI-controller in comparison with the IMC controllers. The first set of oscillations are in the start-up process. Then, the effect of the first disturbance can be observed as the first jump in the output, which happens for all the controllers around the sample time of 100. Figure 4.19 shows that all the three controllers are successful in rejecting the disturbance. Also, it is noticeable that the RL-based PI-controller shows an

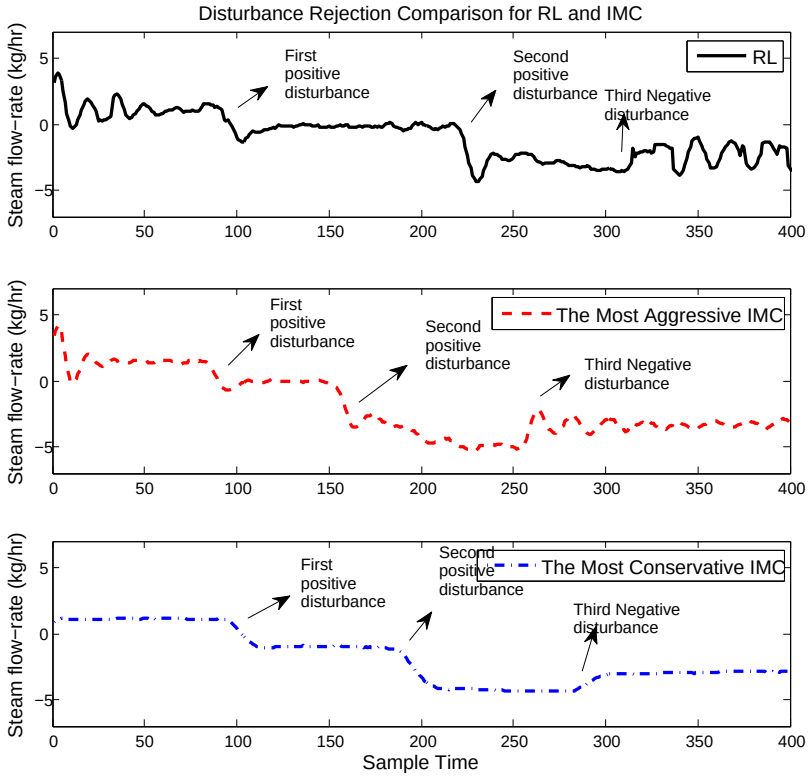

Figure 4.20: The comparison of the control signal of RL and the most aggressive, and the most conservative IMC in the case of disturbance rejection.

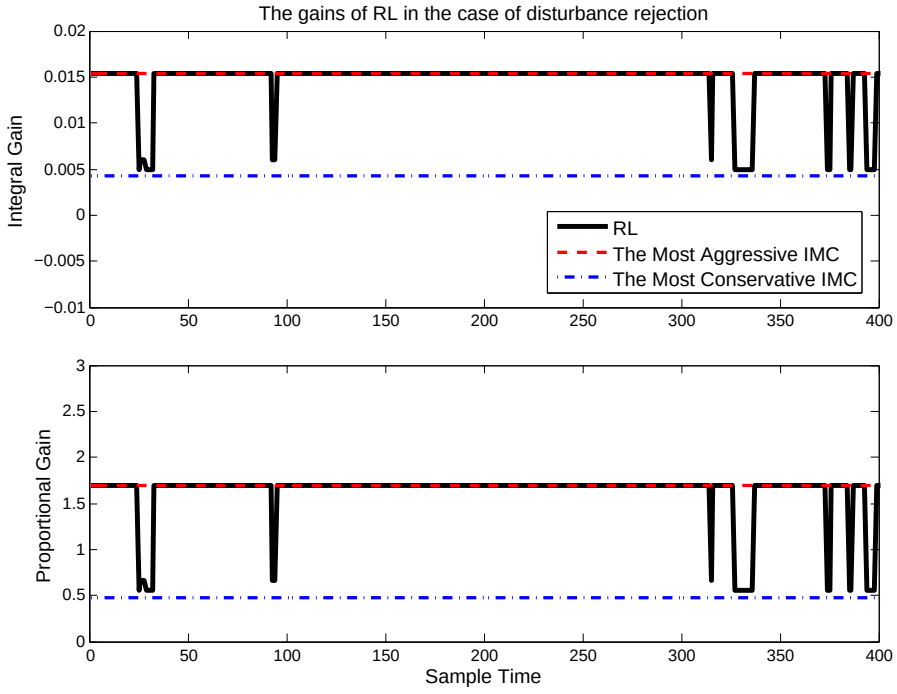

Figure 4.21: The comparison of the control signal of RL and the most aggressive, and the most conservative IMC in the case of disturbance rejection.

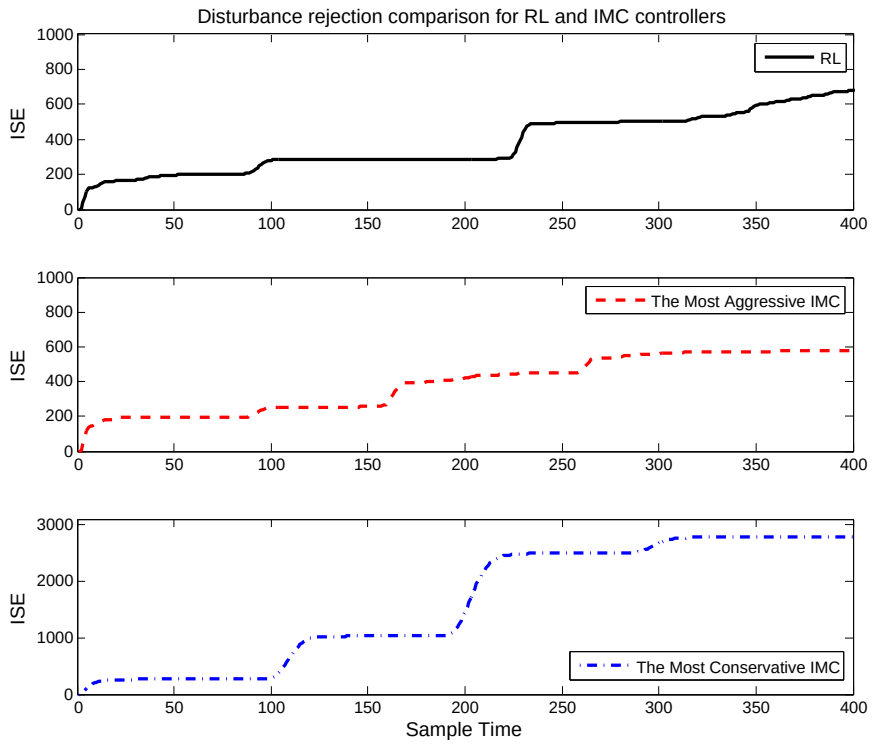

Figure 4.22: The comparison of the ISE for RL and the most aggressive, and the most conservative IMC in the case of disturbance rejection.

overshoot almost as small as the most aggressive IMC controller, while the most conservative IMC controller shows a much higher overshoot.

The second positive disturbance happens for each controller at a different time step, as a result of the different settling times that these controllers have. The second big positive disturbance is detected as the second big jump, in the output of the process. As it can be seen from the graphs, all the controllers perform well, while the RL-based PI-controller shows an overshoot smaller than the overshoot of the most conservative IMC, and bigger than the overshoot of the most aggressive IMC. Finally, the negative disturbance is introduced, which is observable in the graphs with a sudden drop in the process output. As it is demonstrated by Figure 4.19, the performance of both the RL-based, and the most aggressive IMC controllers degrades in this case, while the most aggressive IMC controller is eventually able to control the output on the set-point, unlike the RL-based PI-controller which shows fixed oscillations. The control signal and the gains applied by the RL agent during this evaluative episode are shown in Figures 4.20 and 4.21, respectively.

To numerically compare the RL-based PI-controller with the IMC ones, Figure 4.22 is presented. This figure shows the progress of the ISE value for these three controllers through the evaluative episode on disturbance rejection. As this graph shows, the RL-based PI-controller performs slightly worse than the most aggressive IMC controller, while it has a much higher performance compared to the most conservative one.

Adaptation of the RL-based PI-controller in the case of disturbance

Finally, adaptation of the RL-based PI-controller is studied in the case of disturbance rejection. For this study, first a disturbance is introduced to the system by opening the hot water valve, and setting the flow-rate of the hot water to $0.9\frac{kg}{min}$. Afterwards, the RL agent is set to learn by interacting with the process and setting the gains of the PI-controller. Figure 4.23 demonstrates, the process output of the RL-based PI-controller during this learning period. As this figure shows, at the beginning the process output is highly oscillatory, because of the introduced disturbance. After almost 1.5 hours, the RL agent adapts to the new environment and succeeds in returning the output to the set-point. The Control Signal for the same case is shown in Figure 4.24. Figure 4.25 shows the gains of the RL agent during the learning process. As this figure shows, the dominant gain completely changes when the RL-agent adapts to the new process.

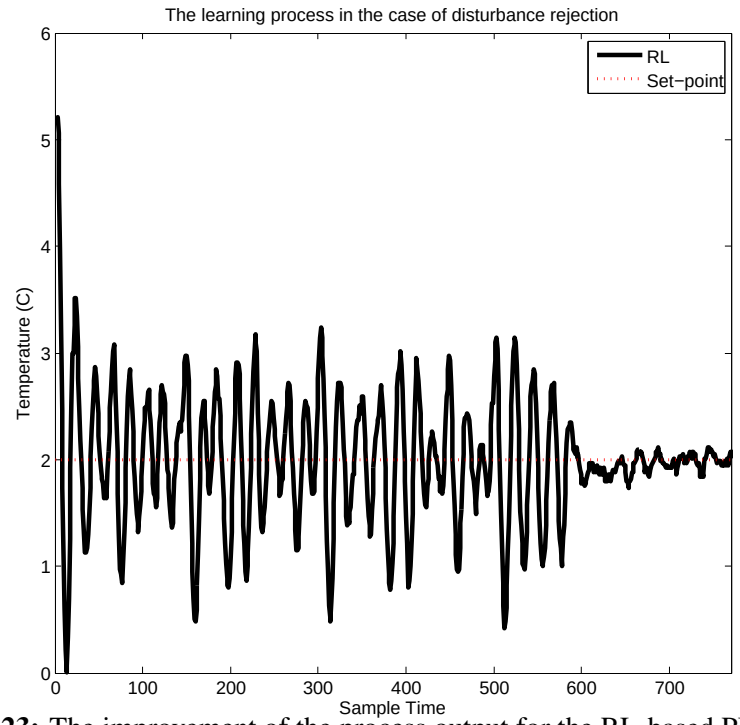

Figure 4.23: The improvement of the process output for the RL-based PI-controller through the learning process in the case of disturbance rejection.

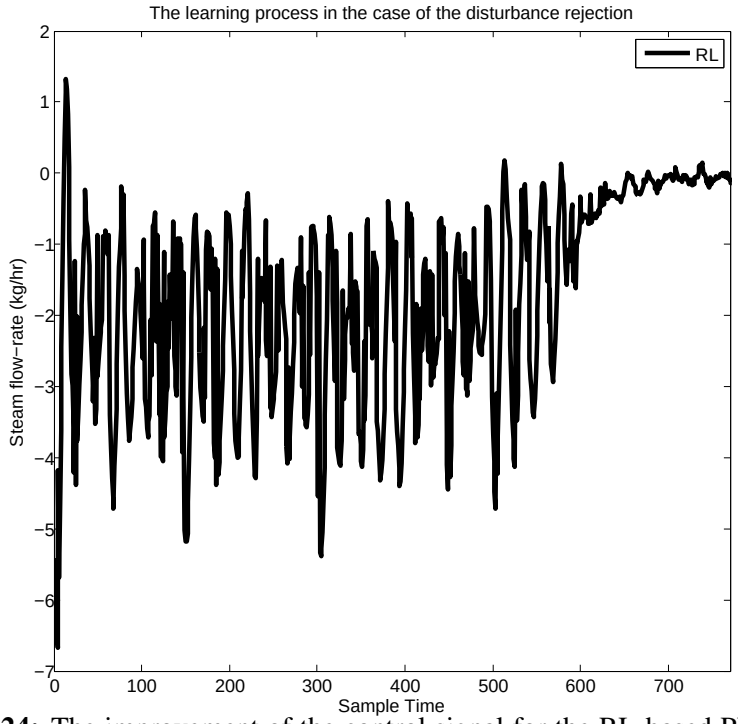

Figure 4.24: The improvement of the control signal for the RL-based PI-controller through the learning process in the case of disturbance rejection.

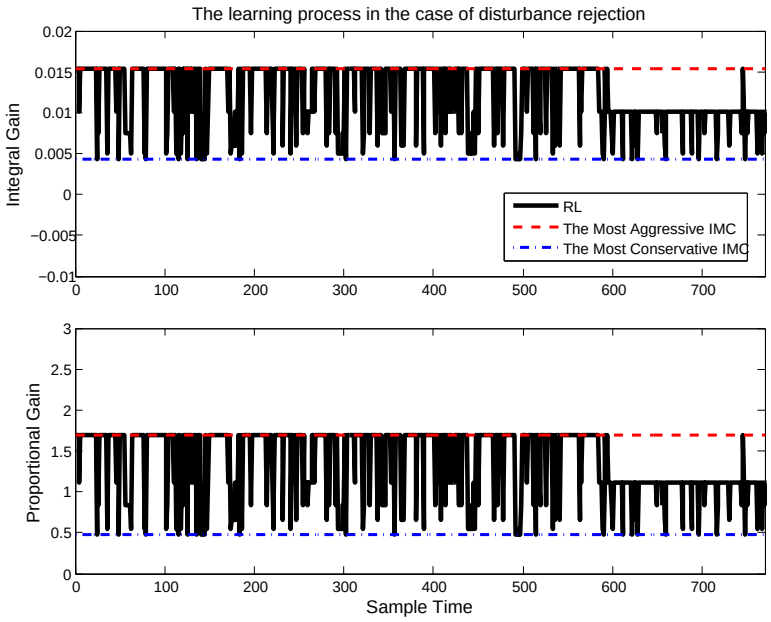

Figure 4.25: The gains of the RL-based PI-controller through the learning process in the case of disturbance rejection.

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this thesis, a self-tuning PI-controller is designed based on reinforcement learning methods. This controller is tested for set-point tracking both in simulation environment and experimentally on the real plant. Disturbance rejection studies are also carried out for the experimental validation part.

The most important characteristic of the proposed PI-controller is its adaptation to the changes occurring in the dynamics of the process. The adaptability of this controller is studied for set-point tracking and disturbance rejection. For both of these cases appealing results are obtained. Based on the obtained results, the RL-based PI-controller detects the process changes, and adapts to them in a way that a satisfactory control performance is re-achieved. This adaptation takes some amount of time depending on the severity of the change in the process, but it is eventually accomplished.

Furthermore, to adjust the model-free RL method to experimental applications and to prevent possible safety issues, an existing rough model of the process is used in forming a simulation environment. The RL procedure used in the simulation part of this thesis does not use any form of the model of the process. In this case, the RL-based PI-controller shows a performance close to the most aggressive IMC controller while it performs better than all the other IMC controllers. Afterwards, the simulatively trained PI-controller is applied to the real plant. The adaptation of this controller to the real process and through that its ability to overcome the plant-

model mismatch is observed. The only reason for importing the simulation results into the experiments is to prevent highly upsetting the plant at the beginning of the learning process. Learning from scratch on the real plant is also possible because the proposed approach is model-free, but it is not suggested due to safety issues. Therefore, the simulation results in this thesis are used to help the RL-agent start from a better initial point. As the performed experiments show, the IMC controllers do not have a high performance on the real plant, due to the existing plant-model mismatch. On the other hand, the RL-based PI-controller after being trained on the experimental setup shows a much better control performance.

Finally, the RL-based PI-controller appears to be more sensitive to the disturbances entering the process, when compared to the IMC controllers. One should not forget that the adaptability of the RL-based PI-controller overcomes this flaw. This controller eventually adapts to the process that is changed due to the disturbance, and brings the process output back to its set-point. This concept is shown in the last set of experiments provided in the experimental validation section.

5.2 Future Work

The main suggestion of this thesis for future work is the implementation of the same idea for the complete state of the system, and not just an observation of that. More studies should be performed on the function approximation, when the complete state of the system is in use. Using the complete state of the system can result in significant improvement in the learning performance of the RL agent.

Also, it is suggested that a more data-efficient reinforcement learning method is used for the same work as this thesis. The efficiency of a reinforcement learning method mainly depends on its embedded function approximation method. Function approximation methods are increasingly improving as a result of the high number of researches that are currently being conducted on this popular field in computing science. A more effective function approximator results in a more rapid learning process. Consequently, a faster adaptation to the model mismatches or process changes can be achieved. As an example, tile coding function approximation can

be studied and compared with the proposed function approximation method in this thesis.

In addition, testing the same approach for different objective functions (specially the average of reward) and different values for the discounting factor (γ) used in the RL procedure would be appealing. A Higher value for the discounting factor increases the optimization horizon of the RL agent, which may result in reducing the amount of oscillation and the size of overshoots (undershoots). Defining the objective function as the average of the reward (immediate cost) received through time may also be effective in decreasing the size of overshoots (undershoots) and improving the overall performance of the RL-based PI-controller. In addition, it is appealing to consider a case in which limitations are forced on the size of the control signal and its rate of change, or where the objective function includes logical rules. The solution to this type of problems is not as straightforward as the case in which the cost-to-go is just a function of the squared error. Hence, the RL-based approach may show a significantly better performance compared to the conventional methods.

Finally it is suggested to consider comparing the proposed approach with the conventional auto-tuning methods, such as self-tuning of PID-controllers based on transfer function estimation [Schei, 1994], or auto-tuning of PID-controllers with an online estimator combined by a static matrix precompensator [Yamamoto and Shah, 2004].

Bibliography

C. W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114, San Mateo, CA, USA, 1987. Morgan Kaufmann.

C. W. Anderson, D. C. Hittle, A. D. Katz, and R. M. Kretchmar. Synthesis of reinforcement learning, neural networks and pi control applied to a simulated heating coil. *Artificial Intelligence in Engineering*, 11(4):421–429, 1997.

C. W. Anderson, C. C. Delnero, D. C. Hittle, P. M. Young, and M. L. Anderson. Neural networks and pi control using steady state prediction applied to a heating coil. In *CLIMA2000*, page 58, Napoli, 2001. Napoli 2001 world congress.

K. J. Astrom and T. Hagglund. *Automatic tuning of PID controllers*. ISA Research Triangle Park, North Carolina, 1988.

K. J. Astrom and B. Wittenmark. *Linear Quadratic Control*, page 259. Computer-controlled systems: theory and design. Prentice Hall New York, 1996.

P. Auer and R. Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Proceedings of the 2006 Conference on Advances in Neural Information Processing Systems*, Cambridge, MA, 2007. MIT Press.

R. Bellman. Dynamic programming. *Princeton, NJ*, 1957.

R. Bellman and S. Dreyfus. Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 13(68):247–251, 1959.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1996.

D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, pages 726–731, San Mateo, CA, USA, 1991. Morgan Kaufmann.

J. Christensen and R. E. Korf. A unified theory of heuristic evaluation functions and its application to learning. In *proceedings of the Fifth National Conference on Artificial Intelligence*, volume 86, page 148152, San Mateo, CA, USA, 1986. Morgan Kaufmann.

J. H. Connell and S. Mahadevan. *Robot learning*. Kluwer Academic Pub, Boston, 1993.

R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems*, pages 1017–1023, 1996.

C. C. Delnero. Neural networks and pi control using steady state prediction applied to a heating coil, 2001.

T. G. Dietterich and N. S. Flann. Explanation learning and reinforcement learning: A unified view. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 176–184, San Mateo, CA, USA, 1995. Morgan Kaufmann.

K. Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, Jan 2000. LR: 20001218; JID: 9426182; ppublish.

T. S. Elali. *Signal Representation*, pages 2–3. Discrete Systems and Digital Signal Processing with MATLAB. CRC Press, Florida, USA, 2004.

D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel. Reinforcement learning versus model predictive control: a comparison on a power system problem. *IEEE transactions on systems, man, and cybernetics.Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 39(2):517–529, Apr 2009. JID: 9890044; 2008/12/16 [aheadofprint]; ppublish.

G. Fedele. A new method to estimate a first-order plus time delay model from step response. *Journal of the Franklin Institute*, 2008.

D. B. Fogel, T. J. Hays, S. L. Hahn, and J. Quon. A self-learning evolutionary chess program. volume 92, page 1947. Citeseer, 2004.

W. T. Fu and J. R. Anderson. From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology-General*, 135(2):184–206, 2006.

C. E. Garcia and M. Morari. Internal model control. a unifying review and some new results. *Industrial and Engineering Chemistry Process Design and Development*, 21(2):308–323, 1982.

G. J. Gordon. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*, page 261, Tahoe City, California, USA, 1995. Morgan Kaufmann.

J. B. He, Q. G. Wang, and T. H. Lee. Pi/pid controller tuning via lqr approach. *Chemical Engineering Science*, 55(13):2429–2439, 2000.

J. H. Holland. *Escaping Brittleness*, volume 2 of *The possibility of general-purpose learning algorithms applied to rule-based systems*, pages 593–623. Morgan Kaufmann, San Mateo, CA, USA, an artificial intelligence approach edition, 1986.

R. A. Howard. *Dynamic programming and Markov process*. MIT press, Cambridge, MA, 1960.

M. N. Howell and M. C. Best. On-line pid tuning for engine idle-speed control using continuous action reinforcement learning automata. *Control Engineering Practice*, 8(2):147–154, 2000.

M. N. Howell, G. P. Frost, T. J. Gordon, and Q. H. Wu. Continuous action reinforcement learning applied to vehicle suspension control. *Mechatronics*, 7 (3):263–276, 1997.

M. N. Howell, T. J. Gordon, and M. C. Best. The application of continuous action reinforcement learning automata to adaptive pid tuning. In *Learning Systems for Control (Ref. No. 2000/069), IEE Seminar*, page 2, 2000.

K. S. Hwang, S. W. Tan, and M. C. Tsai. Reinforcement learning to adaptive control of nonlinear systems. *IEEE transactions on systems, man, and cybernetics.Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 33(3):514–521, 2003. JID: 9890044; ppublish.

A. Jutan. A nonlinear pi (d) controller. *Canadian Journal of Chemical Engineering*, 67(6):485–493, 1989.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4(237-285):102–138, 1996.

M. Kashki, Y. L. Abdel-Magid, and M. A. Abido. A reinforcement learning automata optimization approach for optimum tuning of pid controller in avr system. In *Proceedings of the 4th international conference on Intelligent Computing: Advanced Intelligent Computing Theories and Applications-with Aspects of Artificial Intelligence*, pages 684–692. Springer, 2008.

A. H. Klopf. Brain function and adaptive systems-a heterostatic theory. Technical report, Air Force Cambridge Research Laboratories, 1972.

R. M. Kretchmar. A synthesis of reinforcement learning and robust control theory, 2000.

P. R. Kumar. A survey of some results in stochastic adaptive control. *SIAM Journal on Control and Optimization*, 23:329, 1985.

J. H. Lee and J. M. Lee. Approximate dynamic programming based approach to process control and scheduling. *Computers and Chemical Engineering*, 30(10-12):1603–1618, 2006.

J. M. Lee and J. H. Lee. Value function-based approach to the scheduling of multiple controllers. *Journal of Process Control*, 18(6):533–542, 2008.

A. O'Dwyer. Pid compensation of time delayed processes 1998-2002: a survey. In *American Control Conference*, volume 2, 2003.

S. R. Padma, M. N. Srinivas, and M. Chidambaram. A simple method of tuning pid controllers for stable and unstable foptd systems. *Computers and Chemical Engineering*, 28(11):2201–2218, 2004.

B. Ratitch and D. Precup. Using mdp characteristics to guide exploration in reinforcement learning. *Lecture notes in Computer Science*, pages 313–324, 2003.

H. Robbins. Some aspects of the sequential design of experiments. *Bulltein of the American Mathematical Society*, 58(5):527–535, 1952.

G. A. Rummery and M. Niranjan. On-line q-learning using connectionist system. *Engineering Department Cambridge University*, 166(Technical Report CUED/F-INENG/TR), 1994.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

T. S. Schei. Automatic tuning of pid controllers based on transfer function estimation. *Automatica*, 30(12):1983–1989, 1994.

D. E. Seborg, T. F. Edgar, and D. A. Mellichamp. *Process dynamics and control*. Wiley New York, 1989.

J. S. Shamma. Linearization and gain-scheduling. *The Control Handbook*, 1.

J. S. Shamma and M. Athans. Gain scheduling: Potential hazards and possible remedies. *IEEE Control Systems Magazine*, 12(3):101–107, 1992.

S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. *Advances in neural information processing systems*, pages 974–980–980, 1997.

W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Machine Learning International Workshop*, pages 903–910. Citeseer, 2000.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

C. Szepesvri. Reinforcement learning algorithms for mdps.

I. Szita and A. Lorincz. Reinforcement learning with linear function approximation and lq control converges. *Arxiv preprint cs.LG/0306120*, 2003.

M. Tan. Learning a cost-sensitive internal representation for reinforcement learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 358–362, San Mateo, CA, USA, 1991. Morgan Kaufmann.

G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933.

W. R. Thompson. On the theory of apportionment. *American Journal of Mathematics*, 57:450–457, 1934.

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. *Advances in Neural Information Processing Systems*, pages 385–392, 1995.

S. B. Thrun. *Efficient Exploration in Reinforcement Learning*. School of Computer Science, Carnegie Mellon University, 1992.

J. Tu. Continuous reinforcement learning for feedback control systems, 2001.

C. J. C. H. Watkins. Learning from delayed rewards, 1989.

M. Wiering. Explorations in efficient reinforcement learning, 1999.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

T. Yamamoto and S. L. Shah. Design and experimental evaluation of a multivariable self-tuning pid controller. *IEE Proceedings-Control Theory and Applications*, 151(5):645–652, 2004.

R. C. Yee, S. Saxena, P. E. Utgoff, and A. G. Barto. Explaining temporal differences to create useful concepts for evaluating states. In *Proceedings of the Eighth National Conference on AI, Menlo Park*, pages 882–888, Menlo Park, CA, USA, 1990. AAAI Press.

Z. Y. Zhao, M. Tomizuka, and S. Isaka. Fuzzy gain scheduling of pid controllers. *IEEE Transactions on Systems, Man and Cybernetics*, 23(5):1392–1398, 1993.

J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Journal of dynamic systems, measurement, and control*, 115:220, 1993.

Appendix A

Continuous Stirred Tank Heater

A.1 Process description

The experiments of this thesis are carried on the Continuous Stirred Tank Heater (CSTH) equipment (P6), located in the computer process control lab of the Chemical and Materials Engineering department of the University of Alberta. This experimental setup includes a transparent glass tank which has two inlet streams; the cold and the hot water. A steam coil is also inserted in the tank, to control the output temperature of the content of the tank. A baffle is positioned on the hot water stream line to make turbulence in the flow and hence to obtain a well-mixed load in the tank. Obtaining well-mixing is essential for assuming that the temperature inside the tank at the steady state condition is constant and equal to the output temperature. The schematic figure of this plant is shown in Figure A.1.

A.2 Process Control

There are two controlled variables in this process; The first one is the output temperature, while the second one is the level of the load inside the tank. Both of these variables can be controlled by conventional PID-controllers. In this thesis, we are only concerned about the temperature control loop. Therefore, the process is basically changed to a Single Input - Single Output (SISO) system by fixing the level of the load. The remaining controlled variable which is the output temperature is controlled by changing the flow-rate of the steam, as the manipulated variable. Furthermore, the hot water valve is completely closed through the system

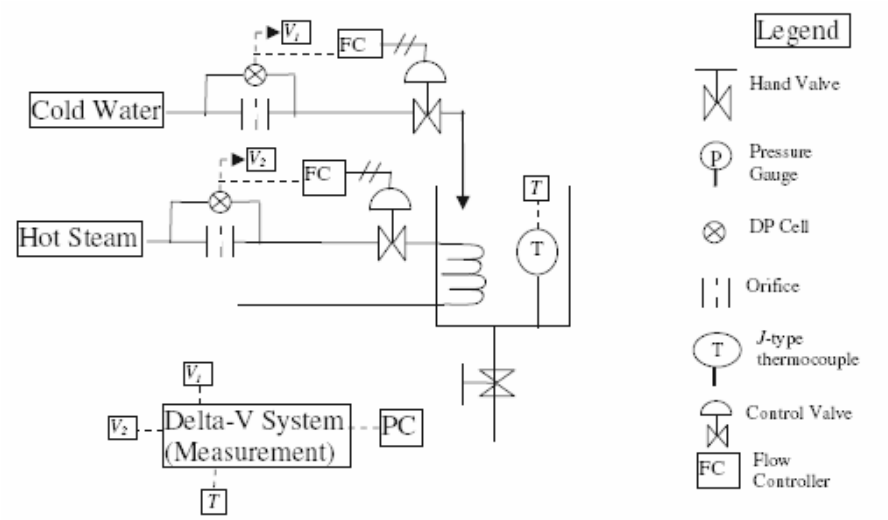

Figure A.1: The schematic figure of the experimental equipment borrowed from the related lab manual

identification process, and the set-point tracking experiments. In disturbance rejection studies, different disturbances are introduced to the system by opening the hot water valve, and setting the flow-rate of the hot water at a desired value.

This process has a time delay that can be set by choosing one of the four thermocouples placed on the downstream. These thermocouples are positioned at different distances from the exit point of the tank, therefore they result in various transportation delays depending on this distance. It should be noted that the time delay also depends on the flow-rate of the cold water. Increasing this flow-rate results in a lower dead time for the system and hence a decreased delay for it. The plant is integrated with DeltaV DCS developed by Emerson company. Through this interface, real-time interaction between the process and the user is possible. Simulink, the simulation environment developed by MathWorks, can also be used as interface to communicate with the process. The latter is what is used in this thesis to interact with the process.

PID-controllers in the process could be set into three different modes: manual, that is used to have the process in the open-loop condition; auto, which is the closed-loop format in order to control the process around a specific set-point, and cascade, in which the slave loop is closed but its set-point is set by a master control loop. Our experiments use a cascade loop to maintain a fixed level of load in the tank.