Reinforcement Learning-based Process Control Under Sensory Uncertainty

by

Oguzhan Dogru

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Process Control

Department of Chemical and Materials Engineering

University of Alberta

Abstract

Process industries involve processes that have complex, interdependent, and sometimes uncontrollable/unobservable features that are subject to a variety of uncertainties such as operational fluctuations, sensory noises, process anomalies, human involvement, market volatility, and so forth. In the face of unpredictability, industrial applications strive to exhibit consistent operational excellence in terms of product quality, economic benefits, process safety, and environmental sustainability. These operational criteria necessitate intelligent solutions to a wide range of operations that can be enhanced without requiring substantial modelling effort. Reinforcement learning (RL), as a data-driven method, can provide a practical answer to such issues by employing various types of sensory information.

A robust interface tracking algorithm is the first contribution of this thesis. In contrast to existing methods, which rely on hand-crafted features, the proposed algorithm provides a tracking methodology comprised of convolutional neural and long short-term memory networks that are jointly optimized for interface tracking. Without any explicit models or restrictive assumptions, this structure integrates neighbouring spatial and spatiotemporal elements. Unlike supervised/unsupervised learning methods, the proposed RL-based tracking algorithm requires only a few images that can be labelled quickly and accurately by a user or a sensor. This agent outperforms some of the existing methods in terms of robustness, which is one of the most important requirements in state estimation and control. Finally, by employing a dimensionality reduction technique, our work contributes to deep learning-based RL solutions.

The second contribution aims to develop an RL-based safe controller while taking safety requirements into account. The suggested approach combines a deep actor-critic agent with random setpoint initialization and a Lagrangian-based soft-constrained learning scheme to achieve this goal. The example demonstrated that the soft-constrained approach can provide smooth state transitions while accelerating the offline training phase with several workers. In addition, an exploration metric inspired by the set theory was developed.

The third contribution takes into account the constrained uncertain reward/cost function, which is often employed in RL and process control. A reduced signal-to-noise ratio in a process can permanently deteriorate the control policy and result in poor tracking/control performance. Taking sensory noise into account, the proposed method models the reward/cost function as a dynamic process, along with transition and observation models. Using a constrained particle filter, the proposed method estimates the first and second moments of the constrained reward.

The fourth contribution addresses the problem of dimensionality increase during online skew state estimation. Although a closed skew-normal distribution increases the degree of freedom in state estimation, its location and scale parameters increase in size at the end of each filtering stage. This problem slows down the inferential calculation, making closed-form solutions impractical and online inference infeasible in the long term. With the rigorous formulation of dimensionality reduction as an optimization strategy, empirical analyses were carried out to compare various statistical distance functions and optimization techniques. Finally, the proposed skew estimation scheme was applied to problems involving reward estimation and state estimation.

The fifth contribution proposes an autonomous PID tuning scheme. Since complex industrial plants can utilize thousands of control loops with unknown models, tuning the PID controllers can be time-consuming. This algorithm is based on a constrained contextual bandit that tunes the PID controllers starting with step-response models and gradually learning the plant model mismatch through online interaction.

The sixth contribution is the development of an autonomous MPC tuner and its integration with an autonomous advanced control infrastructure. Although various traditional approaches may design MPC parameters offline, there can be significant performance deterioration due to model plant mismatch or operational changes. Additionally, establishing specific performance criteria using complex functions can be difficult. However, using smart trial-and-error, the proposed RL agent can produce optimal solutions to such challenges. This modular, model-independent agent, can be pre-trained on step-response models and then integrated into more complicated schemes. By integrating all agents, controllers, and filters, this contribution also includes a proof of concept of an autonomous process control scheme.

Preface

This thesis is original work conducted by Oguzhan Dogru under the supervision of Dr. Biao Huang and is funded in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. Portions of the thesis have been published in peer-reviewed journals. This thesis is written in paper format. Some of the chapters may have some overlaps, which is to ensure each chapter is self-contained.

1. Chapter 3 of this thesis has been published as: **O. Dogru**, K. Velswamy and B. Huang, "Actor–Critic Reinforcement Learning and Application in Developing Computer-Vision-Based Interface Tracking," *Engineering*, vol. 7, no. 9, pp. 1248-1261, 2021.

2. Chapter 4 of this thesis has been published as: **O. Dogru**, Wieczorek, N., Velswamy, K., Ibrahim, F., and Huang, B., "Online reinforcement learning for a continuous space system with experimental validation," *Journal of Process Control*, 104, 86-100., 2021.

3. Chapter 5 of this thesis has been published as **O. Dogru**, R. Chiplunkar and B. Huang, "Reinforcement learning with constrained uncertain reward function through particle filtering," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 7, pp. 7491-7499, 2021.

4. Chapter 6 of this thesis has been submitted to *IEEE Transactions on Industrial Informatics* as **O. Dogru**, R. Chiplunkar and B. Huang, "Skew Filtering for Online State Estimation and Control", 2022.

5. Chapter 7 of this thesis has been published as **O. Dogru**, K. Velswamy, F. Ibrahim, Y. Wu, A. S. Sundaramoorthy, B. Huang, S. Xu, M. Nixon and N. Bell,

”Reinforcement Learning Approach to Autonomous PID Tuning”, *Computers & Chemical Engineering*, vol. 161, p. 107760, 2022.

Dedicated to

My adorable family, Aslihan, Zeynep and Aydogan.

Acknowledgements

This thesis is the work of many students and professionals, including my supervisor, Dr. Biao Huang, who provided me with an outstanding chance in one of the top research organizations on the planet. Our technical talks, as well as his insightful remarks, pushed me out of my comfort zone, broadened my viewpoint, and taught me to embrace and overcome problems of all kinds. His contagious enthusiasm, wisdom, and unique viewpoint enhanced my admiration for the control sector and aided me in completing novel projects. Thanks to his tremendous support, I was able to find remarkable opportunities to work with a diverse group of exceptional individuals throughout this extraordinary experience. From organizing and presenting at professional workshops to training high-performing co-op students, collaborating with other groups' members, and developing practical applications in our advanced control laboratory, Dr. Huang provided me with numerous means for this one-of-a-kind training, which I am indebted for.

I am grateful for my supervisory committee, Dr. Jinfeng Liu and Dr. Zukui Li, who immersed me in challenging questions and further refined my research output.

My interactions with exceptional people elevated my career during my adventurous journey. Words are insufficient to describe my gratitude to Dr. Kirubakaran Velswamy, Dr. Fadi Ibrahim, Jingyi Wang, Dr. Hongtian Chen, Dr. Ranjith Chiplunkar, Camila Santander, Arun S. Sundaramoorthy, and Dr. Junyao Xie. We had fruitful discussions with magnificent people like Agustin, Rui, Faraz, Aswathi, Anu, Dr. Mengqi, Dr. Rahul, Dr. Jayaram, Dr. Santhoosh, Dr. Seshu, Yousef, Dr. Xunyuan, Mia, Alireza, Krishna, Cameron, Dr. Purushottama, and many others. My lovely friends Ali, Arul, Arzu, Bedir, Beril, David, Dr. Desiree, Dr. Eyup, Dr. Felix, Ersal, Etem, Leyla, Sagar, and Yagmur softened Edmonton's harsh weather with their presence. Our precious co-op students Tanvir, Faris, Yvonne, Armianto, Radomir, Darpan, Nathan, Yuqi, Antony, Bowen, Igor, and others have been an essential part of our work and some of our scientific publications. Last but not least, I thank Terry Runyon and Les Dean, crucial team members of ours, for constantly supporting me and our group activities.

I want to acknowledge the financial support from Natural Sciences and Engineering Research Council (NSERC) of Canada. I also express my gratitude to the University of Alberta and the Department of Chemical and Materials Engineering for providing financial support and other resources to conduct research.

I appreciate the support of Dr. Selis Onel and Dr. Zumriye Aksu, who helped me start this journey. Dr. Erdogan Alper familiarized me with the concepts of optimization, process control, and dynamic programming that have been central to my studies. He supported me even in his final days on earth. Without their encouragement, neither my work nor this thesis would exist.

Finally, I appreciate my family's love, support and sacrifices.

These people and my experience showed me the significance of communication, dedication, discipline, hard work, observation, patience, sharing, and love in life.

Contents

List of Tables

List of Figures

List of Algorithms

Glossary

Abbreviations

ANN	Artificial neural networks
CDF	Cumulative distribution function
CNN	Convolutional neural network
ConvLSTM	Convolutional Long short-term memory network
CSN	Closed skew-normal distribution
KL	Kullback–Leibler
LSTM	Long-short term memory network
ML	Machine learning
PCA	Principal component analysis
PDF	Probability distribution function
ReLU	Rectified linear unit
RL	Reinforcement learning
RMSE	Root mean squared error
SL	Supervised learning
SSE	Sum of squared errors
UL	Unsupervised learning

Notations

$\langle\cdot\rangle$	Tuple
A	State evolution matrix
Δ	Fifth parameter of the CSN
$diag(\cdot)$	Diagonal elements of $(\cdot)$ or Diagonal matrix form of vector $(\cdot)$
$\mathbb{E}[\cdot]$	Expectation operation
$\exp(\cdot)$	Exponential operation
Γ	Skewness parameter of the CSN
K_t	Kalman gain matrix
$\ln(\cdot)$	Natural logarithm
μ	Mean of a Gaussian distribution or the first parameter of the CSN
$\mathcal{N}(\cdot)$	Normal (Gaussian) distribution
ν	Fourth parameter of CSN
$p(\cdot)$	Probability distribution
$\Phi(\cdot)$	Cumulative probability distribution of a Gaussian variable
$Q(x,u)$	Action-value function
$V(x)$	State-value function
θ,ω	Neural network parameters
t	Time step

Chapter 1

Introduction

Process industries involve processes with complex, interconnected, and sometimes difficult-to-control or to-observe properties that are subject to numerous uncertainties such as operational variations, sensory noise, process abnormalities, human involvement, market fluctuations, etc. Considering product quality, economic benefits, process safety and environmental sustainability, industrial applications aim to demonstrate continuous operational excellence in the face of uncertainties. These operational criteria necessitate smart solutions to a vast variety of processes that can be improved through an autonomous scheme with various filters, state estimators, and controllers. However, such an ideal solution requires extensive modelling of each process unit, which is not realistic since the real-world problem might be too complex to be modelled explicitly. As a data-driven method, reinforcement learning (RL) can provide a practical alternative to such challenges by utilizing different types of sensory information. This chapter gives an overview of the essentials in the existing literature, as well as the reasoning for the proposed autonomous industrial control approaches.

1.1 Motivation

Process monitoring and control are necessary for the safe, optimal and eco-friendly operation of an industrial process. Such operations may be achieved through first-principle process modelling for accurate behaviour prediction. However, it can be challenging to develop detailed first-principles models due to process complexities.

Although data-driven models provide a practical and perhaps more convenient alternative to developing first-principle models, the predictive abilities of the model, and hence the performance of a controller that uses this model, will depend heavily on the data quality. In addition, differences in process behaviour, types of uncertainty and operational frequency add another layer of complexity to data-driven model developments. This thesis develops robust and safe RL agents in the face of unknown system dynamics and in the presence of process and sensory uncertainty. These topics will be covered in depth in the following subsections.

1.1.1 Operational Variations

Industrial processes utilize numerous raw materials with varying feed compositions that need to be continuously processed. In addition, the physical environment, seasonality, multi-modality, human interaction, and higher-level decisions made by the planning departments create operational variations that can impact the environment, safety, and process economics. Fig. 1.1 shows a typical gain change in an auto-regressive process due to changing operational conditions. Such gain changes can impact the controller performance, resulting in reduced operational performance if not addressed properly.

An autonomous scheme should either be robust to these changes or adapt to them to achieve smooth and effective operation. For robustness, classical methods model as many scenarios as possible during the design of model structures. Some design options include employing multi-modal or nonlinear models or using stochastic models to evaluate various operational scenarios [1]. Alternatively, modern techniques involve machine learning methods that utilize complex functions to approximate the system model/controller with the help of diverse data points. The intuition behind this approach is to maintain the reliability of the approximate function through 'learned experience'. Below are some examples to highlight the difference between classical and modern techniques. Consider an operator that recently joined the organization to control a reactor without prior reactor operation knowledge.

1. Classical techniques would suggest describing the process behaviour under certain operating conditions by using process flow diagrams and physical rules.

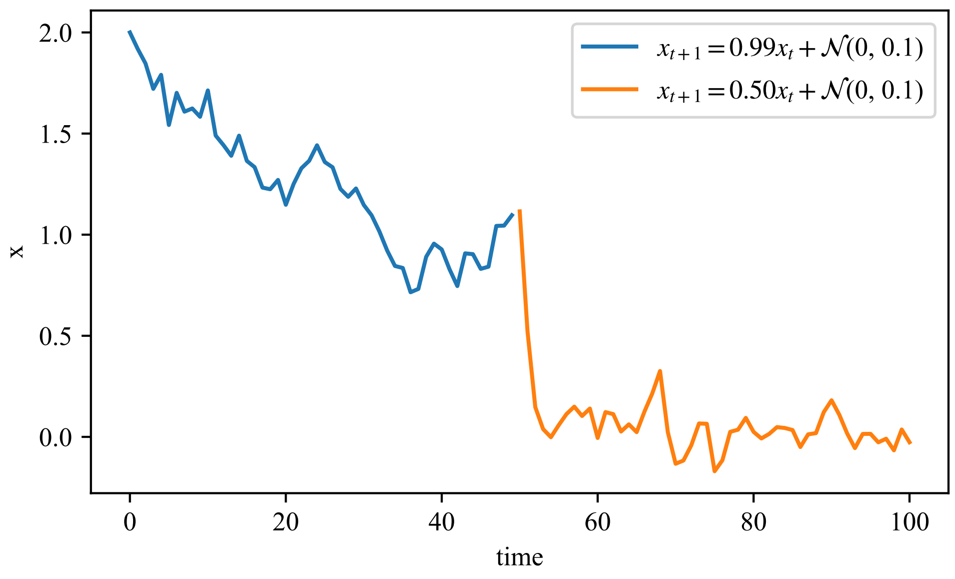


Figure 1.1: Time-series evolution of a state, x, with a gain change due to operational conditions at $time = 50$. Note that the slope of the process drastically changes due to the gain change.

This approach is equivalent to designing a controller/estimator for multiple operational modes.

2. Modern (learning-based) techniques would suggest showing the historical input-output data to the operator and asking the operator to make predictions and update them according to the new data when needed. This approach is equivalent to identifying a system model using data.

Both these techniques have limitations when unknown circumstances are encountered. For example, classical techniques might fail since their strict assumptions and unknown models might not tolerate uncertainty in novel circumstances, and machine learning techniques might fail due to their unrestricted update mechanisms or if their objective is not defined appropriately. A third technique would be combining the physical rules, process diagrams (including causal networks), and data to obtain a hybrid methodology. This thesis, however, will not expand on the hybrid methods.

1.1.2 Sensory Uncertainty

The most important element of data-driven modelling/learning or model update is the process data that is obtained through various types of sensors. A sensory device converts a physical phenomenon into a representation that an operator can evaluate [2]. The representation can be numerical or physical. For example, the lighting condition in a room can be measured through cameras, the pressure of a liquid tank can be measured through digital differential pressure sensors, the temperature of a gas tank can be measured through an analogue manometer, and the acidity of a liquid can be measured through a pH probe or pH strips. Each of these sensors/indicators has different types of uncertainty associated with them. For example, a pressure sensor attached to a tank can have static noise, and a test involving a pH strip will be affected by human interaction. Although there are various types of sensors, this thesis focuses on the digital ones that can connect to an electronic device (e.g., a computer) for further processing.

The properties and sources of sensory uncertainty depend on the type of sensors. For example, vibrations, temperature, the material of the sensor, electrical/magnetic interference, and deposition can have drastic impacts on measurements obtained from the sensors. Such effects change the statistical properties of a signal obtained from the process. Fig. 1.2-a shows different types of noise obtained during operation. Fig 1.2-b shows that the underlying signal value (shown as the blue line) has a sharp peak without any uncertainty. However, noise reduces the reliability of the sensor by increasing the variance. For efficient control and autonomous operations, such sensory noise should be addressed by appropriate methods.

1.2 Background Literature

This section provides background material on the algorithms described in the thesis. The literature review constitutes operational variations and sensory uncertainty, as motivated in the previous section.

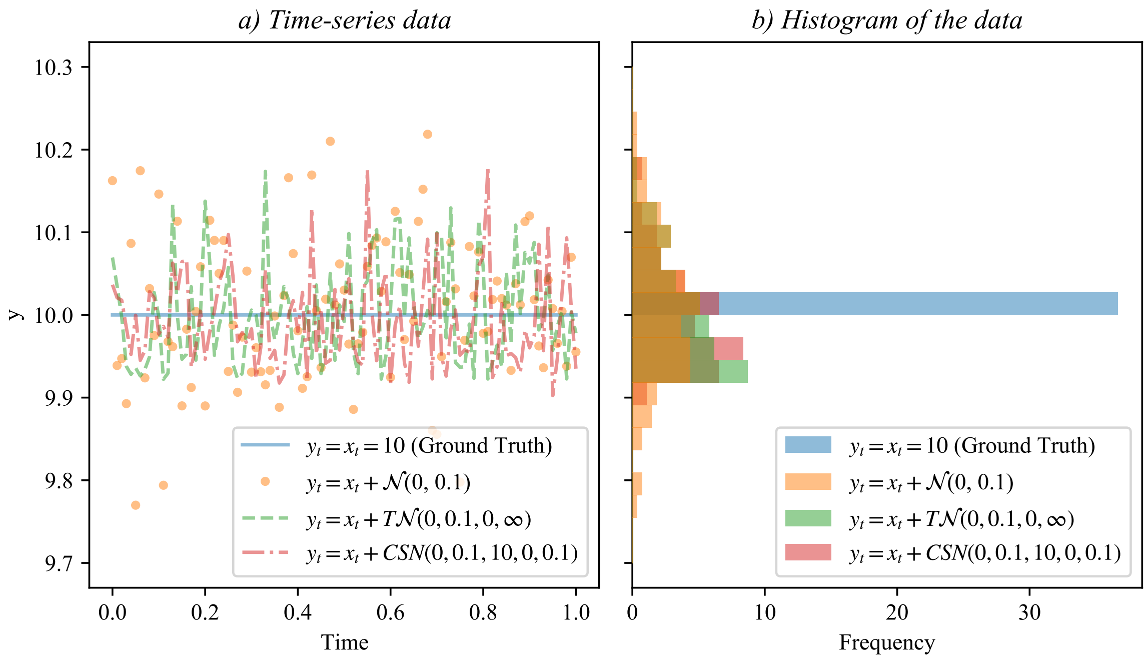


Figure 1.2: Different types of sensory noise in time-series data. a) $y_t = x_t = 10$ represents the ground truth signal (ideal case without noise), the orange dots show the signal with Gaussian noise, the green dashed line represents the signal with Truncated normal noise, and the red dotted dashed line shows the signal with closed skew normal noise. b) Histogram of the time-series data. Note that the sharp blue peak shows the ideal signal has a mean value of ten without uncertainty. The other signals need to be processed to estimate the unknown ground truth.

1.2.1 Operational Variations

Classical techniques address operational variations, nonlinearity and multi-modality by using multiple linear models, or linear parameter varying models with gain scheduling [3, 4]. In these approaches, a system/controller is modelled/designed through multiple models or a single model that varies as a function of a scheduling variable. The scheduling variable can vary according to time, state, or other indicator variables. For example, if there are seasonality effects on the process of interest due to temperature/lighting conditions, time can be used as an indicator. On the other hand, the operating points (e.g., different liquid levels) can have significant impacts on process dynamics if the system gain depends on them. In this case, the current operating point can be used as the scheduling variable. If these multiple models or gain scheduling result in discontinuities in the state/action space, there can be undesired sudden jumps in the process variables. Therefore, they should be designed and smoothened carefully. Alternatively, nonlinear models can be used to address the above-mentioned abruptness challenges [5]. For example, neural networks have shown promising results over the past decade in terms of model accuracy and generality [6]. In addition to these modelling approaches, which can be applied to controllers through modifications, automatic update mechanisms can improve resilience against process uncertainty. For example, various PID auto-tuners have attracted industrial attention due to their practicality [7]. However, these approaches have not considered the saturation problem.

Recent developments in process control and computer science have shown that reinforcement learning with nonlinear functions (such as neural networks) can be an alternative to the above-mentioned approaches [8]. A reinforcement learning agent corresponds to a decision-maker in classical control/optimization to improve the current status of the process. The agent generates an action for the current state to maximize future rewards while dynamically interacting with the process. The information obtained during this interaction is used to improve a control policy through Bellman's optimality criteria [9]. Often, the information constitutes observed reward and state values, which makes the agent superior to model-based approaches like dynamic programming since the agent is process-agnostic. However, model-free agents are sensitive to data, policy and reward design due to process uncertainties. This thesis will propose to regularize the agent's behaviour through prior knowledge of the process to achieve effective autonomous learning infrastructures.

1.2.2 Sensory Uncertainty

Sensory uncertainty can be addressed through hardware or software adjustments. For example, accumulating dust on a level measurement sensor in a dusty tank can reduce accuracy. In this case, one should service the hardware. On the other hand, inevitable signal noise affects sensors, which can be reduced through a set of calculations (i.e., filtering software). These calculations depend on the properties of the sensor, process, sensor design, and other interferences, which influence the noise statistics. The most straightforward and practical approach is to use a moving average filter [9] that updates the current estimation based on the previous one and the sensory measurement (instead of using the sensory measurement directly). However, this averaging method cannot assess the uncertainty associated with the measurement and can result in poor performance. Bayesian filters and smoothers can provide more accurate estimates through rigorous mathematical calculations while incorporating prior process knowledge [10]. However, their online implementation is computationally expensive. Kalman filtering converts the infinite-horizon Bayesian filtering into a tractable estimation through Markovian assumptions [11]. Although Kalman filtering is a remarkable technique that can be used in numerous industrial processes, its performance deteriorates if the distribution of the measurement differs from the Gaussian distribution assumption. For example, sensory interference can skew the measurement distribution, or mathematical operations (e.g., constraint enforcement) can truncate the distribution, thus affecting the statistics of the variables of interest. Fig 1.3 shows various distributions, including the ones without the normality and symmetry properties. In the presence of skewed noise, filters can utilize distributions such as skew-t [12], Gumbel [13], normal-skew mixture [14], etc. However, these filters normally require high computation. Closed skew-normal distribution (CSN) is another skewed distribution, which generalizes the Gaussian distribution with additional skewness, location and scale parameters [15]. Its closure property under linear transformation and the Bayesian rule makes it a viable alternative to the Gaussian distribution in the presence of skew measurements. In addition to these sensory challenges, mathematical operations (like constraint utilization) can result in asymmetric distributions like the truncated normal distribution, which have been addressed through various filters [16,17]. In addition to these classical techniques, neural networks have been used to address sensory uncertainty without explicit models [18]. In addition to noise reduction, recursive neural networks can provide robustness in the presence of missing data, bounded noise, and visual uncertainties in a camera application [19]. Considering these criteria, optimal filtering of the process measurement is a necessary element of robust and autonomous control.

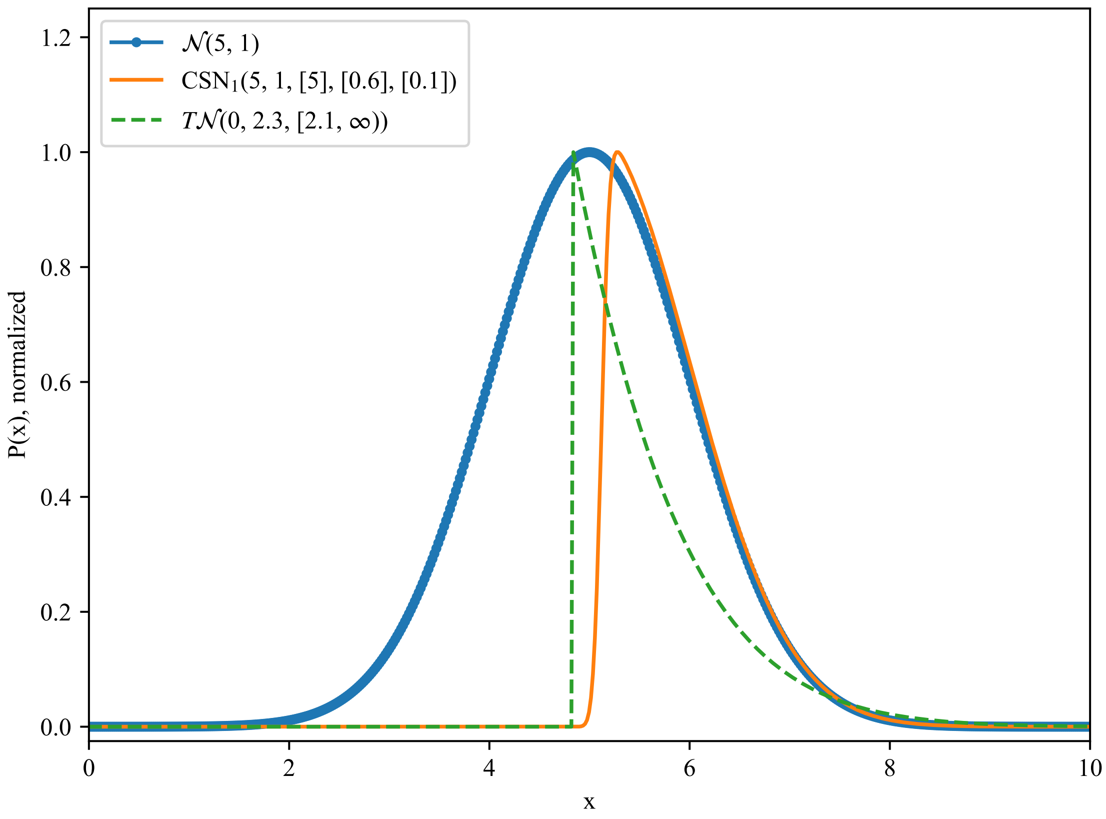


Figure 1.3: Illustration of the normal ($\mathcal{N}$), closed skew-normal (CSN), and truncated normal ($T\mathcal{N}$) distributions. The normal distribution is often used due to its convenience during mathematical derivations. However, sensory interference can result in skewed distributions, which is illustrated using the closed skew-normal distribution, and mathematical operations can yield variables with truncated normal distribution. Kalman filtering can fail when properties like normality and symmetry are no longer valid.

1.3 Thesis Outline

Following the presentation of motivation and the analysis of background material in Chapter 1, the thesis continues by discussing each of the problems and their solutions in the following chapters.

The literature review on actor-critic reinforcement learning algorithms and a robust interface tracking technique are discussed in Chapter 3. The presented approach focuses on the instrumentation level, which is the lowest level of the control hierarchy. The suggested approach offers an end-to-end interface tracking methodology by using a convolutional long short-term memory network. This structure integrates the nearby spatial and spatiotemporal aspects without any explicit models or restrictive assumptions. The proposed RL-based tracking algorithm requires significantly fewer images than supervised/unsupervised learning techniques, and these images can be quickly labelled by a user (manually) or a low-uncertainty sensor such as a pressure sensor (automatically). In terms of robustness, which is one of the key requirements in state estimation and control, this agent performs better than some of the existing techniques. Finally, by employing a dimensionality reduction technique to demonstrate the value functions of the high-dimensional states as a performance indicator, our study contributes to deep learning-based RL solutions.

Chapter 4 aims to create an RL-based safe controller while taking into account operational constraints such as safety requirements. To accomplish this, the suggested method combines a deep actor-critic agent with random setpoint initialization and a Lagrangian-based soft restricted learning scheme. The experimental studies show that the soft-constrained approach can provide smooth state transitions with low variance in control actions, while several workers accelerate the offline training phase. In addition, an exploration metric based on set theory was proposed.

Chapter 5 focuses on the constrained nature of the uncertain quadratic/non-quadratic cost function employed in RL and process control. This chapter demonstrates how lowering the signal-to-noise ratio in a process irreversibly degrades tracking/control performance. Taking sensory noise into account, the proposed method models the reward/cost function as a dynamic process, along with transition and observation models. The proposed method uses a constrained particle filter to estimate the first and second moments of the constrained reward. Furthermore, to save computational time, the constrained estimations are calculated in several threads of a computer's processing unit.

Chapter 6 analyses the dimensionality increase problems in online state estimation by using skew distributions. Although a closed skew-normal distribution enhances the degree of freedom in state estimation, its position and scale parameters increase in size at the end of each filtering phase. This challenge slows down the inferential process, making closed-form/analytical formulations impossible to derive and online inference impractical over time. Following the formal formulation of dimensionality reduction as an optimization strategy, empirical studies were carried out to compare various statistical distance functions and optimization techniques. Finally, the proposed skew estimation scheme was applied to problems of reward and state estimation.

Chapter 7 offers a practical solution to the PID tuning problem. Because complex industrial plants can have thousands of control loops of varying types, calibrating PID controllers can be a time-consuming task with unknown system behaviour. Inspired by the actor-critic paradigm, this chapter designed a constrained contextual bandit that autonomously adjusts the PID controllers using simple step-response models. The proposed agent was deployed in a distributed control scheme after a preliminary offline training phase to learn the model plant mismatch through online interaction. This chapter also presents a methodology for MPC tuning by adjusting the weight parameters in the MPC objective function.

Chapter 8 focuses on linking the methodologies and theories developed in the previous chapters and integrating them into an autonomous control automation infrastructure. Implementing non-linear/non-quadratic functions or modifying performance criteria using complex functions might be complicated. The proposed autonomous control automation infrastructure may provide optimal solution alternatives to such issues through intelligent trial-and-error. Because of the modular nature of the infrastructure and the model-agnostic character of the agents, these agents can be trained on simulated system models and integrated into the infrastructure, highlighting the versatility of the methodologies developed in this thesis. This chapter also demonstrates the robustness of the proposed methodologies in the presence of various uncertainties and disturbances in an experimental setup. Furthermore, challenging aspects of autonomy and automation are discussed in this chapter.

Chapter 9 outlines the findings drawn from the various proposed models and algorithms. This chapter also discusses potential future work.

1.4 Main Contributions

The main contributions of the thesis are outlined in the following points.

1. An RL-based robust interface tracking methodology that utilizes convolutional networks and actor-critic technique is proposed. The resulting algorithm is tested against numerous elements of visual disturbance.

2. An RL-based low-level process control method with soft constraints for safety is developed. The proposed algorithm is implemented in a pilot-scale experimental setup.

3. An online and constrained particle filter for robust state estimation under truncated state and measurement noise is developed. The proposed algorithm is integrated into reward estimation in an RL problem.

4. An online and skew Kalman filter for robust state estimation under skewed measurement noise is rigorously derived and implemented in real-time estimation and control.

5. An autonomous PID tuner with operational soft constraints is developed and implemented in a pilot-scale distributed control system in real-time.

6. An autonomous MPC tuner is proposed and integrated into a complex pilot-scale control infrastructure.

7. A proof of concept of an autonomous process control system through a pilot-scale experiment setup by combining various proposed algorithms.

Chapter 2

Mathematical Background

This chapter provides a review of the various algorithms employed in the methods proposed in this thesis. As discussed, although each proposed method creates an individual element of an autonomous scheme, their combination forms a robust decision-making scheme. Chapter 3 develops an actor-critic interface tracker that uses convolutional and long short-term memory networks, and hence mathematical foundations of each of these are presented in detail. Chapters 3, 4, 5, and 7 use the actor-critic reinforcement learning methodology, whose mathematical background is described here. Chapter 6 develops an online skew filter, and this chapter introduces its offline counterpart and Kalman filtering. Chapters 6 and 7 involve model predictive controller, which is also presented in this chapter.

2.1 Regression Analysis and Neural Networks

An example of pattern recognition is curve fitting, as shown in Fig. 2.1, where the goal is to estimate the pattern in the blue dots' sinusoidal behaviour that is given in Eqn. (2.1).

$$y = sin(3\pi x) + \mathcal{N}(0, 0.05) \tag{2.1}$$

where $\mathcal{N}(0, 0.05)$ represents a Gaussian variable with mean zero and standard deviation of 0.05. After observing these noisy data points, one can develop a mathematical model to express the noise-free behaviour of the sinusoidal wave. In this example, a nonlinear model structure is selected to highlight the importance of prior knowledge.

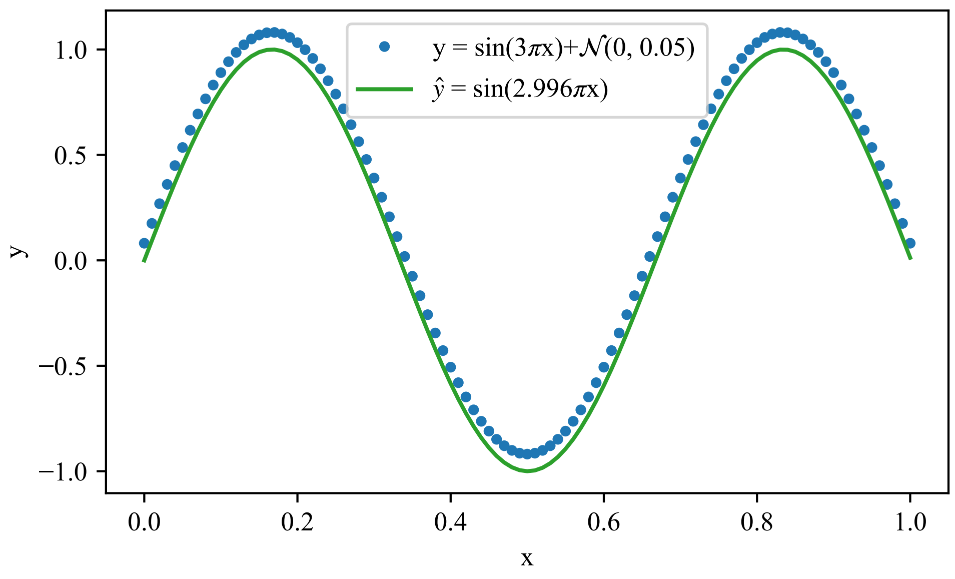


Figure 2.1: A prediction for $y = sin(3\pi x) + \mathcal{N}(0, 0.05)$. The green curve shows an estimation due to accurate model selection.

This model is in the form of $\hat{y} = sin(A\pi x)$, where A represents the parameter(s) to be estimated. This estimation can be done by minimizing an error function that indicates the disparity between the function $\hat{y}(x, A)$ for any A and the data points present in the training set. A straightforward choice of error function is given by the sum of the squares of the error (SSE) between $\hat{y}(x, A)$ for each data point x, and the corresponding target values y. Such an error function is shown in Eqn. (2.2)

$$\varepsilon(A) = \frac{1}{2}\sum_{n=1}^{N}[\hat{y}(x_n, A) - y_n]^2 \tag{2.2}$$

where 1/2 is introduced for convenience in the calculation. This error function is non-negative and would be zero if $\hat{y}(x, A)$ strictly passes through each data point, y. Following these properties, A can be found by using an optimization method. The most primitive way is to randomly choose an A value, evaluate Eqn. (2.2), and repeat it until a desired error value $\varepsilon(A) < \delta$ is achieved, where δ is a user-defined threshold. However, randomizing the value of A can be time-consuming, and more efficient methods can result in satisfactory accuracy. For example, gradient descent can update A in the direction of the negative gradient of the error value as shown in

Eqn. (2.3).

$$A_{t+1} = A_t - \eta_t \nabla_A \varepsilon(A_t) \tag{2.3}$$

where $t \geq 0$ is the iteration step, $\eta_t > 0$ is called the learning rate, and A_0 is often initialized randomly such that $\varepsilon(A_0) \neq 0$. This recursive calculation continues until $A_{t+1} \approx A_t$, which results in a locally optimal point in the parameter-error space. Following this numerical optimization, $A_{final} \approx A_{final-1} = 2.996$ can be found as shown in Fig. 2.1. Despite an insignificant disparity, due to numerical difference, the green line captures the overall trend of the observed data points. This example shows that a parameterized function can satisfactorily explain the underlying behaviour of $y = sin(3\pi x)$ given an input data point, x. Moreover, iterative numerical methods (like gradient descent) can be used to adapt to small variations over time, where the true value of A (i.e., $A_{true} = 3$ in the abovementioned example) is non-stationary.

However, as mentioned earlier, these model structures might be unknown in complex problems. In such cases, one can parameterize another function, $\hat{y} = f(A^T, x)$, where the transpose operator indicates that A is a set of parameters (instead of a scalar), and f is a continuous, differentiable and nonlinear function with a user-defined arrangement in A, as shown in Fig. 2.2. The structure of the neural network and its optimization function can vary according to implementation. For example, convolutional networks provide more robust and computationally efficient solutions, and recursive (e.g., long short-term memory) networks offer more powerful feature representations compared to vanilla neural networks.

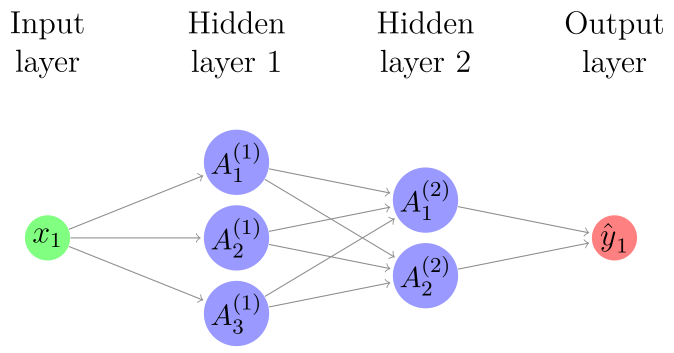


Figure 2.2: A neural network structure with an input (x), five weights (A), and an output ($\hat{y}$). The subscripts indicate the identity of the weight, and the superscripts stand for the layer identity.

2.1.1 Convolutional Neural Networks

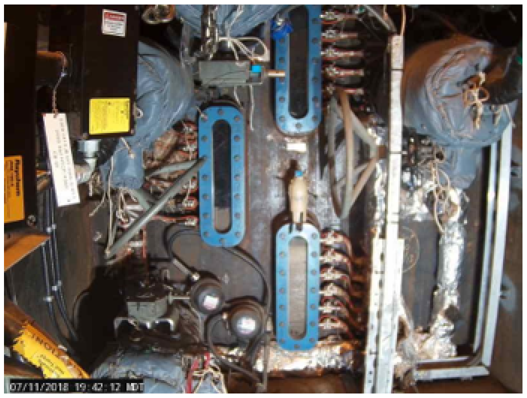
Process industries use various types of sensors such as for pressure, temperature and image measurements. This variety makes designing a unified monitoring/control structure challenging since the numerical values obtained through these sensors can vary. For example, an ideal thermocouple can measure any temperature from 0K to ∞K. However, an RGB image is a graphical representation of three matrices with each element representing the intensity of red, green, and blue colour channels. Each of these channels includes an intensity value that ranges between 0-255 for an 8-bit sensor. Fig. 2.3a demonstrates a colour image of an industrial plant, and Fig. 2.3b explicitly shows the numerical values of the red, green, and blue channels of that image. Additionally, the temperature obtained from a single thermocouple is a scalar value; however, an RGB camera has $(W \times H \times 3)$-elements, where W and H represent the width and height of the camera (camera resolution). Each of these elements represents a sensory measurement and contains local information that cannot be extracted through basic neural networks. Inspired by parameter learning (shown in Eqn. (2.3)) and classical image processing techniques that use basic matrix operations with certain types of filters, convolutional neural networks (CNNs) learn the filter parameters. Table 2.1 shows an edge detection filter, and Table 2.2 presents a task-agnostic filter representing a kernel of a CNN, with learnable parameters, θ. During learning, a CNN takes an image as an input and outputs abstract features for further calculations. Such networks are invariant to small perturbations and transformations and are preferred for dimensionality reduction, local feature extraction and noise filtering [20].

Table 2.1: A common 3×3 filter for edge detection.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Table 2.2: A 3×3 filter of a CNN.

$$\begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ \theta_7 & \theta_8 & \theta_9 \end{bmatrix}$$


(a) A color image of an industrial process.

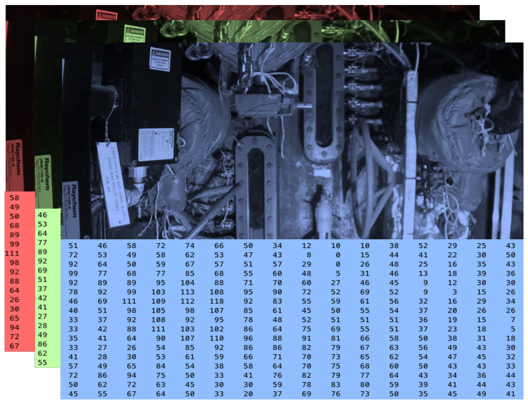

(b) Matrix representation of the three channels of Fig. 2.3a.

Figure 2.3: Representations of color (RGB) images.

2.1.2 Long Short-Term Memory Networks

Unlike the static data shown in Fig. 2.1, industrial processes often involve dynamic data with temporal connections. For example, in a video, each image frame is connected to the previous frames. Furthermore, physical processes frequently entail retention time, which results in transitional behaviour such as gradual temperature increase, where the current temperature value is dependent on the prior temperature values and the heating source. Temporal data in continuous time have the following form.

$$\frac{dy(\tau)}{d\tau} = f(y(\tau), u(\tau), w(\tau)) \tag{2.4}$$

where f, y, u, w, and τ are a function, noisy observation, exogenous input, noise, and time, respectively. Since the digital measurements are obtained at discrete timesteps, Eqn. (2.4) can be discretized as follows.

$$y_{t+1} = f_D(y_t, u_t, w_t) \tag{2.5}$$

where t and f_D represent the discrete sampling instant and a discretized function, respectively. This relationship can be captured via a parameterized recurrent network, which models the temporal relationship as shown in Eqn. (2.6).

$$\begin{aligned}
i_t &= \sigma(x_t W^{xi} + h_{t-1} W^{hi}) \\
f_t &= \sigma(x_t W^{xf} + h_{t-1} W^{hf}) \\
o_t &= \sigma(x_t W^{xo} + h_{t-1} W^{ho}) \\
g_t &= tanh(x_t W^{xg} + h_{t-1} W^{hg}) \\
C_t &= \sigma(f_t \circ C_{t-1} + i_t \circ g_t) \\
h_t &= o_t \circ tanh(C_t) \qquad (2.6)
\end{aligned}$$

where the superscripts denote the layer identity, $\circ$ denotes the Hadamard product, U and H are the weights, σ and $tanh$ are the activation functions (which add non-linearity), i, f, o, g, C, and h stand for input gate, forgetting gate, output, candidate state/modulation gate, cell state and hidden state. x corresponds to the input-output data. Bias terms, which can increase the degree of freedom of the network, are ignored for simplicity.

A common technique to train neural networks (i.e., updating its parameters) is using gradient-based approaches as long as the network is differentiable. However, the computational complexity of the gradient operation increases as the network grows. Due to recent advancements in computer science, automatic differentiation became one of the most popular techniques [21]. This technique applies the chain rule on the network with respect to its parameters, starting at the output of the network. Consider an input sequence $\{x_1, ..., x_T\}$, and the following output and error equations for the LSTM described in Eqn. (2.6).

$$\hat{y}_t = softmax(W^{hz} h_t) \qquad (2.7)$$

$$\varepsilon_t = \frac{1}{2}(y_t - \hat{y}_t)^2 \qquad (2.8)$$

where $softmax$ is an activation function. Automatic gradient methods apply the chain rule to pass the spatio-temporal information through the network. For simplicity, derivatives with respect to the final step will be shown. Taking the derivative of the error (loss) function with respect to $\hat{y}_t$ and W^{hz} yields [22]:

$$\begin{aligned}
d\hat{y}_t &= y_t - \hat{y}_t && (2.9)\\
dW^{hz} &= \sum_t^T h_t d\hat{y}_t && (2.10)\\
dh_T &= W^{hz} d\hat{y}_T && (2.11)\\
do_t &= tanh(C_t) dh_t && (2.12)\\
dC_t &= (1 - tanh(C_t)^2) o_t dh_t && (2.13)\\
df_t &= C_{t-1} dC_t && (2.14)\\
dC_{t-1} &= dC_{t-1} + f_t \circ dC_t && (2.15)\\
di_t &= g_t dC_t && (2.16)\\
dg_t &= i_t dC_t && (2.17)\\
& && (2.18)
\end{aligned}$$

Considering the sigmoid and tanh functions, backpropagation from $t = 1$ to $t = T$ can be derived as follows.

$$\begin{aligned}
dW^{xo} &= \sum_t^T o_t(1-o_t)x_t do_t && (2.19)\\
dW^{xi} &= \sum_t^T i_t(1-i_t)x_t di_t && (2.20)\\
dW^{xf} &= \sum_t^T f_t(1-f_t)x_t df_t && (2.21)\\
dW^{xc} &= \sum_t^T (1-g_t^2)x_t dg_t && (2.22)\\
dW^{ho} &= \sum_t^T o_t(1-o_t)h_{t-1} do_t && (2.23)\\
dW^{hi} &= \sum_t^T i_t(1-i_t)h_{t-1} di_t && (2.24)\\
dW^{hf} &= \sum_t^T f_t(1-f_t)h_{t-1} df_t && (2.25)\\
dW^{hc} &= \sum_t^T (1-g_t^2)h_{t-1} dg_t && (2.26)\\
dh_{t-1} &= o_t(1-o_t)W^{ho}do_t + i_t(1-i_t)W^{hi}di_t \\
&+ f_t(1-f_t)W^{hf}df_t + (1-g_t^2)W^{hc}dg_t + W^{hz}dz_{t-1} && (2.27)
\end{aligned}$$

Following these derivations, the objective function can be written in compact form as follows.

$$\mathcal{L}(x, A) = \min_A \sum_t^T \frac{1}{2}(y_t - \hat{y}_t)^2 \tag{2.28}$$

where $A = \{W^{xo}, W^{xi}, , W^{xf}, W^{xc}, W^{ho}, W^{hi}, W^{hf}, W^{hc}, W^{hz}\}$. Consider $\mathcal{L}(t) = \varepsilon_t$ and take the derivative of the loss function at $t = T$ with respect to C_T and at $t = T-1$ with respect to C_{T-1}.

$$\begin{aligned}
\frac{\partial \mathcal{L}(T)}{dC_T} &= \frac{\partial \mathcal{L}(T)}{\partial h_T}\frac{\partial h_T}{\partial C_T} && (2.29)\\
\frac{\partial \mathcal{L}(T-1)}{dC_{T-1}} &= \frac{\partial \mathcal{L}(T-1)}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial C_{T-1}} + \frac{\partial \mathcal{L}(T)}{dC_{T-1}} && (2.30)\\
&= \frac{\partial \mathcal{L}(T-1)}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial C_{T-1}} + \frac{\partial \mathcal{L}(T)}{dh_T}\frac{\partial h_T}{\partial C_T}\frac{\partial C_T}{\partial C_{T-1}} && (2.31)
\end{aligned}$$

where the chain rule, shown in Eqn. (2.31), can be rewritten as follows.

$$dC_{T-1} = dC_{T-1} f_T \circ dC_T \tag{2.32}$$

More information regarding the backpropagation in an LSTM network can be found in [22]. Furthermore, there are numerous variants of recurrent networks [23,24] for different tasks. Some examples include gated recurrent networks [25], convolutional LSTMs [26], bidirectional LSTMs [27], etc. Since the LSTMs retain temporal information while learning significant features in the input data, they can provide robust estimations given the historical data.

2.2 State Estimation with Closed Skew-Normal Distribution

The former sections described the fundamentals of data-driven techniques that will be covered in this thesis. However, neural network-based estimations often provide point estimations in the state/action space, which ignore the uncertainty associated with the data. Moreover, most neural networks inherently ignore the temporal connection between the data points. State estimation techniques provide a probabilistic scheme based on the Bayes rule to address uncertainty while considering the time-related details. Unlike data-driven techniques, state estimation utilizes prior information related to the process. Although there are various filtering and smoothing techniques, this thesis focuses on online variants for practicality. A challenge with the analytical techniques covered in this section is the dimensionality issue. That is, these techniques do not scale up feasibly (due to the amount of time required for calculations and the virtual space needed to store the variables of interest) when the dimensions of the state/action spaces increase. For example, a Kalman filter calculates an $n \times n$ covariance matrix for an n-dimensional state space, which should be considered if the states constitute, for example, pixels of a large image. In such cases, dimensionality reduction techniques (e.g., principal component analysis [28]) can be used to pre-process the data, or numerical estimation methods can be preferred for approximate solutions.

2.2.1 Kalman Filtering (KF)

Industrial processes involve physical or chemical processes that evolve over time. A fundamental property in such processes is their dynamic nature, which can be measured through various sensors. This property can be mathematically expressed through a discrete-time state-space model as shown in Eqns. (2.33) and (2.34).

$$x_{t+1} = A_t x_t + B_t u_t + E_t \epsilon_t \tag{2.33}$$

$$y_t = C_t x_t + D_t u_t + F_t \psi_t \tag{2.34}$$

where $x_t \in \mathbb{R}^n$ is the true state, and $\epsilon_t \sim \mathcal{N}(0, \Sigma_{\epsilon,t})$ and $\psi_t \sim \mathcal{N}(0, \Sigma_{\psi,t})$ are the state and measurement noises respectively. A_t, B_t, C_t, D_t, E_t, and F_t are system matrices that regulate the contribution of each element to the state/measurement and can be determined by using various system identification techniques [29]. After estimating the parameters, the true state can be estimated by combining the state-space model and measurements through Bayes' rule.

Bayesian filtering aims to find the distribution of x_t at time t given the historical measurements up to t, which is shown as follows:

$$p(x_t|y_{1:t}) \tag{2.35}$$

Following its initialization with a prior distribution $p(x_0)$, a Bayes filter can predict the state, x_t, and update its prediction through the noisy measurement, y_t. The predictive distribution can be calculated by the Chapman-Kolmogorov equation as follows.

$$p(x_t|y_{1:t-1} = \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1} \tag{2.36}$$

After receiving a noisy measurement, the posterior distribution of the state can be obtained by using Bayes' rule as shown below.

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{\int p(y_t|x_t)p(x_t|y_{1:t-1})dx_t} \tag{2.37}$$

A Kalman filter simplifies the Bayesian filtering by assuming the Markovian property, linear state and measurement models with Gaussian state and measurement noises.

The Markov property states that given x_{t-1}, the current state, x_t, and its future values, $x_{t+1:t+\infty}$, are independent of the past prior to the time step $t-1$, as shown in Eqn. (2.38) [30].

$$p(x_t|x_{1:t-1}, y_{1:t-1}) = p(x_t|x_{t-1}) \tag{2.38}$$

After considering $D_t = 0$ and $E_t = F_t = 1$, for simplicity, Eqns. (2.39) and (2.40) can be written as:

$$p(x_{t+1}|x_t) {=} \mathcal{N}(x_{t+1}|A_t x_t + B_t u_t, \Sigma_{\epsilon,t}) \tag{2.39}$$

$$p(y_t|x_t) {=} \mathcal{N}(y_t|C_t x_t, \Sigma_{\psi,t}) \tag{2.40}$$

Following these probabilistic relationships, the Kalman filter can be derived as follows [31]:

$$\hat{x}_t^- = A_{t-1}\hat{x}_{t-1}^+ + B_{t-1}u_{t-1} \tag{2.41}$$

$$P_t^- = A_{t-1}P_{t-1}^+ A_{t-1}^T + \Sigma_{\epsilon,t-1} \tag{2.42}$$

$$K_t = P_t^- C_t^T (C_t P_t^- C_t^T + \Sigma_{\psi,t})^{-1} \tag{2.43}$$

$$\hat{x}_t^+ = \hat{x}_t^- + K_t(y_t - C_t\hat{x}_t^-) \tag{2.44}$$

$$P_t^+ = (I - K_t C_t)P_t^- \tag{2.45}$$

where $(\cdot)^-$ and $(\cdot)^+$ represent a priori and a posteriori estimates, and K_t is the Kalman gain respectively. Note that if $C_t = 1$ and $D_t = 0$, then estimating $\hat{y}_t$ is equivalent to estimating $\hat{x}_t$.

Under the normality assumption, a Kalman filter can be used to estimate the state of interest. However, measurements can be skewed due to instrumentation and the properties of the noisy signal. In such cases, the normal distribution cannot model the underlying dynamics, and hence, a different filter is needed.

2.2.2 CSN Distribution

Although a Kalman filter is commonly used with Gaussian noise assumption, sensory measurement can be significantly affected by instrumentation (such as interference in sensors). For example, physical obstacles can cause asymmetric/skewed measurements [32]. In such cases, the performance of a Kalman filter can deteriorate due to the filter's normality assumption. A skew filter can address this challenge by utilizing asymmetric (e.g., skew-t, Weibull or skew-normal) measurement distributions. An alternative distribution is the closed skew-normal (CSN) distribution that generalizes the Gaussian distribution by adding skewness into the model. Some representative examples are shown in Fig. 2.4.

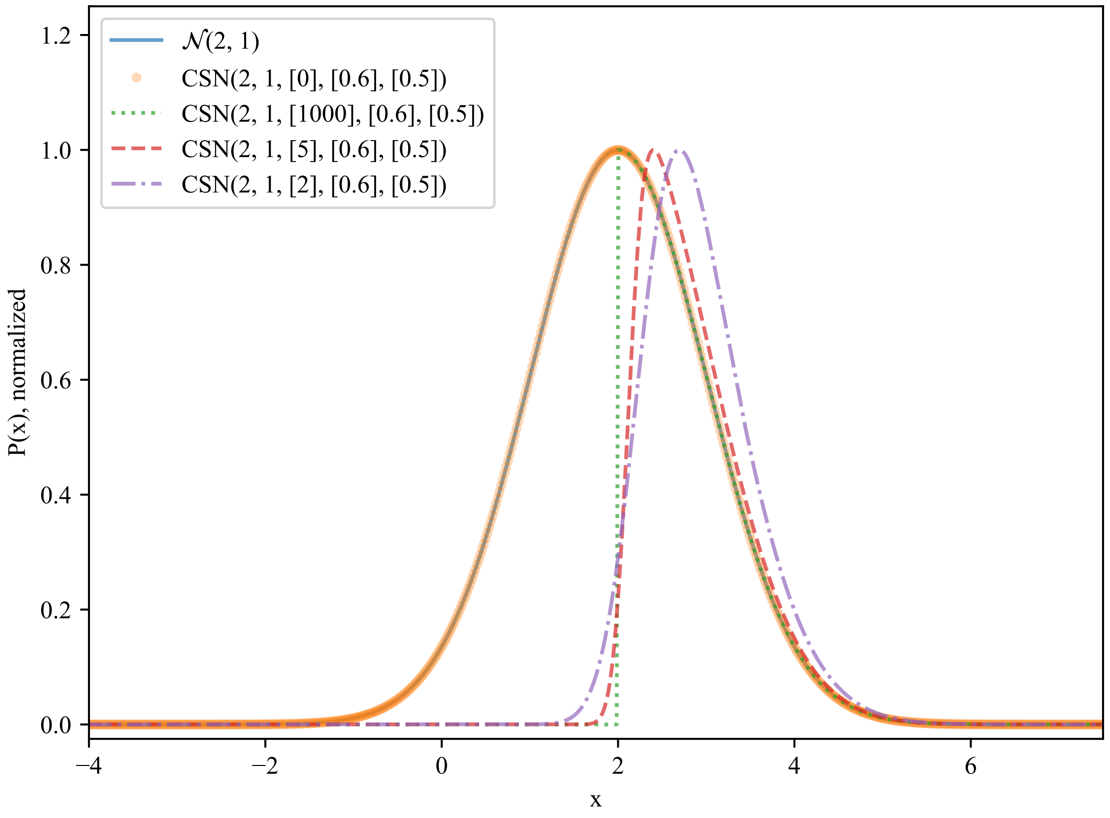


Figure 2.4: A Gaussian ($\mathcal{N}(2, 1)$) and various CSN distributions. A CSN with zero skewness (the third parameter) is equivalent to a Gaussian distribution, as shown with the orange dots.

Moreover, a CSN distribution is closed under linear transformations, and a convolution or Bayesian inversion operation as long as the distributions involved in these operations are also CSN [33]. The closedness property makes a CSN a viable candidate distribution for recursive estimations via the Bayesian framework. However, the dimensionality of the skew parameters of a CSN increases following the measurement update step. Due to this issue, the mean of the resulting CSN cannot be calculated analytically, and the CSN parameters cannot be feasibly stored during online operations.

2.3 Model Predictive Control

The purpose of state estimation is to calculate the statistics of the process variables given noisy measurements. The estimated state then can be utilized in control to improve the performance of the process while maintaining safety. Model predictive control (MPC) is a practical method to achieve optimal performance in the presence of accurate system models.

$$\begin{aligned}
\min_{\mathbf{u}_t} J(\mathbf{u}_t) = \min_{\mathbf{u}_t} \sum_{k=0}^{N_P-1} ||\hat{x}_{t+k} - x_{t+k,sp}||^2_{Q_k^{[1]}} + ||u_{t+k} - u_{t+k,ref}||^2_{\lambda_k} + ||\Delta u_k||^2_{Q_k^{[2]}} \\
+ ||\hat{x}_{t+N} - x_{t+N,sp}||^2_{Q_N^{[3]}} \qquad (2.46) \\
\textbf{s.t. } \text{Eqns. } (2.33), (2.34), \\
u_{min} \leq u_t \leq u_{max}, \forall t \\
\hat{x}_{min} \leq \hat{x}_t \leq \hat{x}_{max}, \forall t \\
\hat{x}_0 = \mathbb{E}[\hat{x}], \hat{x} \in \mathcal{X}, u \in \mathcal{U} \qquad (2.47)
\end{aligned}$$

where J is a quadratic cost function that is minimized at every discrete time step with $\mathbf{u}_t = \{u(0), u(1), ..., u(N_C - 1)\}$. $x_{t,sp}$ and $u_{t,ref}$ are state and action setpoints, $\Delta u = (u_{t+1} - u_t)$ is a control velocity term, N_P and N_C are the prediction and control horizons, $(\cdot)_{min}$ and $(\cdot)_{max}$ are the lower and upper limits respectively. k is a discrete time step, $Q_k^{[1]}$, $Q_k^{[2]}$, $Q_N^{[3]}$ and λ_k are the square weight matrices, and $\mathcal{X}$ and $\mathcal{U}$ are the state and action spaces, respectively. After the input sequence, $\mathbf{u}_t$ (i.e., the open-loop optimization solution) is calculated according to Eqn. (2.48), the first control action, $u_t = u(0) \in \mathbf{u}_t$, is sent to the plant. This strategy is also known as the receding horizon control.

$$\mathbf{u}_t = \arg\min_{\mathbf{u}_t} J(\mathbf{u}_t) \qquad (2.48)$$

As shown in Eqn. (2.47), an MPC can handle various constraints in linear/nonlinear form given linear/nonlinear system models. However, in the presence of constraints, the resulting optimization problem cannot be solved analytically, necessitating the use of numerical methods.

2.4 Reinforcement Learning

Unlike MPCs, which use the process model directly, machine learning methods require training a neural network based on process data. That is, the data used in MPC is the current feedback from the system, which is an estimation of the state of interest. Although a conventional time-invariant MPC provides a locally optimal controller output at each step, it does not handle variations in the operational conditions since it does not have an update mechanism.

Inspired by model-based dynamic programming [34] and animal learning, reinforcement learning (RL) provides an alternative framework to optimal control by training an agent in the process environment [35]. The agent optimizes a control policy while interacting with the environment. The policy defines the behaviour of the agent given a state during the interaction. After an action is selected according to the policy, it is implemented in the environment, and the agent receives a reward signal, indicating the goodness of the decision. Although the agent can learn a policy offline (through historical data) [36], this thesis will cover online/recursive RL algorithms that can learn autonomously without explicit system knowledge or any process model.

2.4.1 Markov Decision Process

Sequential problems require optimal decisions with temporal order. Considering the complex nature of the environment, the outcomes of the actions of an agent are often uncertain. The Markov decision process (MDP) is a probabilistic framework for sequential decision making where the agent receives a state, $x \in \mathcal{X}$, chooses an action $u \in \mathcal{U}$, receives a reward r and the next state $x' \in \mathcal{X}$. This framework assumes the Markov property, where the information in the next time step depends only on the current time step, as shown in Fig. 2.5. The system dynamics are governed by a transition probability function, $p(x', r|x, u)$, and the agent's goal is to maximize a return function, G, which is shown in Eqn. (2.49).

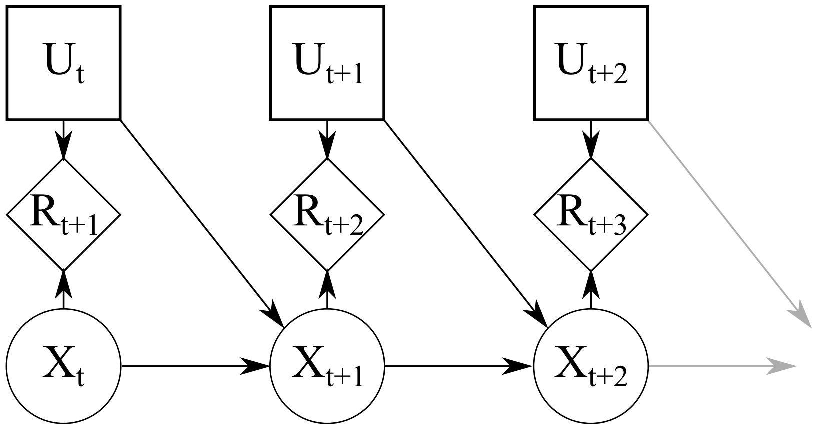


Figure 2.5: A graphical representation of the Markov decision process. The next state and the reward depend only on the current state and action. The capital letters indicate that the state, action and reward are random variables.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.49}$$

where the capital letters denote that the reward is a stochastic variable, and $\gamma \in [0, 1]$ is a weight that controls how much future gains will contribute to the return function. During its interactions with the environment, the agent samples its actions from a stochastic policy, $\pi(u|x)$, the performance of which is tracked using a value function. There are two kinds of value functions: v(x) and q(x, u), and the type depends on the policy evaluation approach. Learning can be performed by solving the Bellman equations iteratively, as illustrated in Eqns. (2.50)-(2.51) [37].

$$\begin{aligned} v_\pi(x) &= \mathbb{E}_\pi \left[G_t | X_t = x \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | X_t = x \right], \forall x \in \mathcal{X} \\ &= \sum_u \pi(u|x) \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | X_{t+1} = x' \right] \right] \\ &= \sum_u \pi(u|x) \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma v_\pi(x') \right] \end{aligned} \tag{2.50}$$

$$\begin{aligned} q_\pi(x, u) &= \mathbb{E}_\pi \left[G_t | X_t = x, U_t = u \right], \forall x, u \in \mathcal{X} \times \mathcal{U} \\ &= \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma \sum_{u'} \pi \left(u' | x' \right) q_\pi(x', u') \right] \end{aligned} \tag{2.51}$$

where $\mathbb{E}[\cdot]$ represents the expected value of a random variable. Eqns. (2.53) can be used to find the optimal value functions after the recursions.

$$v^*(x)=\max_{\pi} v_\pi(x), \forall x \in \mathcal{X} \tag{2.52}$$

$$\begin{aligned} q^*(x,u)&=\max_{\pi} q_\pi(x,u), \forall x,u \in \mathcal{X} \times \mathcal{U} \\ &=\mathbb{E}\left[R_{t+1} + \gamma v^*(X_{t+1}) | X_t = x, U_t = u\right] \end{aligned} \tag{2.53}$$

Finally, Eqn. (2.54) may be used to calculate the optimum (also known as greedy) policy.

$$\pi^*(x) = \arg\max_{u} q^*_\pi(x,u) \tag{2.54}$$

However, because the system model, $p(\cdot)$, is often unknown, Eqns. (2.50) and (2.51) cannot be solved analytically. The lack of $p(\cdot)$ in RL and the existence of $\pi(\cdot)$ in MPC are crucial differences. That is, the agent learns the value function from the data and stores it in memory, whereas the MPC recalculates it at each step using the system model. As a result, an RL agent learns to control and can save computational time upon learning.

2.4.2 Value-based (Critic-only) RL

Initial implementations used the value-based (critic-only) methodology [38, 39] to solve control problems. In these methodologies, actions are derived directly from a value function, which predicts the long-term outcomes of a specific policy. The state value function (shown in Eqn. (2.50)) is the expected return obtained from state x while following policy π. The action-value function (shown in Eqn. (2.51)) is the expected return after taking action u in state x and following the policy π thereafter. The optimal value functions, $v^*(\cdot)$ and $q^*(\cdot)$ (shown in Eqns. (2.52) and (2.53)) are the unique value functions that maximize the value of every state.

Value-based methodologies, during policy evaluation, estimate $V(x) \approx v_\pi(x)$ or $Q(x,u) \approx q_\pi(x,u)$ for the current policy and improve the policy iteratively. A common example is greedily selecting actions with respect to the updated value function.

Although dynamic programming (DP) [34] and Monte Carlo (MC) techniques can be used to solve an RL problem, these methods are computationally infeasible, need to wait until an episode ends, have high variance, or require a perfect system model. Temporal difference (TD) methodology provides a practical alternative to these algorithms by combining the bootstrapping (estimating the values by using the previously estimated values) ability of DP and the model-free nature of MC. As a result, TD algorithms can update their policies before an episode ends. Two examples of state and action-value function update rules are given in Eqns. (2.55) and (2.56).

$$V(X_t) \leftarrow V(X_t) + \alpha(R_{t+1} + \gamma V(x_{t+1}) - V(x_t)) \tag{2.55}$$

$$Q(X_t, U_t) \leftarrow Q(X_t, U_t) + \alpha(R_{t+1} + \gamma Q(x_{t+1}, U_{t+1}) - Q(X_t, U_t)) \tag{2.56}$$

where "$\leftarrow$" represents the update operation. Note the similarity between the value function update rules and the update rule in Kalman filtering (shown in Eqn. (2.44)), where $(R_{t+1} + \gamma V(X_{t+1}))$, $V(X_t)$, and α in TD learning respectively correspond to y_t (measurement), $\hat{x}_t^-$ (prediction), and K_t (Kalman gain) in state estimation.

Built upon the TD learning methodology, two RL algorithms, namely SARSA (state-action-reward-state-action) and Q-learning [35], have shown promising results in process control applications [40–42]. These algorithms differ in terms of their policy evaluation/improvement strategies. For example, SARSA is an *on-policy* algorithm since it improves its policy that is used to make decisions. In contrast, the Q-learning algorithm is *off-policy* since it improves a policy different from that used to generate its data. The policy improvement steps are shown in Step 3.c.iii of Algorithm (2.1) and Step 3.b.iii of Algorithm (2.2) [35]. These steps show that the SARSA algorithm uses U_{t+1} in the process and updates the Q-function by using it. However, the Q-learning algorithm performs an additional greedy action selection to find U and updates the action-value (Q) function by using U. Since U is not used to control the system (but is used to update the Q-function), it can act as an additional exploration factor in Q-learning. On the other hand, the optimal Q-function can result in aggressive control actions, which can compromise safety, as Sutton and Barto showed [35] through a cliff walking problem. Therefore, it is the practitioners' responsibility to select and test the algorithm considering mathematical and safety requirements.

The terminal state in RL can be defined based on an event or time. Some ex-

Algorithm 2.1 SARSA algorithm.

```
1. Hyperparameters: $\alpha \in (0,1]$, discount factor, $\gamma \in [0,1]$, an exploration prob-
   ability, $\epsilon \in [0,1]$.
2. Initialize $Q(x,u) \forall x \in \mathcal{X}, u \in \mathcal{U}$ arbitrarily.
3. Repeat for each episode
   (a) Initialize $X_t$
   (b) Choose $U_t$ for $X_t$ using policy derived from $Q(\cdot)$.
   (c) Repeat for each step in the episode until $X_t$ is terminal:
         i. Take action $U_t$, observe $X_{t+1}, R_{t+1}$
        ii. Choose $U_{t+1}$ given $X_{t+1}$ using policy derived from $Q(\cdot)$.
       iii. $Q(X_t,U_t) \leftarrow Q(X_t,U_t) + \alpha(R_{t+1} + \gamma Q(X_{t+1},U_{t+1}) - Q(X_t,U_t))$
        iv. $X_t \leftarrow X_{t+1}, U_t \leftarrow U_{t+1}$
```

Algorithm 2.2 Q-learning algorithm.

```
1. Hyperparameters: $\alpha \in (0,1]$, discount factor, $\gamma \in [0,1]$, an exploration prob-
   ability, $\epsilon \in [0,1]$.
2. Initialize $Q(x,u) \forall x \in \mathcal{X}, u \in \mathcal{U}$ arbitrarily.
3. Repeat for each episode
   (a) Initialize $X_t$
   (b) Repeat for each step in the episode until $X_t$ is terminal:
         i. Choose $U_t$ for $X_t$ using policy derived from $Q(\cdot)$.
        ii. Take action $U_t$, observe $X_{t+1}, R_{t+1}$
       iii. $Q(X_t,U_t) \leftarrow Q(X_t,U_t) + \alpha(R_{t+1} + \gamma \max_U Q(X_{t+1},U) - Q(X_t,U_t))$
        iv. $X_t \leftarrow X_{t+1}$
```

amples of the terminal state include but are not limited to an ‘unsafe’ state (when the temperature of a reactor reaches a certain value), a specific time step (when the plant has been operated for x hours), and average return value (when the agent has achieved a specific performance metric). If the hyperparameters are selected appropriately, SARSA and Q-learning algorithms will asymptotically converge their optimal values [35]. Successful SARSA and Q-learning applications have been reported in [43–46].

A challenge with SARSA and Q-learning algorithms is that the action value function is stored as a look-up table, where the Q-value is represented explicitly for each state-action pair. On one hand, large discretization steps can reduce the accuracy of the Q-table. On the other hand, selecting small discretization steps makes it infeasible to store and update the Q-table for large or continuous state/action spaces. Therefore, for large state/action spaces, a practical solution is to use approximate value functions, such as $V(x|\omega)$ or $Q(x,u|\omega)$, instead of storing $v\,(x)$ or $q\,(x,u)$ for each state-action pair. The parameters of the value functions are specified by ω in this case. A variety of applications have utilized deep neural networks to train RL agents in large/continuous state spaces [47–53]. Nevertheless, the value-based RL algorithms often generate discrete and deterministic actions (which can be insufficient for continuous state-action space control problems) and have been reported to be divergent for large-scale problems [54, 55].

2.4.3 Policy-based (Actor-only) RL

Process industries often involve large/continuous action spaces with stochastic state transitions. Policy-based (actor-only) methods [56–58] learn stochastic and continuous actions by parameterizing a policy, $\pi_\theta(u|x,\theta)$, and directly optimizing it by using a performance metric, as demonstrated in Eqn. (2.57).

$$J_2(\theta) = \max_{\theta} \mathbb{E}_{\pi_\theta}[G_t|\theta] \tag{2.57}$$

where $J_2(\theta)$ is the agent’s objective function. As stated in Eqn. (2.58), the policy update rule can be obtained by using the policy gradient theorem [35].

$$\Delta\theta = \theta_{t+1} - \theta_t = \alpha G_t \nabla_\theta \ln \pi_\theta(U_t|X_t,\theta_t) \tag{2.58}$$

where α denotes the learning rate. Although policy gradient methods can converge to at least locally optimal policies, learn continuous actions and 'fuzzy' strategies that are a mixture of different actions, and often converge better than the value-based methods, they generally suffer from higher variance than the value-based methods. To reduce the variability during learning, REINFORCE algorithm modifies the policy gradient algorithm as shown in Eqn. (2.59) [59].

$$\Delta\theta = \alpha(G_t - b(X_t))\nabla_\theta \ln \pi_\theta(U_t|X_t, \theta_t) \tag{2.59}$$

where $b(X_t)$ is a baseline independent on the action, U_t. Sutton et al. have modified this methodology further by replacing the actual return with the action value function, $q_\pi(X_t, U_t)$ as shown in Eqn. (2.60) [60].

$$\Delta\theta = \alpha(Q_\pi(X_t, U_t) - b(X_t))\nabla_\theta \ln \pi_\theta(U_t|X_t, \theta_t) \tag{2.60}$$

Although Eqn. (2.58)-(2.59) update θ in the direction of high return values, α and G remain crucial for θ to converge. Furthermore, $|\Delta\theta| = |\theta_{t+1} - \theta_t| \geq 0$ if $|G| \geq 0$ or $|\nabla \ln \pi| \geq 0$. Moreover, Q_π in Eqn. (2.60) is the expected return, which is initially unknown. Despite these challenges, policy-gradient methods have been applied to continuous action spaces in various domains [61–67].

2.4.4 Actor-Critic RL

Similar to a student-teacher pair or generative adversarial networks (GANs) [68] that utilize generative and discriminative networks, actor-critic algorithms generate control actions and examine the outcomes by using a scalar reward signal without any labels [55, 69–71]. As shown in Fig. 2.6, these algorithms combine policy and value-based methods via an actor and a critic respectively. The actor and the critic can be represented as two neural networks, $\pi(u|x, \theta)$ and $V(x|\omega)$ (or $Q(x, u|\omega)$), respectively. This value-assisted policy learning methodology reduces θ variability while promoting convergence to optimal policies [35, 72]. Although various actor-critic algorithms have been proposed to solve the optimal control/RL problems, an early example combines the policy gradient with state-value estimation [35], as shown in Eqn.(2.61).

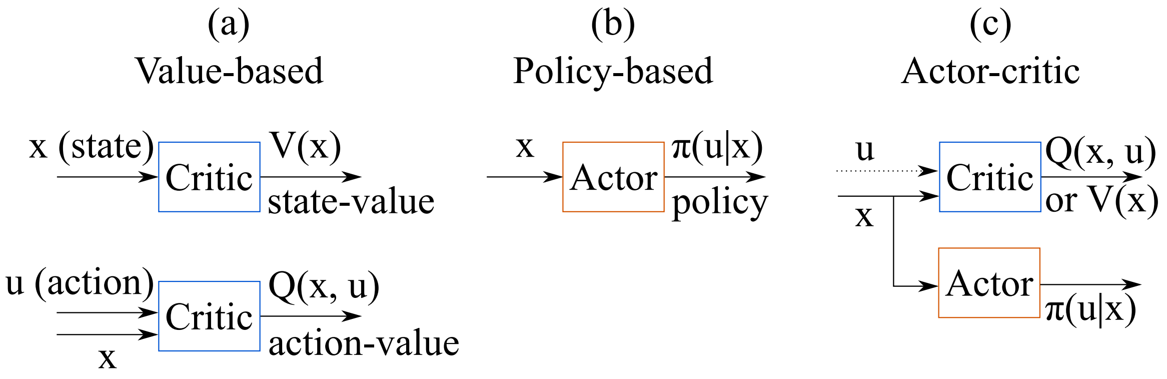


Figure 2.6: Comparison of value, policy and actor-critic based RL. The value-based methods derive the policy based on the value functions (which estimate the future return values), the policy-based methods directly optimize the policy, and the actor-critic methods simultaneously learn the policy and the value functions.

$$\begin{aligned}\theta_{t+1} - \theta_t &= \alpha(G_t - V(X_t, \omega))\nabla \ln \pi(U_t|X_t, \theta_t) \\ &= \alpha(R_{t+1} + \gamma V(X_{t+1}, \omega) - V(X_t, \omega))\nabla \ln \pi(U_t|X_t, \theta_t) \end{aligned} \tag{2.61}$$

Initially, $V(X_t, \omega) \neq (R_{t+1} + \gamma V(X_{t+1}, \omega))$ since ω is often a set of randomly initialized parameters. As a result, high values of $(R_{t+1} + \gamma V(X_{t+1}, \omega))$ will result in large $\Delta\theta$. However, as $V(X_t, \omega)$ approaches $(R_{t+1} + \gamma V(X_{t+1}, \omega))$, the variability in θ decreases over time [73].

This subsection focuses on the most commonly used model-free algorithms that are represented in Table 2.3. Some of these methods use entropy regularization, whereas others take advantage of heuristic methods. A common example of these methods is the ε-greedy approach, where the agent takes a random action with a probability $\varepsilon \in [0, 1)$. $\varepsilon = 1$ corresponds to random search since it learns a policy but does not utilize the learned policy in the decision-making process. Other exploration techniques include but are not limited to introducing additive noise to the action space, introducing noise to the parameter space, utilizing the upper confidence bound, etc. The readers can see [40] for more detail. The actor-critic algorithms are summarized as follows:

Table 2.3: A comparison of the actor-critic algorithms based on the type of action spaces & the exploration method. The state space can be either discrete or continuous for all of the algorithms.

Algorithm	**Action Space**	**Exploration**
DDPG	Continuous	Noisy actions
A2C or A3C	Discrete/Continuous	Entropy regularization
ACER	Discrete/Continuous	Entropy regularization
PPO	Discrete/Continuous	N/A
ACKTR	Discrete/Continuous	N/A
SAC	Continuous	Entropy regularization
TD3	Continuous	Noisy actions

2.4.4.1 Deep deterministic policy gradient (DDPG)

The DDPG algorithm [74] has been proposed to generalize discrete, low-dimensional value-based approaches [39] to continuous action spaces. This algorithm uses two deep neural networks, namely the deep policy gradient (DPG) and deep Q-learning algorithms, to map the states into actions and estimate the action-value function (Q-function) respectively. The resulting architecture is shown in Fig. 2.7. Similar to the policy update methodology shown in Eqn. (2.61), this algorithm updates the policy by using the derivative of the Q-function with respect to ω. This update rule helps the agent maximize the expected return while improving the value estimation and policy. In addition to this improvement, this algorithm utilizes copies of the actor ($\pi(u|x, \theta)$ and critic ($Q(x, u|\omega)$) as target networks ($\pi(u|x, \theta')$ $Q(x, u, \omega')$). After observing a state, real-valued actions are sampled from the actor network and are mixed with a random process (e.g. Ornstein-Uhlenbeck Process, [75]) to encourage exploration as shown in Eqn. (2.62).

$$U_t = \pi(U_t|X_t, \theta) + OU_t \tag{2.62}$$

where $dOU = -(OU_\eta)OUdt + \sigma dW_t$, $OU_\eta > 0$ and $\sigma > 0$ are tuning parameters, and W_t is the Wiener process. Note that the OU process is a correlated noise, but the use of white noise has also been reported for exploration purposes [76]. The agent stores state, action, and reward samples in an experience replay buffer to break the correlation between consecutive samples to improve learning. It minimizes the mean square error of the loss function to optimize its critic, as shown (for one sample) in

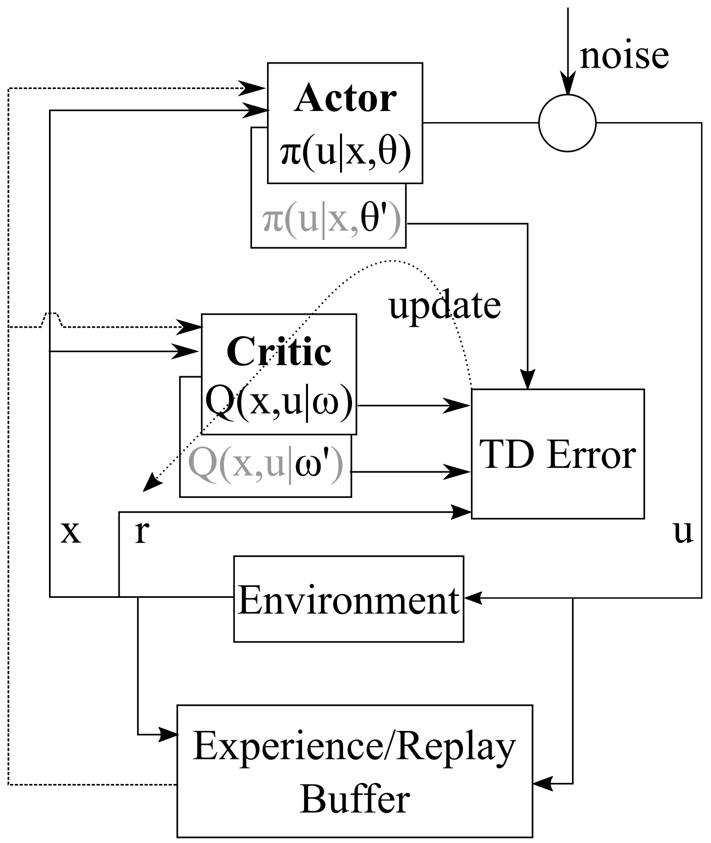


Figure 2.7: A schematic of the DDPG algorithm. The solid lines show the data flow, and the dashed lines show the update mechanism.

Eqn. (2.63), and updates the policy parameters using the policy gradient shown in Eqn. (2.64).

$$L = \left(R_t + \gamma Q\left(X_{t+1}, \pi(X_{t+1}|\theta')|\omega'\right) - Q\left(X_t, U_t, \omega\right)\right)^2 \quad (2.63)$$

$$\nabla_\theta(Q(X, \theta(X))) = \nabla_{U_t} Q(X_t, \pi(U_t|X_t, \theta)|\omega)\nabla_\theta \pi(X_t|\theta) \quad (2.64)$$

The target networks are updated using a low-pass filter, as shown in Eqn. (2.65)

$$[\omega', \theta']^T \leftarrow \tau_{ddpg}[\omega, \theta]^T + (1 - \tau_{ddpg}[\omega', \theta']^T \quad (2.65)$$

where τ_{ddpg} is the filter coefficient that adjusts the contribution of the observation (ω and θ, which are a function of the empirical/observed reward) and the previous values of the target parameters. Since the value function is learned for the target policy based on a different behaviour policy, DDPG is an **off-policy** method.

2.4.4.2 Asynchronous advantage actor-critic (A2C/A3C)

Instead of storing the experience in a replay buffer that requires memory, this scheme involves local workers that interact with their environments and update a global network asynchronously, as shown in Fig. 2.8. This update scheme inherently increases exploration since the individual experience of the local workers is independent [73]. Instead of minimizing the error based on the Q function, this scheme minimizes the

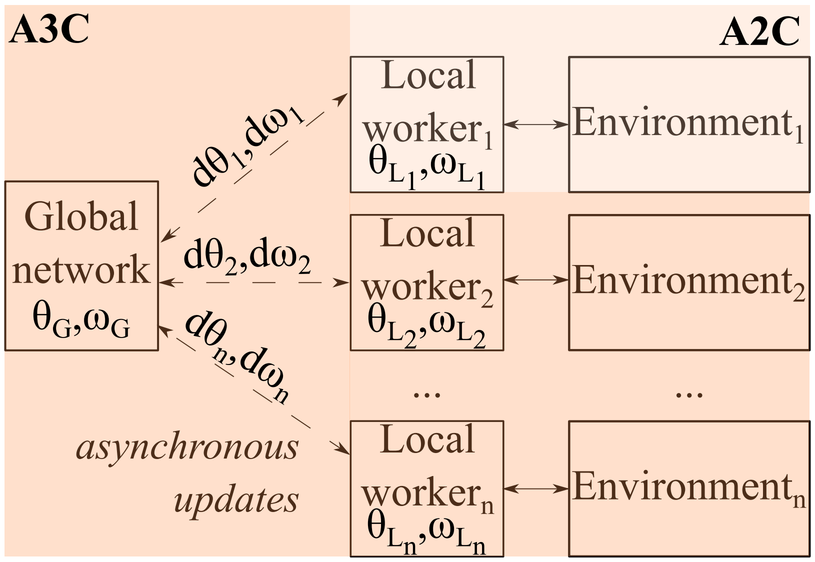


Figure 2.8: Multiple worker scheme in the A3C algorithm. Local workers interact with their environment and update a global network. Using a single A3C worker results in an A2C agent.

mean square error of the advantage function (A or δ) for critic update, as shown in Eqn. (2.66).

$$A_t = \delta = R_t + V\left(X_{t+1}|\omega\right) - V\left(X_t|\omega\right) \tag{2.66}$$

In this scheme, the global critic is updated by using Eqn. (2.67) and the entropy of the policy is used as a regularizer in the actor loss function to increase exploration, as shown in Eqn. (2.68).

$$d\omega_G \leftarrow d\omega_G + \alpha_c \nabla_{\omega_L} \delta\left(x_t|\omega_L\right)^2 \tag{2.67}$$

$$d\theta_G \leftarrow d\theta_G + \alpha_a \nabla_{\theta_L} \delta\left(x_t|\omega_L\right) \ln \pi\left(u_t|x_t, \theta_L\right) + \beta \pi\left(u_t|x_t, \theta_L\right) \ln \pi\left(u_t|x_t, \theta_L\right) \tag{2.68}$$

where initially $d\theta_G = d\omega_G = 0$. Left arrow ($\leftarrow$) represents the update operation, α_c

and α_a are the learning rates for critic and actor respectively, ∇ is the derivative with respect to its subscript, and β is a fixed entropy term that is used to encourage exploration. Subscripts L and G stand for the local and the global networks respectively. Multiple workers (A3C) can be used in an off-line manner and the scheme can be reduced to a single worker (A2C) to be implemented online. Even though the workers are independent, they predict the value function based on the behaviour policy of the global network, which makes A3C an **on-policy** method.

2.4.4.3 Actor-critic with experience replay (ACER)

ACER was proposed to address sample inefficiency of A3C and improve learning stability [77]. The algorithm utilizes 'truncated importance sampling with bias correction, trust region policy optimization, stochastic 'duelling' network architectures, and the Retrace algorithm [78]. the ACER algorithm modifies the policy update rule shown in Eqn. (2.61) for a trajectory $\{X_0, U_0, R_1, \mu(\cdot|X_0), ...X_k, U_k, R_{k+1}, \mu(\cdot|X_k)\}$ and calculates the importance weighted policy shown in Eqn. (2.69).

$$\Delta\theta = \left(\prod_{t=0}^{k} \rho_t\right) \sum_{t=0}^{k} \left(\sum_{i=0}^{k} \gamma^i r_{t+1+i}\right) \nabla_\theta \log \pi_\theta(U_t|X_t) \tag{2.69}$$

where $\rho_t = \frac{\pi(U_t|X_t)}{\mu(U_t|X_t)}$ is the important weight, and μ and π are the behaviour and the target policies respectively. To reduce the variance, the algorithm estimates the action-value of the policy by using the Retrace algorithm shown in Eqn. (2.70).

$$Q^{ret}(s_t, a_t) = R_t + \gamma \bar{\eta}_{t+1}[Q^{ret}(X_{t+1}, U_{t+1}) - Q(X_{t+1}, U_{t+1})] + \gamma V(X_{t+1}) \tag{2.70}$$

where the truncated importance weight, $\bar{\eta}_t = \min\{c, \rho_t\}$, and c is a clipping constant. The algorithm updates the actor and critic using the clipped trust region policy optimization technique and the Retrace algorithm shown in Eqns. (2.71) – (2.75).

$$
\begin{aligned}
Q^{ret} &= R_{t+1} + \gamma V(X_t|\theta') && (2.71)\\
g &= \min\{c, \rho_t\}\nabla_{\phi_{\theta'}(X_t)} \log f(U_t|\phi_{\theta'}(X_t))(Q^{ret}(X_t, U_t) - V_\omega(X_t))\\
&+ \left[1 - \frac{c}{\rho'_t}\right]_+ (Q(X_t, U'_t|\omega') - V(X_t|\omega'))\nabla_{\phi_{\theta'}(X_t)} \log f(U'_t|\phi_{\theta'}(X_t)) && (2.72)\\
k &= \nabla_{\phi_{\theta'}(X_t)} D_{KL}[f(\cdot|\phi_\theta(X_t))|f(\cdot|\phi_{\theta'}(X_t))] && (2.73)\\
\Delta\theta &= \nabla_{\theta'}\phi_{\theta'}(x)\left(g - \max\left\{0, \frac{k^T g - \delta}{||k||_2^2}\right\} k\right) && (2.74)\\
\Delta\omega' &= (Q^{ret} - Q(X_t, U_t|\omega'))\nabla_{\omega'} Q(X_t, U_t|\omega')\\
&+ \min\{1, \rho_t\}(Q^{ret}(X_t, U_t) - Q(X_t, U_t|\omega'))\nabla_{\omega'} V(X_t|\omega') && (2.75)
\end{aligned}
$$

where f represents a sampling distribution, ϕ is a neural network that generates the statistics of f, $\rho'_t = \frac{f(U'_t|\phi_{\theta'}(X_t))}{\mu(U'_t|X_t)}$, and D_{KL} is the KL divergence. Because of its Retrace algorithm, ACER is an off-policy method.

2.4.4.4 Proximal policy optimization (PPO)

The trust region policy optimization (TRPO) algorithm suggests maximizing a soft-constrained objective function shown in Eqn. (2.76)

$$
\max_\theta J^{\text{KL}} = \max_\theta \mathbb{E}\left[\frac{\pi_\theta(U_t|X_t)}{\pi_{\theta_{\text{old}}}(U_t|X_t)} A_t - \beta D_{KL}[\pi_{\theta_{\text{old}}}(U_t|X_t)|\pi_\theta(U_t|X_t)]\right] \qquad (2.76)
$$

where β is a weight for the KL divergence term, A_t is the advantage estimate that represents how good the agent's actions are, as shown in Eqn. (2.66), and the KL divergence creates a lower bound on the performance of π. However, using a fixed weight, β, in the objective function shown in Eqn. (2.76) can result in large policy updates. To avoid abrupt changes in the policy, the PPO algorithm suggests [79] clipping the surrogate objective function as shown in Eqn. (2.77).

$$
J^{\text{CLIP}}(\theta) = \mathbb{E}\big[\min\big(r(\theta) A_t, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) A_t\big)\big] \qquad (2.77)
$$

where θ_{old} represents the old policy parameters, $r(\theta) = \frac{\pi_\theta(u|x)}{\pi_{\theta_{\text{old}}}(u|x)}$, and ϵ is the clipping constant. Then, the algorithm includes a KL penalty and a value loss function in the objective function, which results in Eqn. (2.78)

$$J^{PPO} = \mathbb{E}[J^{\text{CLIP}} - c_1 J^{\text{VF}} + c_2 \mathcal{H}] \tag{2.78}$$

where c_1 and c_2 are weight coefficients, $\mathcal{H} = \mathbb{E}[-\log \pi(U_t|x_t, \theta)]$, and $J^{\text{VF}} = (V(X_t|\omega) - V(X_t|\omega'))^2$. The resulting architecture of the PPO algorithm is shown in Fig. 2.9. Due to the practical advantages of these modifications, PPO and its variants have been one of the most commonly used algorithms to solve control problems [80].

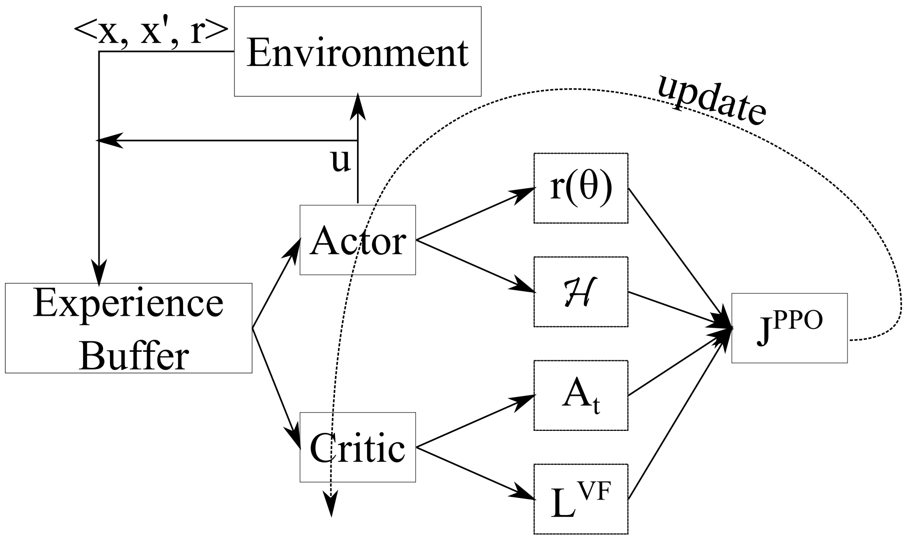


Figure 2.9: A schematic of the PPO algorithm. The solid lines show the data flow, and the dashed lines show the update mechanism.

2.4.4.5 Actor-critic using Kronecker-factored trust region (ACKTR)

Classical gradient descent/ascent algorithms, such as the general policy gradient algorithm (shown in Eqn. (2.61)), update the parameters by solving the optimization function shown in Eqn. (2.79).

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \Psi^t \nabla_\theta \log \pi(U_t|X_t|\theta)] \tag{2.79}$$

where Ψ is the magnitude of the update, which is often selected as G_t, $Q(X_t, U_t|\omega)$, or A_t. Such RL algorithms aim to maximize a non-convex function, $J(\theta)$, in the steepest ascent direction while calculating $\Delta\theta$ such that $J(\theta + \Delta\theta)$ is maximized. In general, the goal is to keep $||\Delta\theta||_z = (\Delta\theta^T z \Delta\theta)^{0.5} < 1$, and z is a positive semidefinite matrix. The result of this optimization problem is in the form of $\Delta\theta \propto z^{-1}\nabla_\theta J(\theta)$, with $z = I$, as shown in Eqn. (2.61). Instead of gradient descent algorithm [81] to optimize the actor and the critic networks, ACKTR [82] utilizes second-order optimization, which provides more information. That is, $z = F \neq I$, where F is the Fisher information matrix, which is challenging to calculate, store and invert. ACKTR overcomes the computational complexity by using Kronocker-factored approximation [83, 84] to approximate the inverse of Fisher information matrix (FIM) that, otherwise, scales exponentially with respect to the neural network parameters. Moreover, ACKTR keeps a track of the Fisher statistics, which yields better curvature estimates. The resulting algorithm updates the parameters as shown in Eqn. (2.80).

$$\Delta\theta = \alpha F^{-1} \nabla_\theta \delta \tag{2.80}$$

where $F = \mathbb{E}_{p(\tau)}[\nabla_\theta \log \pi(U_t|X_t, \theta)(\nabla_\theta \log \pi(U_t|X_t, \theta))^T]$, and τ is shown in Eqn. (2.81).

$$\tau = p(X_0) \prod_{t=0}^{T} \pi(U_t|X_t) p(X_{t+1}|X_t, U_t) \tag{2.81}$$

where p denotes the probability distribution functions. As a result of these improvements, ACKTR has shown successful results in various applications [85–87].

2.4.4.6 Soft actor-critic (SAC)

Unlike methods like A3C and PPO, which use the entropy of the policy as a loss regularizer [73, 79, 88, 89], SAC augments the reward function with the entropy term (as shown in Eqn. (2.82)) to encourage exploration while maintaining learning stability.

$$J(\theta) = \sum_{t \in T} \mathbb{E}_{(x_t, u_t) \sim \pi} \left[R\left(x_t, u_t\right) + \alpha \mathcal{H}\left(\pi_\theta\left(\cdot\right)\right) \right] \tag{2.82}$$

where θ represents the parameters of the policy, and α is a user-defined (fixed or time-varying) weight to adjust the contribution of the entropy, $\mathcal{H}$. This scheme relies on both $Q(\cdot, \phi)$ and $V(\cdot, \omega)$ functions to utilize the soft-policy iteration. The parameters of the neural networks are updated as shown in Eqns. (2.83) –(2.85).

$$\begin{aligned}\nabla_\omega J_V(\omega) &= \mathbb{E}_{X_t\sim\mathcal{D}}[0.5(V(X_t|\omega) - \mathbb{E}_{U_t\sim\pi_\theta}[Q_\phi(X_t,U_t) - \log\pi_\theta(U_t|X_t)])^2] \\ &= \nabla_\omega V(X_t|\omega)(V(X_t|\omega) - Q_\phi(X_t,U_t) - \log\pi_\theta(U_t|X_t)) \qquad (2.83)\end{aligned}$$

$$\begin{aligned}\nabla_\phi J_Q(\phi) &= \mathbb{E}_{(X_t,U_t)\sim\mathcal{D}}[0.5(Q_\phi(X_t,U_t) - \hat{Q}(X_t,U_t))^2] \\ &= \nabla_\phi Q_\phi(X_t,U_t)(Q_\phi(X_t,U_t) - R_{t+1} - \gamma V_{\omega'}(X_{t+1})) \qquad (2.84)\end{aligned}$$

$$\nabla_\theta J_\pi(\theta) = \nabla_\theta \log\pi_\theta(X_t|U_t) + (\nabla_{U_t}\log\pi_\theta(X_t|U_t) - \nabla_{U_t}Q(X_t,U_t))\nabla_\theta f_\theta(\epsilon|X_t) \qquad (2.85)$$

where U_t is evaluated at $f_\theta(\epsilon|X_t)$, ϵ is a noise vector, and $\hat{Q}(X_t,U_t) = R_{t+1} + \gamma\mathbb{E}_{X_{t+1}}[V_{\omega'}(X_{t+1})]$. Similar to DDPG and PPO, SAC stores the transitions in a replay buffer, indicated as $\mathcal{D}$, to address sample efficiency. This approach has also been reported [72, 90] to improve the robustness of the policy against model and estimation errors. Besides enhancing the exploration, this off-policy training methodology has been used in several control and optimization applications [91–93] and reported to improve stability since it utilizes target networks.

2.4.4.7 Twin delayed deep deterministic policy gradient (TD3)

TD3 is an extension to the DDPG algorithm [76]. It addresses error propagation (which is a non-trivial challenge in statistics and control [94]) due to function approximation and **bootstrapping** (i.e. instead of an exact value, using an estimated value in the update step). To reduce the overestimation bias, the scheme predicts two separate action-value functions and prefers the pessimistic value to update the network parameters, avoiding sub-optimal policies. TD3 utilizes target networks, delays the update to the policy function, and uses an average target value estimate by sampling N-transitions from a replay buffer to reduce variance during learning. The scheme introduces exploration by adding Gaussian noise to the sampled actions and performs policy updates using the deterministic policy gradient [57]. As a result of these modifications, TD3 has been considered one of the state-of-the-art RL algorithms in control and optimization [95–100].

2.4.4.8 More Application Examples from the Process Control Literature

Though the above-mentioned algorithms provide general solutions to control problems, they may remain inadequate for more complex or specific tasks. Several other algorithms have been proposed to address these shortcomings. For example, [101] has extended the discrete actor-critic method, which had been proposed by [69], to continuous time and space problems via Hamiltonian-Jacobi-Bellman (HJB) equation [34, 102]. This proposed algorithm has been tested in an action-constrained pendulum and a cart-pole swing-up problem. [103] has employed an actor-critic algorithm on a constrained MDP with detailed convergence analysis. [104] has showcased four incremental actor-critic algorithms based on regular and natural gradient estimates. [105] has introduced natural actor-critic (NAC) and demonstrated its performance on the cart pole problem and a baseball swing task. [106] has presented a continuous time actor-critic via converse HJB and tested the convergence in two nonlinear simulation environments. [107] has proposed an online actor-critic algorithm for the infinite horizon and continuous time problems with a rigorous convergence analysis and linear and nonlinear simulation examples. [108] has proposed an incremental, online and off-policy actor-critic algorithm. The proposal has analyzed the convergence qualitatively and supported it with empirical results. Moreover, the TD methods have been compared with gradient-TD methods that minimize projected Bellman error [35]. [109] has proposed an actor-critic-identifier that could provably approximate the HJB equation without the knowledge of system dynamics. After the learning step, the scheme showed improved process stability. However, this algorithm required knowledge of the input gain matrix. [110] has used a nominal controller as a supervisor to guide the actor and to yield a safer control in a simulated cruise control system. [111] has proposed to learn the solution of an HJB equation for a partially unknown input-constrained system without satisfying the persistent excitation conditions while preserving stability. By considering the Lyapunov theory, [112] has designed a fault-tolerant actor-critic algorithm and tested its stability on the Van der Pol system. [113] has formulated an input-constrained nonlinear tracking problem by using the HJB equation and a quadratic cost function to define the value function. The scheme has obtained an approximate value function with an actor-critic algorithm. [114] has combined classification and time-series prediction techniques to solve an optimal control problem and showcased the effectiveness of the proposed algorithm on a simulated continuous stirred tank reactor (CSTR) and a simulated nonlinear oscillator. The mean actor-critic algorithm, introduced in [115], estimates the policy gradient using a smooth Q-function, averaging over the actions to reduce variance. The algorithm has demonstrated promising results on Atari games, which can be extended to stochastic process control applications. [116] has utilized an event-triggered actor-critic scheme to control a heating, ventilation, and air conditioning (HVAC) system. In addition, more recent studies on different actor-critic algorithms and their applications have been reported in [40, 117–126].

2.4.4.9 Techniques to Improve Actor-Critic Algortihms

Several methods have been proposed to improve value estimation in RL [127–129], which can be used in actor-critic algorithms. Moreover, different techniques [77, 130] have been reported to improve the sample efficiency (i.e. to reduce the amount of data needed to learn the optimal policy). Unlike these techniques that used experience replay [131] or supervised data [132], ”parallel learning” makes use of multiple randomly initialized workers (**local networks**) that interact with different instances of the environment independently to reduce the variance in the policy during learning. These workers have the same infrastructure as a **global network**, and after collecting k-samples, the workers' individual experience is used to update the parameters of the global network. This reduces the amount of memory used and improves exploration because workers have independent trajectories. Task distribution can be performed via multiple machines [133] or multiple central processing unit (CPU) threads of a single computer [73]. Fig. 2.8 shows an example of this implementation for the A3C algorithm where each worker interacts with its environment to generate the experience that is used to update the global network. A single worker represents the A2C algorithm in this structure.

The optimal policy and the optimal critic are different in each process and they are often unknown *a priori*. Monte Carlo-type methods calculate empirical return (given in Eqn. (2.49)) at the end of the process (or an episode), which may be lengthy and noisy. Similar to Pavlovian conditioning [134] in psychology, temporal difference (TD) learning predicts the value of the current state. Unlike Monte Carlo methods, it makes predictions for a small horizon, as low as one step. This converts the infinite horizon problem into a finite horizon prediction problem. Instead of calculating the expected return, the critic network can be updated using $k-$step ahead estimation of temporal difference error, δ, as shown in Eqn. (2.86).

$$\delta\left(x_t|\omega_L\right) = \sum_{i=0}^{k-1}\left(\gamma^i R_{t+i} + \gamma^k V\left(x_{t+k}|\omega_L\right)\right) - V\left(x_t|\omega_L\right) \tag{2.86}$$

Here, δ is the temporal difference error for state x at a discrete sampling instant, t, given critic parameters of the local network w_L and k represents the horizon length. If k approaches infinity, the summation terms converge to the empirical return given in Eqn. (2.49). Baseline $V(x_t|\omega_L)$ is used to reduce the variance compared to the policy gradient algorithm [35]. At the end of k steps, parameters of the global network (i.e. θ_G and ω_G) are updated using Eqns. (2.67) and (2.68).

In addition to the abovementioned topics, some outstanding research topics include but are not limited to state/reward prediction, constraint enforcement on state/action spaces, sample efficiency (using samples/models), process safety, learning/process stability, robustness, episodic/continual learning, and exploration/exploitation.

Chapter 3

Actor–Critic Reinforcement Learning and Application in Developing Computer-Vision-Based Interface Tracking *

This chapter synchronizes control theory with computer vision by formalizing object tracking as a sequential decision-making process. A reinforcement learning agent successfully tracks an interface between two liquids that is often a critical variable to track in many chemical, petrochemical, metallurgical and oil industries. This method utilizes less than 100 images for creating an environment, from which the agent generates its own data without the need for expert knowledge. Unlike supervised learning methods that rely on a huge amount of parameters, this approach requires much fewer parameters which naturally reduces its maintenance cost. Besides its frugal nature, the agent is robust to environmental uncertainties such as occlusion, intensity changes and excessive noise. From a closed-loop control context, an interface location-based deviation is chosen as the optimization goal during training. The methodology showcases reinforcement learning for real-time object-tracking applications in the oil sands industry along with the presentation of the interface tracking problem.

*This chapter has been published as: **Dogru, O.**, Velswamy, K. and Huang, B., 2021. Actor–Critic Reinforcement Learning and Application in Developing Computer-Vision-Based Interface Tracking. Engineering, 7(9), pp.1248-1261.

3.1 Introduction

Oil sands ore contains bitumen, water and minerals. Bitumen is a high-viscosity hydrocarbon mixture, which can be extracted through several chemical and physical processes. The product is further treated in upgrader units or refineries ([135]) to obtain more valuable by-products (e.g. gasoline, jet fuel, etc.). Oil sands are mined from open pits and loaded into trucks to be moved into the crushers ([126]). Following this, the mixture is treated with hot water for hydro-transporting it to the extraction plant. Aeration and several chemicals are introduced to enhance the process. In the extraction plant, the mixture is settled down in a **primary separation vessel (PSV)**. A water-based oil sands separation process is summarized in Fig. 3.1.

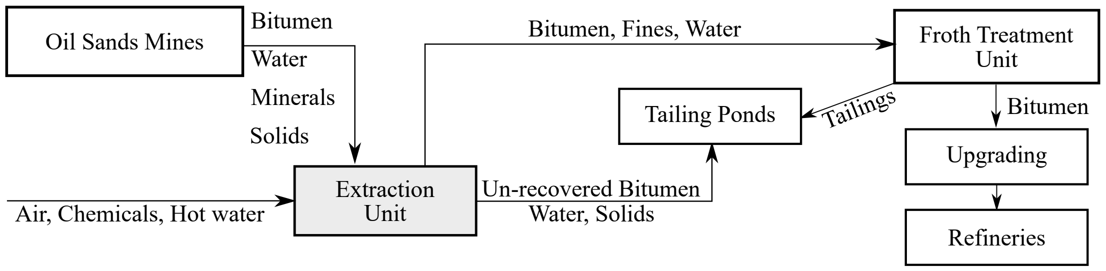


Figure 3.1: A simplified illustration of water-based oil sands separation process. PSV is in the extraction unit.

During the separation process inside the PSV, three layers are formed: froth, middlings and tailings as shown in Fig. 3.2. An **interface** (will be called **FMI** henceforth) is formed between the froth and middlings layer. Its level with reference to the PSV unit influences the quality of the extraction.

To control the FMI level, it is crucial to have reliable sensors. Traditionally, differential pressure (DP) cells, capacitance probes or nucleonic density profilers are used to monitor its level. However, they are either inaccurate or are reported to be unreliable ([136]). Sight glasses are used to manually monitor the interface for any process abnormalities. To utilize this observation in closed-loop control, [136] proposed using a camera as a sensor. This scheme utilized an edge detection model with particle filtering on the images to obtain the FMI level, using which feedback control was established. More recently, [137] combined edge detection with dynamic frame differencing to detect the interface. This method directly uses an edge detection technique

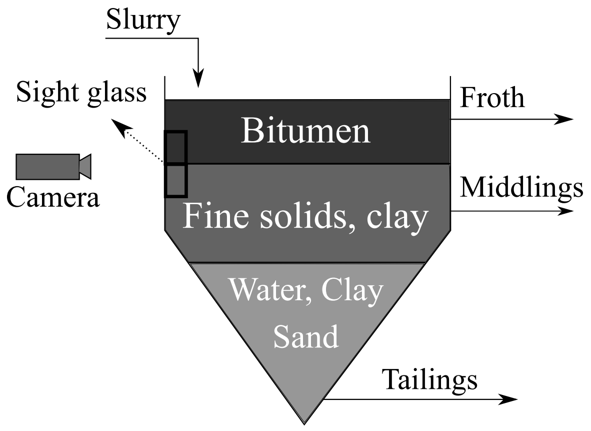


Figure 3.2: A schematic of PSV. During the separation process, three layers are formed. The camera is used to monitor the interface between the middlings and the froth layers to control the FMI level optimally.

to detect the interface along with a frame comparison mechanism that estimates the quality of the measurement and also detects faults. [138] used a mixture of Gaussian distributions to model the froth, interface and middlings appearances and predicted the interface using spatio-temporal Markov random field. Despite these techniques addressing several challenges, utilizing *models* based on the interface appearance or behaviour, they fail to address the sensitivities to uncertain environmental conditions such as occlusion and excessive/non-Gaussian noise.

Supervised learning (SL) methods try to build a map from input (i.e. image, x) to output (i.e. label, y) data by minimizing a cost (or loss) function. Usually, the cost function is convex and the optimal parameters are calculated by applying a stochastic gradient descent algorithm ([81,139]) to the cost function. **Unsupervised learning (UL)** methods, on the other hand, are used to find the hidden features in the unlabeled data (i.e. uses x only) ([140]). The goal is usually to compress the data or find similarities within the data. Nevertheless, UL techniques do not consider the impact of the input on the output, even if there exists such a causal relationship. In computer vision, these methods are implemented using convolutional neural networks (CNN). CNN is a parametric function that applies convolutional operation on the inputs. It can extract abstract features by processing not just a pixel, but also its neighbouring pixels. It is used for classification, regression, dimensionality reduction, etc. ([141–144]). Even though it has been used for decades

([145–147]), only lately has it gained significant popularity in different domains ([148–151]). This is owing to the developments in hardware technology ([152]) and data availability ([153]). Parallel to these, recurrent neural network (RNN) is used for time-series prediction, where the previous output of the network is fed back into itself ([154]). This can be considered a recursive matrix multiplication. However, vanilla RNN ([155]) suffers from diminishing or exploding gradients, because it repeatedly feeds the previous information back into itself leading to uneven back propagated data sharing in between hidden layers. Therefore it tends to fail when the data sequence is arbitrarily long. To overcome this issue, more complex networks such as long short-term memory (LSTM) ([156]) and gated recurrent unit ([157]) were proposed. These networks facilitate data transfer between hidden layers to make learning more efficient. More recently, a variant of LSTM called convolutional LSTM (ConvLSTM, [158]) has been reported to improve LSTM's performance by replacing matrix multiplications with convolutional operations. Unlike fully connected LSTM, ConvLSTM receives an image rather than one-dimensional data and utilizes spatial connections that are present within the input data and enhances estimation. Networks with many layers are considered *deep* structures ([159]). Various deep architectures have been proposed ([160–164]) to enhance the prediction accuracy even further. However, these structures suffer from over-parameterization (i.e. number of training data points is less than the number of parameters). Several regularization techniques (e.g. dropout, L2 etc.) ([148]), as well as transfer learning (also called fine-tuning) methods ([165,166]), try to find a workaround to improve the network's performance. However, the transferred information (e.g. network parameters) may not be general enough for the target domain. This becomes significant especially when training data is not sufficient or their statistics are significantly different. Moreover, currently efficient transfer learning for recurrent networks remains an opportunity.

Reinforcement learning (RL) ([35]) combines the advantages of both of the techniques and formalizes the learning process as a Markov decision process (MDP). Inspired from animal psychology ([167]) and optimal control ([34,37,168–171]), this learning scheme involves an *intelligent* agent (controller). Unlike SL or UL methods, RL does not rely on an off-line/batch dataset but generates its own data by interacting with the environment. It evaluates the impacts of its actions by considering immediate consequences and predicts the value via **roll-out**. Hence, it is more suitable for real or continuous processes, where decision-making for complex systems is involved. However, in data-driven schemes, data distribution may be significantly different during training, which may cause a high variance of estimations ([35]). **Actor-critic** methods have been proposed ([55, 69, 104]) to combine the advantages of value estimation with that of the policy gradient. This approach segregates the agent into two parts: **actor** decides which action to take, **critic** estimates the goodness of that action using an action-value ([74]) or a state-value ([73]) function. These methods do not rely on any labels or system models. Therefore, **exploration** of the state/action space is an important factor that affects the agent's performance. In system identification ([29, 172, 173]), this is known as an identification problem. Different methods address the exploration issue ([35, 72, 73, 174–179]).

As a subfield of machine learning ([180–182]), RL is used in but not limited to process control ([37, 40, 119, 126, 183–187]), game industry ([39, 131, 188–194]), robotics and autonomous vehicles ([195–198]).

FMI tracking can be formulated as an object-tracking problem, which can be solved in one or two steps using detection-free or detection-based tracking approaches respectively. Previous works ([199–201]) used RL for object detection/localization that can be combined with a tracking algorithm. In the case of such a combination, the tracking algorithm also needs to be reliable and fast for real-time implementations. Several object-tracking algorithms, including multiple object tracking using RL, have been proposed ([202–207]). The schemes proposed combined pre-trained object detection with RL-based tracking or a supervised tracking solution. These simulations were carried out under ideal conditions ([208, 209]). The performance of object detection-based methods often depends on detection accuracy. Even if the agent learns to track based on a well-defined reward signal, one should ensure that the sensory information is (or their features are) accurate. Model-based algorithms often assume that the object of interest has a rigid or a non-rigid shape ([137]) and the noise or the motion has a particular pattern ([136]). These assumptions may not hold when unexpected events occur. Therefore a model-free approach may provide a more general solution.

Since a CNN may extract abstract features, it is important to analyze it after training. Common analysis techniques utilize the information of the activation functions, kernels, intermediate layers, saliency maps, etc. ([161, 210–212]). In the RL context, a popular approach has been to reduce the dimensions of the observed features using t-distributed stochastic neighbour embedding (t-SNE, [213]) to visualize the agent in different states ([189, 214, 215]). This helps cluster the behaviour with respect to different situations that the agent encounters. Another dimensionality reduction technique, namely uniform manifold approximation and projection (UMAP, [216]), projects the high dimensional input (which may not be meaningful in Euclidean space) into Riemannian space. This way, the dimensionality of nonlinear features can be reduced.

Fig. 3.3 illustrates a general control hierarchy in process industries. In a continuous process, each level in the hierarchy interacts with the other at different sampling frequencies. The interaction starts at the instrumentation level, which affects the upper levels significantly. Recently, [126] proposed a solution for the execution level. However, addressing other levels remains challenging. Here, a novel interface tracking scheme based on RL that is trained for a model-free sequential decision-making agent is being proposed. This chapter

- focuses on the instrumentation level to improve the overall performance of the hierarchy,
- formulates interface tracking as a model-free sequential decision-making process,
- combines CNN and LSTM to extract spatio-temporal features without any explicit models or unrealistic assumptions,
- utilizes DP cell measurements in a reward function without any labels or human intervention,
- trains the agent using temporal difference learning that allows the agent to learn continuously in a closed-loop control setting,
- validates robustness amidst uncertainties in an open-loop setting,

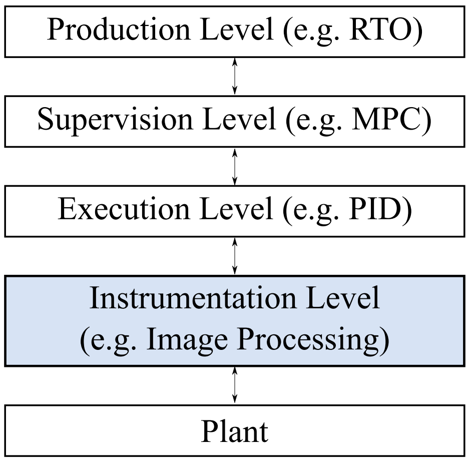

Production Level (e.g. RTO)
Supervision Level (e.g. MPC)
Execution Level (e.g. PID)
Instrumentation Level (e.g. Image Processing)
Plant

Figure 3.3: A general control hierarchy in process industries. RTO refers to real-time optimization, MPC is model predictive control, and PID stands for the proportional-integral-derivative controller.

- analyzes the agent's beliefs in a reduced feature space.

This chapter is organized as follows: preliminary information is provided in Section 3.2, interface detection is formulated in Section 3.3, training and test results are presented in Section 3.4 in detail, and conclusions are drawn in Section 3.5.

3.2 Background

Reinforcement learning is a rigorous mathematical concept ([34, 35, 37]), where an agent learns a behaviour that maximizes an overall return in a dynamic environment. Similar to a human being, the agent learns how to make intelligent decisions by considering future rewards. This implies contemplating temporal aspects of the observations, unlike simple classification or regression approaches. This allows RL to be used under uncertain conditions ([169]) with irregular sampling rates. Its versatile nature makes it adaptive to different environmental conditions and allows it to be transferred from simulation environments to real processes ([197]).

3.2.1 Markov Decision Processes (MDPs)

Markov decision process formulates discrete sequential decision-making processes via a tuple, $\mathcal{M}$, that consists of $\langle \mathcal{X}, \mathcal{U}, \mathcal{R}, P, \gamma \rangle$, where $x \in \mathcal{X}$, $u \in \mathcal{U}$, $r \in \mathcal{R} \subset \mathbb{R}$ are the **state**, **action** and **reward** respectively. $P(x', r|x, u)$ is the system dynamics, or state transition probabilities, which may be deterministic or stochastic. It satisfies Markov property ([217]), i.e. the future state depends solely on the current state, not the history prior to that. In this work, system dynamics are unknown to the agent to make this approach more general. Discount factor $\gamma \in [0, 1)$ is a weight for the future rewards to make their summation bounded. Stochastic policy, $\pi(u|x)$, is a mapping from the observed system states to the actions.

In an MDP, the agent observes a state $x_0 \sim \sigma_0$, where σ_0 represents the distribution of the initial states. It then selects an action $u \sim \pi(u|x)$ that carries the agent to the next state $x' \sim P(x', r|x, u)$ and yields a reward $r \sim P(x', r|x, u)$. By utilizing the sequence (i.e. x, u, r, x) the agent learns a policy, π, that leads to maximizing discounted return, G, as defined in equation (3.1) ([35]).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.1}$$

where t represents a discrete time step. The state-value, $v_\pi(x)$, and the action-value functions, $q_\pi(x, u)$, are calculated using Bellman equations (3.2)-(3.3).

$$\begin{aligned} v_\pi(x) &= \mathbb{E}_\pi \left[G_t | X_t = x\right], \qquad \forall x \in \mathcal{X} \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | X_t = x\right] \\ &= \sum_u \pi(u|x) \sum_{x'} \sum_r Pr\left(x', r|x, u\right) \left[r + \gamma v_\pi(x')\right] \qquad (3.2) \\ q_\pi\left(x, u\right) &= \mathbb{E}_\pi \left[G_t | X_t = x, U_t = u\right], \qquad \forall x, u \in \mathcal{X} \times \mathcal{U} \\ &= \sum_{x'} \sum_r Pr\left(x', r|x, u\right) \\ &\qquad \left[r + \gamma \sum_{u'} \pi\left(u'|x'\right) q_\pi\left(x', u'\right)\right] \qquad (3.3) \end{aligned}$$

After the value functions are estimated for each state, the optimal value functions can be found using equations (3.4) and (3.5).

$$\begin{aligned} v^*_\pi(x) &= \max_\pi v_\pi(x), \forall x \qquad (3.4) \\ q^*_\pi(x, u) &= \max_\pi q_\pi(x, u), \forall x, u \in \mathcal{X} \times \mathcal{U} \\ &= \mathbb{E}[R_{t+1} + \gamma v^*(X_{t+1}) | X_t = x, U_t = u] \qquad (3.5) \end{aligned}$$

Following that, the optimal policy, π^*, can be found as

$$\pi^*(x) = \arg\max_u q^*_\pi(x, u) \tag{3.6}$$

For large-scale problems, linear or nonlinear function approximation techniques can be used to find approximate value functions by $\widehat{Q}(x, u|\omega)$, $\widehat{V}(x|\omega)$ or both, where ω is the parameters of the approximation. These structures are also called *critic.* This work focuses on the state-value estimation and simplifies its notation as $V(\cdot)$.

3.3 Formulating the interface tracking as a sequential decision making process

3.3.1 Interface Tracking

A model is a mathematical means of describing the process dynamics that can occur either in a physical/chemical/biological system ([218]) or in a video ([219]). The models derived for images often suffer from inaccuracies when there is an unexpected event (e.g. occlusion). To overcome this, either the information from the last valid observation is used in the next observation ([137]) or the images are reconstructed ([219]). Though these may substitute actual measurements for a short period of time, prolonged exposure can deteriorate closed-loop stability. As a consequence, if the FMI's level is too low, the bitumen from the froth layer drains into the tailings. This lowers the product quality and creates environmental footprints. In contrast, if its level is closer to the extraction point, the solid particles in the froth being extracted complicate downstream operations ([136]). Since deviations in the FMI level affect the downstream processes, it is important to regulate the FMI at an optimum point.

Reinforcement learning can address inaccuracies during occlusion and excessive noise. This can be done by combining DP cell measurement or measurement from any other reliable instrument with the current FMI prediction by the agent to provide an accurate cost in the reward function, without external labels such as bounding boxes, during the training phase. Removing the dependence upon such labels minimizes human error. To achieve this, an agent can move a cropping box on the vertical axis over the PSV's sight glass and compare its center with the DP cell measurement.

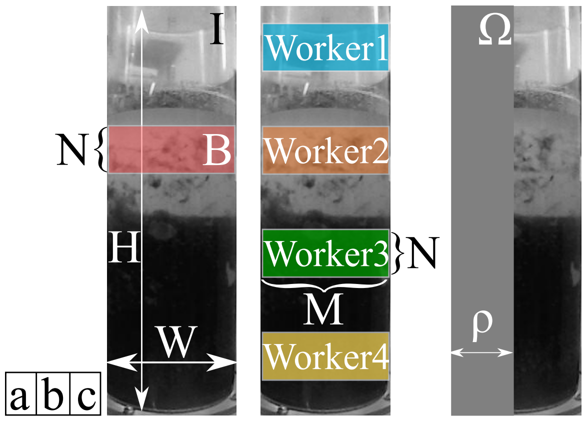


Figure 3.4: A frame (I) obtained using camera. **(a)** Sizes of the image ($H \times W$) and the cropping box ($N \times W$). **(b)** Sizes of the cropping boxes ($N \times M$) and the initial cropping box positions. **(c)** An example occlusion with its ratio, ρ.

Based on this deviation, the agent can move the box to an optimal position, where the center of the box matches that of the FMI. This deviation-minimizing feedback mechanism is inspired by control theory and it can enhance an image-based estimation using the measurement obtained from the real process.

Consider a gray-scale image, I, sampled from a video stream as $I \in \mathbb{R}^{H\times W}$ with arbitrary width, W, and height, H, which captures the entire PSV. Consider a rectangular cropping box, $B \in \mathbb{R}^{N\times M}$, that has an arbitrary width, M and height N, where $\{N\colon N = 2\hat{z} - 1, \hat{z} > 1 \in \mathbb{N}\}$ and $\hat{z}$ is the center of the rectangle. An example image and a cropping box are shown in Fig. 3.4-a. This rectangle crops I at $\hat{z}$ into a size of $N \times M$. For the sake of completeness, $H > N$ and $W = M$. Consider an interface measurement obtained from a DP cell at time t as z. Note that the DP cell is used only in the off-line training of the RL agent and can be replaced by other interface measurement sensors, which are considered to be accurate in off-line laboratory environments. Components of the MDP for this problem then can be defined as:

States: The pixels inside the rectangle, $\mathbf{x} \in B \subset \mathcal{X} \subseteq I$. These pixels may be thought of as $N \times M$ independent sensors.

Actions: Move the center of the cropping box up or down by 1 pixel, or freeze. $\mathbf{u} \in \mathcal{U} = \{-1, 0, 1\}$.

Reward: The difference between the DP cell measurement and the position of the center of the box (with reference to the bottom of the PSV), at each time step, t given in equation (3.7).

$$R_t = -|z_t - \hat{z}_t| \tag{3.7}$$

The relation between u_t and $\hat{z}_t$ is given as equation (3.8).

$$\hat{z}_t = \hat{z}_0 + \sum_{i=0}^{t-1} u_i \tag{3.8}$$

where $\hat{z}_0$ is an arbitrary initial point, and the summation term represents the actions taken up to the t-th instant ($u_i = +1$ for up, $u_i = -1$ for down).
Discount factor: $\gamma = 0.99$.
The goal of this agent is to generate a sequence of actions to overlay the cropping box, B, on the vertical axis of the PSV with the interface at its center. To achieve this, the agent needs to make long-term planning and preserve the association between its actions and the information obtained from DP cell measurement. The flowchart of the proposed scheme is shown in Fig. 3.5. In addition, Fig. 3.6 shows the networks in detail. More details about the ConvLSTM layer can be found in [158]. Unlike the previous works ([137, 138]) that make predictions in the state space, this approach optimizes the value and the policy spaces, by using equations (2.86), (2.67), and (2.68) respectively. Moreover, the CNN and ConvLSTM layers are updated by using equation (3.9).

$$\begin{aligned} \Psi \quad &\leftarrow \Psi + 0.5 \times \alpha_c \nabla_{\Psi_L} \delta\left(\cdot|\Psi_L\right)^2 \\ + \quad &0.5 \times \alpha_a \nabla_{\Psi_L} \delta\left(\cdot|\Psi_L\right) \ln \pi\left(\cdot|\Psi_L\right) + \beta \pi\left(\cdot|\Psi_L\right) \ln \pi\left(\cdot|\Psi_L\right) \end{aligned} \tag{3.9}$$

where $\Psi = [\psi_{CNN}, \psi_{ConvLSTM}]$ represents the parameters of the CNN and the ConvLSTM layers. This scheme trains the entire network end-to-end by using only the TD error. Multiple workers ([73]) that are initialized at different points (Fig. 3.4-b) can be used to improve the exploration and hence generalization.

After a sub-optimal policy is found, the agent is guaranteed to find the interface in a limited time-step k, independent of the initial point as shown in Lemma 1.

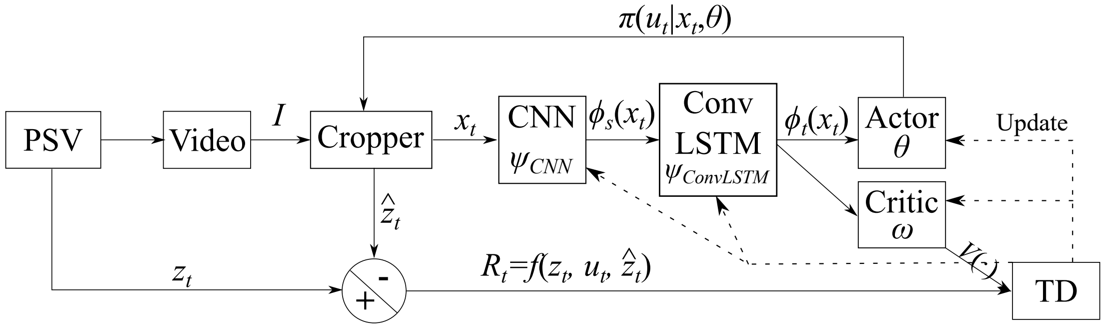


Figure 3.5: Flow diagram for proposed learning process. The update mechanism is shown in equations (2.67) and (2.68) with *k*-step policy evaluation, as shown in equation (2.86). Detailed structures of CNN, ConvLSTM, Actor and Critic are given in Table 3.1.

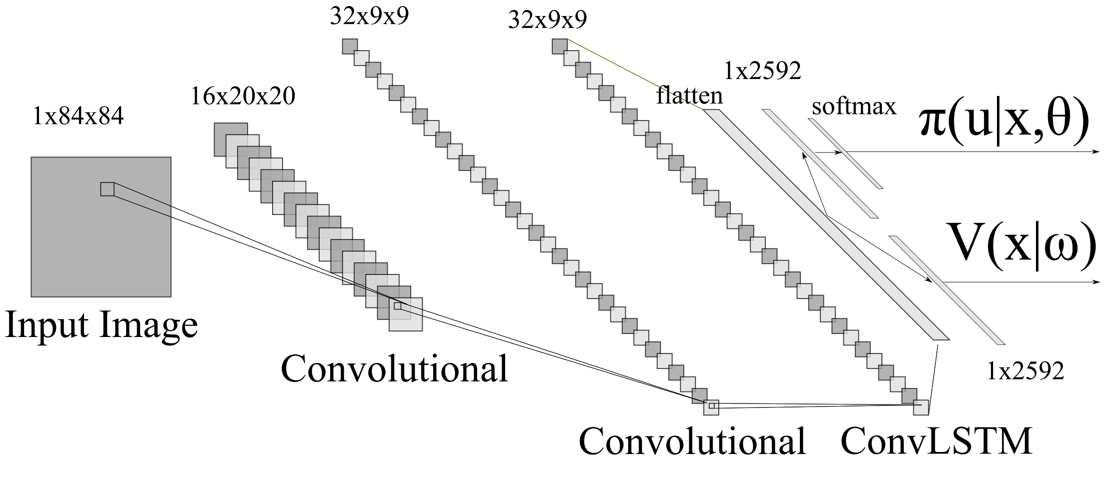


Figure 3.6: A detailed structure of the CNN, ConvLSTM, Actor, Critic networks.

Lemma 1 () *At any time t, for a constant z_t, with*

$$P = 1, \exists k : z_t - \left(\hat{z}_0 + \sum_{i=0}^{k} u_i\right) = 0$$

$(\forall u \sim \pi(\cdot|\theta^*)) \wedge (\forall x, u \in \mathcal{X} \times \mathcal{U})$*, as* $k \to N$*,*
for $(k \leq N < |\mathcal{X}| \ll \infty) \wedge (\forall z_0, z_t \in \mathcal{Z} \equiv |\mathcal{X}|)$*.*

Assume $\hat{z}_0, z_t \sim \mathcal{Z}$, $\|z_t - \hat{z}_0\|_\infty \leq |\mathcal{X}| \ll \infty$ and sub-optimal parameters θ^*, ω^* are obtained using iterative stochastic gradient descent over a continuous policy function $\pi(\cdot|\theta^*)$. $V(\cdot|\omega^*)$ is a Lipschitz continuous critic network, parameterized by ω, and estimates the value of policy $\pi(\cdot)$ for a given state.
$\because u_i \sim \pi(\cdot|\theta^*), |z_t - \hat{z}_0| \geq |z_t - \hat{z}_0 - u_0| \geq |z_t - \hat{z}_0 - u_0 - u_1|$
$\Rightarrow V_{\pi^*}\left(x' = x(\hat{z}_0 + u_0)\right) \geq V_{\pi^*}(x(\hat{z}_0))$.
Similarly,
$V_{\pi^*}\left(x'' = x(\hat{z}_0 + u_0 + u_1)\right) \geq V_{\pi^*}(x(\hat{z}_0 + u_0))$

$$\because \|z_t - \hat{z}_0\|_\infty \ll \infty, \lim_{k \to N \ll \infty} z_t - \left(\hat{z}_0 + \sum_{i=0}^{k} u_i\right) = 0$$

This derivation can be extended to a variable $z_t \in \mathcal{Z}$.

3.3.2 Robustness to occlusion via training

CNNs interpret the spatial information by considering the connectivity of the pixels and this improves robustness up to a certain point. However, this does not guarantee robustness to occlusion and the agent may fail, even though a good policy is obtained under normal conditions. To overcome this issue, the agent may be trained using synthetically occluded images during the training phase. Another way is to re-calibrate a policy (that is trained by using occlusion-free images) with occluded images.

An occluding object, Ω, with an arbitrary pixel intensity, $\kappa \in [0, 255]$, can be defined as $\{\Omega\colon \Omega \in \mathbb{R}^{H \times (N \times \rho)}\}$, where $\mathbb{E}[\Omega] = \kappa$. $\rho \in [0, 100\%]$ represents the ratio of occlusion, as shown in Fig. 3.4-c. If $\rho = 1$, the agent observes only the occlusion in that video frame (i.e. $\mathbf{x}_t = \Omega$ if $\rho = 100\%$). After defining its size, the ratio of occlusion can be sampled from an arbitrary probability distribution (i.e. continuous or discrete, e.g. Gaussian, uniform, Poisson etc.). During training, the duration of the instance at which the occlusion appears may be adjusted arbitrarily. These can be stochastic or deterministic. That is, the occlusion may appear at a random (or specific) time for a random (or particular) duration. If multiple workers are used, different occlusion ratios at different time instances with different durations may be introduced to each worker. This improves the diversity of the training data, which makes the processing time efficient because the agent does not need to wait for a long time to observe different types of occlusion.

3.4 Results and Discussion

3.4.1 Experimental Setup

A lab-scale setup that mimics an industrial PSV is used for the proposed scheme. This setup allows for the movement of the interface to the desired level using pumps shown in Fig. 3.7. Two DP cells are used to measure the interface level based on the liquid density as described in [138].

Data acquisition and simulation creation: Images were obtained using D-Link DCS-8525LH camera at 15 frames per second (FPS). From the 15 FPS footage, a representative image for each second was obtained. Hence, 80 images from 80 consecutive seconds were obtained with necessary down-sampling. These images were processed to showcase the PSV portion, void of unwanted background. They are then converted into gray-scale images. The DP cell measurements (for the same contiguous time period as the images), which are available in terms of water head (water-in) are converted to pixel positions as given in [137]. After each action is taken, the video frame changes. Every action the agent takes generates a scalar reward, which is later utilized to calculate the TD error that gets used in training the agent's parameters.

3.4.2 Implementation Details

3.4.2.1 Software and network details

Both training and testing phases were conducted using an Intel Core i7-7500U CPU at 2.90 GHz (2 cores, 4 threads), 8 GB RAM at 2133 MHz, and 64-bit Windows using Tensorflow 1.15.0. Unlike deeper networks (e.g. [163]) that consist of tens of

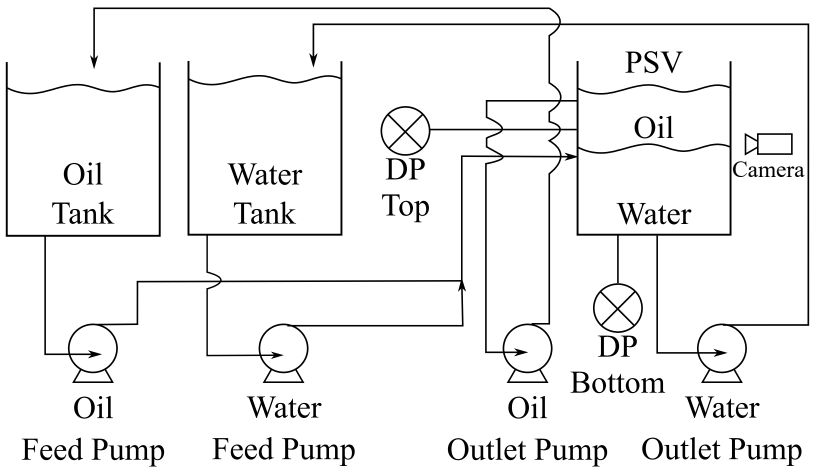


Figure 3.7: Experimental PSV setup.

millions of parameters, this agent consisted of fewer parameters as summarized in Table 3.1. This prevents over-parameterization and reduces the computational time significantly, with the disadvantage of the inability to extract higher-level features ([220]). After each action was taken, the cropping box was resized to 84x84 pixels. Adam optimizer with a learning rate of 0.0001 was used to optimize the parameters of the agent (including CNN, ConvLSTM, actor, and critic), in a sample-based manner. This momentum-based stochastic optimization method has been reported to be computationally efficient ([221]).

Table 3.1: Structure of the Global Network (same as workers).

#	**Layer type**	**Output Dimension**	**Filter size**	**# of parameters**
1	Convolutional	$20 \times 20 \times 16$	8×8	1040
2	Convolutional	$9 \times 9 \times 32$	4×4	8224
3	Convolutional LSTM	$9 \times 9 \times 32$	3×3	73856
4	Fully Connected (Actor)	3		7776
5	Fully Connected (Critic)	1		2592
	Total			93488

3.4.2.2 Training without occlusion

A3C algorithm was used during the experiments to reduce the training time and improve exploration, and convergence to a sub-optimal policy during learning ([73]). All of the initial network parameters were sampled randomly from a Gaussian distribution with zero mean and unit variance. Off-line training was performed after creating a **continuous** trajectory of the interface level by manually ordering 80 unique images, as shown in Fig. 3.8. This trajectory was then repeatedly shown to the agent for 470 steps for 2650 episodes (i.e. an episode consists of 470 steps). At any time, the agent observed only the pixels within the cropping box. The cropping box of each agent was initialized at four different positions as shown in Fig. 3.4-b. The agent's goal was to minimize the deviation of the center of the cropping box with respect to the DP cell measurements, given a maximum velocity of 1 pixel/step. The agent was not exposed to occlusion during training and was capable of processing 20 FPS (i.e. computational execution time) for 4 workers.

3.4.2.3 Fine-tuning (FT) with occlusion

The global network parameters were initialized using the parameters obtained at the end of the training without occlusion. The local networks initially shared the same parameters as the global network. All of the training hyperparameters (e.g. learning rate, interface trajectory etc.) were kept unchanged. The images used in the previous training phase were overlayed with occlusion, whose ratio, ρ, was sampled from a Poisson distribution, as shown in equation (3.10). The distribution, $Pois(x, \lambda)$, is given in equation (3.11).

$$\rho \sim \frac{\rho_{max} - Pois(x, \lambda)}{10} \times 100\% \tag{3.10}$$

$$Pois(x, \lambda) = \frac{e^{-\lambda}\lambda^{x}}{x!} \tag{3.11}$$

Equation (3.10) bounds ρ between 0 and $\rho_{max} = 80\%$ at the beginning of an episode. The shape factor is arbitrarily defined as $\lambda = 1$. In each episode, occlusion occurs at the 200^{th} step to the following 200 steps with a probability of 1. The intent behind fine-tuning is to make sure the agent is robust to the occlusion. The agent, with 4 workers, was trained for an arbitrary amount of 730 episodes until the episodic cumulative reward improved.

Table 3.2: Definition of noisy images based on their identities. $\odot$ represents the Hadamard product.

Identity of noisy image	Noisy image	Condition
1	$I_t = I_t + \nu \odot \zeta$	$t < 300$
2	$I_t = I_t \odot (1 + \nu \odot \zeta)$	$300 \leq t < 700$
3	$I_t = I_t \odot (1 + \nu \odot 2 \times \zeta)$	$700 \leq t$

3.4.2.4 Interface tracking test

For a 1000-step episode, the agent was tested using a **discontinuous** trajectory that contained previously **unseen** images that were either noiseless or were laden with Gaussian noise, $\nu \in \mathbb{R}^{H \times W} \sim \mathcal{N}(0, 1)$, in three ways as shown in Table 3.2. These images were also occluded using a synthetic occlusion, whose constant intensity was arbitrarily selected as the mean of the image (i.e. $\kappa = 128$) while the occlusion ratio varied linearly from $\rho = 20\%$ to 80%.

3.4.2.5 Feature analysis

To illustrate the effectiveness of the network, a previously unseen PSV image was manually cropped starting from the top of the PSV to the bottom. These manually cropped images were then passed one-by-one through the CNN prior to training, the CNN trained as in Section 3.4.2.2, and the CNN fine-tuned as discussed in Section 3.4.2.3 to extract the features. These spatial features, ϕ_s, were then collected in a buffer of size $9 \times 9 \times 32 \times 440$, from which the reduced dimension (2×440) features are obtained using UMAP ([216]). These lower-dimensional features will be represented in Section 3.4.6.

3.4.3 Training

The best policies are obtained at the end of training and fine-tuning when there was no improvement in the cumulative reward for 500 consecutive episodes. Fig. 3.8 shows the trajectories using these policies. Position of the cropping box is initialized with its center at 60% of the PSV's maximum height. At the end of this phase, the agent tracked the interface with a negligible amount of offset. An example obtained

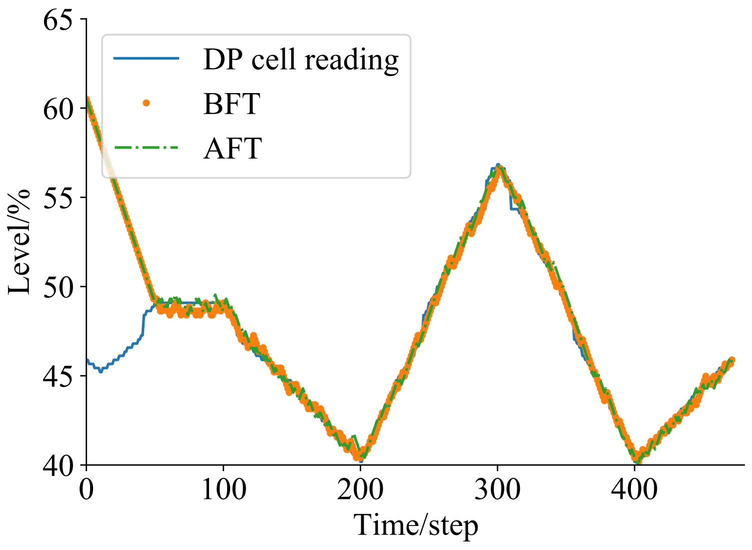


Figure 3.8: Training results at the end of training (2650 episodes) and fine-tuning (3380 episodes).

Table 3.3: Pixel- and level-wise MAE at the end of training and fine-tuning.

	MAE pixel	**MAE level**
After training	4.9852	1.1382
After fine-tuning	4.9597	1.1324

from the 80th step is shown in Fig. 3.9-a. The green star represents where the agent thinks the interface is for the current frame.

3.4.4 FT re-calibration for occlusion

Fine-tuning improved the agent's overall performance, even for the occlusion-free images, by reducing the level-wise mean average error (MAE) by 0.51% as summarized in Table 3.3. This indicates that the agent adapts to the new environmental conditions without forgetting the previous conditions. This was due to the improvements in the value estimation and the policy, which started from near-optimal points. Note that the minimum value for the MAE is limited by the initial position of the cropping box as shown in Fig. 3.8.

Fig. 3.10 shows the cumulative rewards from one of the workers during training and after fine-tuning as shown in solid and dash-dot lines respectively. Note that the

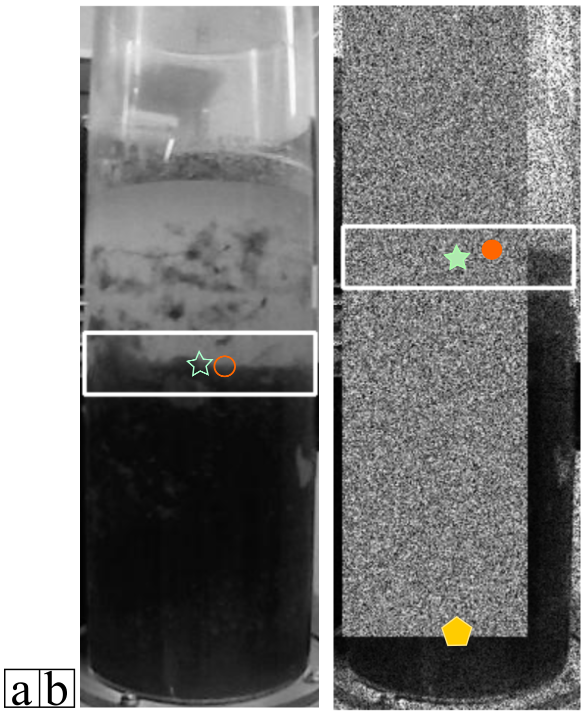


Figure 3.9: **(a)** Training result at the 80th frame. **(b)** Test result after fine-tuning with 80% occlusion and excessive noise, at the 950th step. The white boxes represent the cropping box that the agent controls. The stars represent the center of the cropping box, the circles are the exact interface level. The pentagon is the bottom of the occlusion, which looks like the FMI.

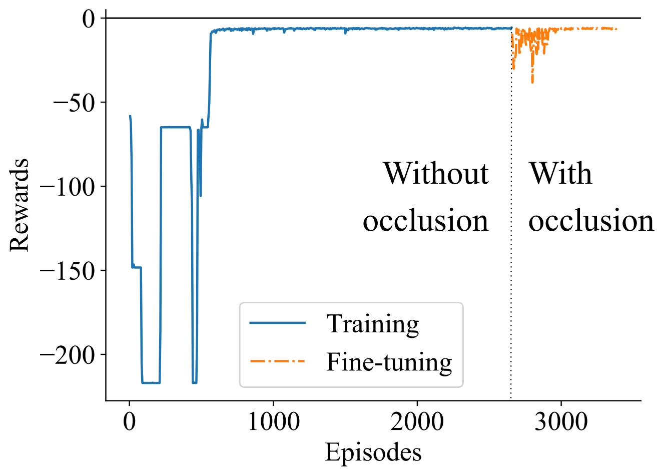


Figure 3.10: Cumulative rewards. The graph shows that the agent can learn the occlusion and track the interface successfully.

initial decrease during fine-tuning was caused by the occlusion because the agent was not able to track the interface level when the occlusion occurred. This new feature was learned successfully by the closed-loop reward mechanism within 400 episodes. Note that the final cumulative reward obtained at the end of fine-tuning is almost the same as that obtained at the end of training. This is because the cumulative reward represents only the tracking performance during training, and depends on the initial position of the cropping box as shown in Fig. 3.8. This value can be zero only if the center of the box and the DP cell measurement overlap completely at the beginning of the episode and the agent tracks the interface without any offsets during the episode. The necessity of fine-tuning is more pronounced when the agent is exposed to unseen environmental conditions such as excessive noise and occlusion as discussed in Section 3.4.5.

3.4.5 Test

Before fine-tuning (BFT): The initial test was conducted at the end of the initial training (i.e. the 2650th episode as shown in Fig. 3.10). Note in the testing (on-line application) phase, DP cell information is not being used and the RL agent works on its own. In fact, even if a DP cell is available, in the field application environment it will not be accurate. Fig. 3.11 shows that the agent was robust to up to 50% occlusion and additional noise prior to fine-tuning. This is a significant improvement over the existing schemes, all of which do not address occlusion. The reason for this is that the neural networks extract more abstract features than edge and histogram information, in both spatial and temporal domains. This is due to the convolutional operations that also smooth out disturbances and improve the agent's overall performance.

On the other hand, any further increase in the occlusion ratio resulted in a failure to track the interface. Since occlusion is of lighter intensity, policy naturally moved towards the bottom of the PSV (where pixels of higher intensity were abundant) to find the interface.

After fine-tuning (AFT): Re-calibrating the agent for occlusion improved its performance significantly as seen from its ability to track the interface more accurately (shown in Fig. 3.11). Additional noise caused its performance to degrade when the interface offset between the consecutive frames was around 5%. However, the agent was successful when this interface offset was reduced to 2.5% as shown in Fig. 3.11. This is because the excessive noise corrupts the image significantly and the agent fails to locate the interface. An example frame obtained at the 950$^{\text{th}}$ frame is shown in Fig. 3.9-b. It should be noted that the noise is accompanied by 80% occlusion. This makes the tracking problem more challenging since the amount of useful information extracted by the agent from the image is significantly reduced. That is, only 20% of the pixels can be used to locate the interface. This performance is due to CNN and ConvLSTM combination. Fig. 3.12 shows the agent's beliefs (predicted by the critic) about the states (obtained from an unseen frame) using parameters obtained from a random network (solid), after training (dash-dot) and after fine-tuning (dot). According to equation (3.2), this figure defines the value of a state, assuming that the best trajectory towards the interface level would be generated by the policy. Fig. 3.12 also shows that prior to any training, the value predicted for any state is similar. However, during training, the agent regrets being in bad states and the DP cell readings reinforce that moving the cropping box closer to the interface (i.e. vertical solid line) yields better value than being further away from it. At the end of fine-tuning, with more data, the agent further improves its parameters and therefore its actions to move the cropper box so that it becomes more accurate. This goes to show that the agent tries to improve its actions based on a constantly changing belief (value). Note that the increase in AFT after a deviation value of 200 corresponds to the yellow pentagon in Fig. 3.9, which looks like an interface and causes an increase in the value function. However, the value obtained from that part is lower than that of the interface, meaning that the agent is more confident when it is close to the star, rather than the pentagon.

3.4.6 Understanding the network: feature analysis

The training and test results focused on the progress of the learning and control abilities of the agent. These alone may not be sufficient to explain whether the agent's decisions are meaningful given an observation in the form of an image.

Fig. 3.13 shows the reduced dimensionality as a two-dimensional graph by repre-

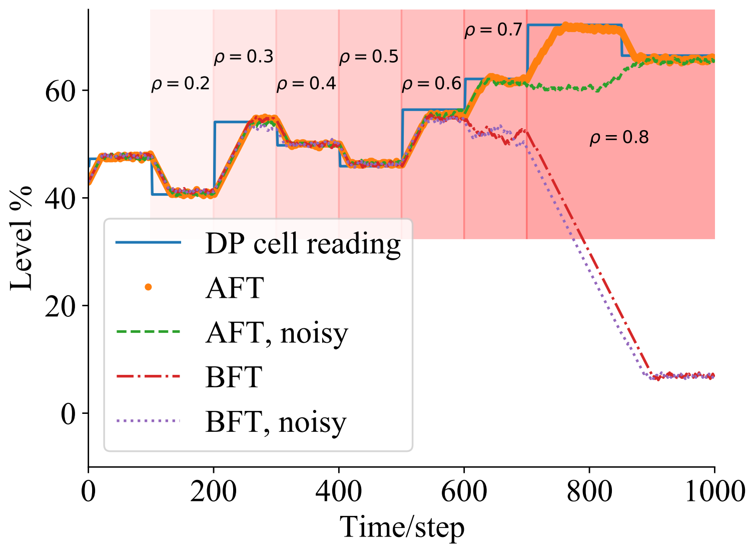


Figure 3.11: Test results: tracking. ρ is the occlusion ratio. E.g., $\rho = 0.8$ means that the image is occluded by 80%.

senting the values of the corresponding cropped images (obtained in Section 3.4.2.5) using gradual intensities of colour. The curve (from left to right) corresponds to the cropped images from top to bottom of the PSV tank side glass as explained in Section 3.4.2.5. The coloured pentagons in Fig. 3.13-a–c correspond to three points in Fig. 3.13-d. According to the results, the features obtained from the network prior to training are similar to each other without any particular arrangement. However, as training proceeds, features with similar values get closer. Combining Fig. 3.13 with Fig. 3.12, it could be inferred that the CNN was able to extract the features in a meaningful way. This is despite using unlabeled data in a model-free context owing to the RL methodology. This was possible because the texture and pixel intensity pattern of each cropped image was successfully converted into the value and the policy functions by employing a CNN-ConvLSTM combination. Also, the reward signal obtained from the DP cell (which was used as a feedback mechanism) trained the agent's behaviour.

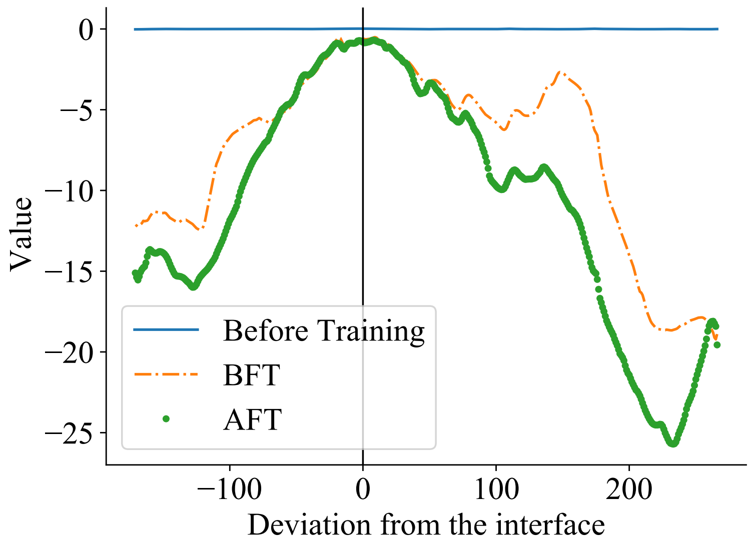


Figure 3.12: Test results: value function versus deviation from the interface.

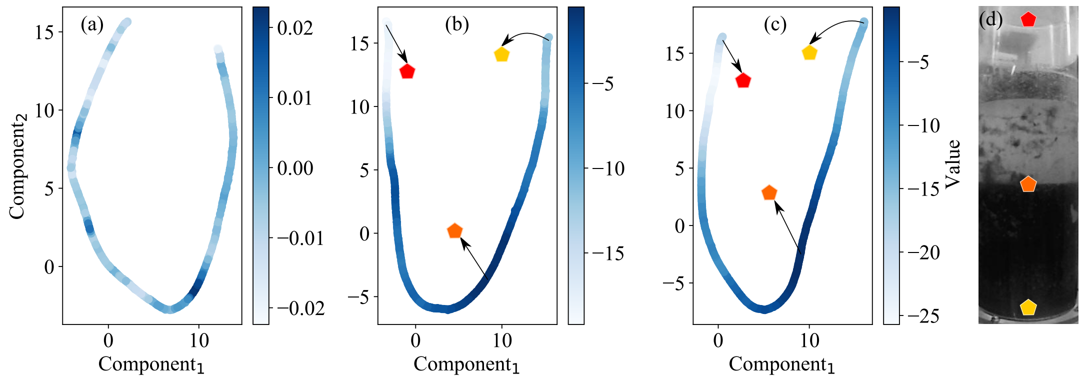


Figure 3.13: Dimensionality reduction applied on the features of the states $\mathbf{x} \in \mathcal{X}$ obtained from an unseen image. The features are obtained using the parameters obtained from a random **(a)**, trained **(b)**, fine-tuned networks **(c)**. The data points are then coloured by their corresponding values. Three regions that correspond to the top and the bottom of the tank and the FMI are highlighted on the unseen image **(d)**. As the agent trains, the extracted features from similar regions get clustered closer in the Riemannian space.

3.5 Conclusion

This chapter proposed a novel RL scheme that targets the instrumentation level of the control hierarchy in order to improve the performance of the entire structure. To achieve this result, interface tracking was formulated as a sequential decision-making process that requires long-term planning. The agent was composed of a CNN and ConvLSTM combination that does not require any shape or motion models and is hence more robust to uncertainties in the process conditions. Inspired by the feedback mechanism used in control theory, the agent utilized readings from DP cells to improve its actions. This technique removes the dependencies on explicit labels that are required for SL schemes. The agent's performance during validation using untrained images under occlusion and noise showed that the interface can be tracked under up to 80% occlusion and excessive noise. An analysis of the high-dimensional features validated the agent's generalization of its beliefs around its observations.

Chapter 4

Online Reinforcement Learning for a Continuous Space System with Experimental Validation [*]

In this chapter, an RL-based low-level controller with safety constraints is developed. Although various RL agents exist, this thesis explores a data-driven approach to learn a constraint function based on process-specfic criteria. During training, the agent learns an optimal policy with the guidance of the regularizing constraints. Moreover, several training strategies are studied to achieve effective exploration, which is a crucial concept in the RL literature. The proposed approaches are implemented on a pilot-scale experimental setup to demonstrate their effectiveness.

4.1 Introduction

Reinforcement learning (RL) has received increasing attention in recent years with significant applications in games ([39,189,222,223]), optimization ([224,225]), control ([226–228]), and autonomous operations ([229, 230]). As a combination of optimal control ([169]) and animal psychology ([167]), this scheme was designed as a trial and error trade-off. In a problem that is solved using RL, an agent observes a set of **states** (*e.g.*, temperature, pressure, position, *etc.*) of an **environment** (*e.g.*, a chemical plant, a simulation model, *etc.*). The agent takes an **action** (*e.g.*, manipulating

[*]This chapter has been published as **Dogru, O.**, Wieczorek, N., Velswamy, K., Ibrahim, F., Huang, B. (2021). Online reinforcement learning for a continuous space system with experimental validation. Journal of Process Control, 104, 86-100.

a process variable), receives the next state and a **reward** and utilizes these in a sequential manner to improve its decisions over time. This improvement works toward optimizing a long-term goal ([40]).

Generic supervised learning (SL) schemes (regression/ classification) ([150,231]) cater to approximate solutions based on parameterized functions. Usually, optimization here constitutes of the level of accuracy achieved with each sample based on an offline, labeled dataset. On the contrary, an RL agent directly interacts with an environment. An environment can be defined as a real system or a model of a system that an agent can act in. Such an environment yields an immediate reward and attains a new state based on the action provided by the agent's policy, governed by the system dynamics. Unlike SL schemes that minimize an immediate loss (e.g. Euclidean norm), RL considers the future impacts of the agent's actions.

Throughout the learning, the agent tries to find the best actions for each state. Systems with small state/action spaces can be represented using lookup tables (*i.e.*, Q-tables [35]). These tables constitute of states, actions and their action-value estimates. Action-value indicates how good an action is given a state and this type of learning is called Q-learning ([38]). Initial schemes involved a brute force type approach (testing all combinations of state-action pairs in a discrete state/action space environment) to obtain the most viable pairs. However, tabular methods are not sufficient to represent continuous state/action space due to the curse of dimensionality ([102]), since these were treated as exact dynamic programming problems. Function approximators address this issue by bringing a generalized approximate solution into the picture. Due to their capabilities in representing complex functions, neural networks ([232]) have been preferred as nonlinear function approximators ([222]).

The action space is in the purview of the agent's estimation of an action. In deterministic cases, this action is obtained as a point estimate, whereas, in stochastic schemes, a distribution of the probable actions is provided. An action is later sampled from this estimated distribution. The states' trajectories are influenced by such choice of actions. A thorough generalization of the RL agent can be made possible only via proper coverage of the state/action space. Though thorough coverage of the continuous states/action spaces is impossible, neural network-based methods can provide an approximate generalized solution to the learning problem. In this case, **exploration** plays a major role in the neural network-based RL solutions.

Exact solutions to the problems represented by Q-tables can be obtained using temporal difference (TD) learning, Monte Carlo sampling, or dynamic programming methods. These iterative methods require initial estimates of the action-values. Often the initial estimates are not the same as the true value, hence a thorough **exploration** of the state/action spaces is necessary. This is similar to the identification problem in process control, where sufficient exploration of the system dynamics is required to control the system successfully. Scoring better rewards using known actions is called **exploitation** of the explored spaces and there is a trade-off between exploration and exploration.

Exploration in Q-tables may be achieved by initializing the action-values with values that are higher than the possible rewards (*i.e.*, biasing the initial estimates) for those state-action pairs ([35]). Upper confidence bound (UCB) action selection ([233]) can be used to encourage selecting less frequently selected actions for better exploration in the multi-bandit problems ([234]). However, these methods cannot be used for continuous space control problems. Alternatively, **ε-greedy** action selection is one of the simplest methods for exploring continuous state/action spaces. In this method, instead of taking an action that gives the maximum reward at a state, the agent takes a random action with a probability of ε (specified by the user, [35]). It is also possible to use a decaying ε scheme, so that the agent reduces the amount of exploration by time while *exploiting* the already-discovered actions. Exploration can also be enhanced by perturbing the state space and selecting actions from a Boltzmann probability distribution ([174]). As an alternative, perturbing the parameters of neural networks may also result in efficient learning ([175]). Combination of a model-based and a model-free algorithms has been reported for a continuous control task ([179]) where the model-based part produced a trajectory (for a finite horizon), which was then optimized by the model-free part by means of action samples. Though there are significant advances in the literature ([176–178]), exploration is still an open challenge for continuous state/action space in real-time process control applications.

Q-learning has been implemented on Atari games ([176]) with high dimensional discrete space environments using neural networks (Deep Q Network or DQN). However, nonlinear function approximators may diverge or yield unstable learning ([39]) and DQN may be insufficient for continuous control tasks ([79]). It has been shown that the instability due to function approximation can be reduced and a sufficient learning can be obtained more quickly ([74]) by using an agent consisting of two elements: actor and critic. Instead of estimating an action-value only, the agent samples an action (actor) and estimates an action-value based on this decision (critic). This two-element structure has also been used in computer vision applications. For example, generative adversarial networks (GANs) employ a **generator** element (actor) that produces an image and a **discriminator** element (critic) that decides whether this image is similar to a labeled image ([68]).

In addition, asynchronous advantage actor-critic (A3C) algorithm ([73]) involves multiple workers that are allocated to different threads of a central processing unit (CPU). As shown in Figure 4.1, these workers interact with their own environments and transfer their experience to a global network, asynchronously. Due to this non-synchronous training, at any time *t*, the experience that the global network acquires is different from that of the other workers. This type of trainings reduces the dependence of the workers on each other, hence improving the exploration. The agent may be trained using a single set point for several hours. However there is no guarantee that the learning will be sufficient to track multiple set points. Moreover, the agent may show unstable behaviour tracking different set points. To overcome this issue and to improve exploration, a single set point "trajectory" (SSPT) can be used to train all the workers. However, such a training scheme may be inadequate to obtain a generalized agent for different regions of operation. In order to improve generalization and enhance exploration further, one possible approach is to take advantage of the independence of the workers and train them on different set point trajectories (MSPT). Another possible approach consists of training the workers gradually: starting from a single set point, then leveraging to a single set point trajectory, afterwards to different set point trajectories. Inspired from [235], this agent is called the proximal agent (PA) in the following sections.

The proposed work in this chapter investigates the differences between these three

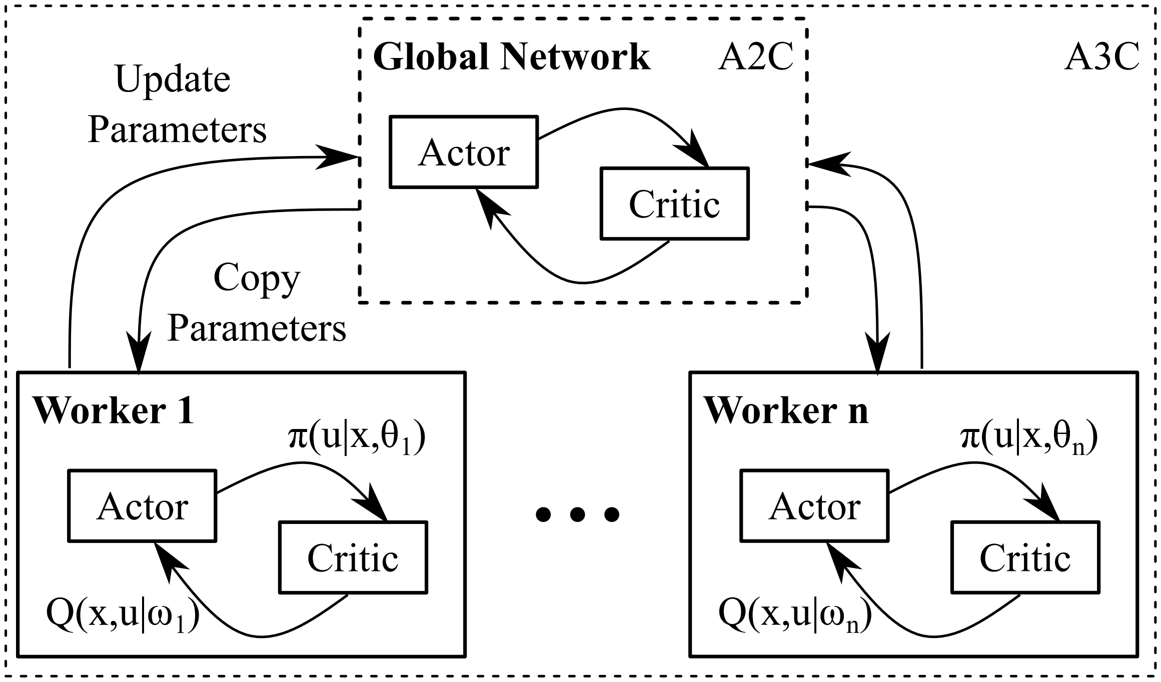


Figure 4.1: Graphical representation of asynchronous (A3C) and synchronous (A2C) advantage actor-critic algorithms.

approaches to find the best approach for training. In addition, it demonstrates how state constraints can yield a safer control.

Originally developed for computer based environments, such as simulations and games, A3C scheme requires multiple instances of the environment class for its execution. However, in process industries, with no two plants being identical, such scheme becomes inconvenient for real-time learning. A predecessor of this scheme, advantage actor-critic (A2C), can be utilized in such scenario as the nearest possible alternative.

In the RL literature, the above-mentioned methods (*i.e.*, Q-learning, A3C, A2C) are called 'model-free', meaning that the agent interacts directly with the environment without prior knowledge of the system dynamics, and does not have to update any process models (unlike model-based methods, [236, 237]). It learns a policy (also a critic in the actor-critic setups) to best act in its environment, based on trial and error. Instead of solving a model or making plans (without executing actions), the agent collects state, action and reward information to improve its behaviour in an iterative manner. To avoid locally optimal policies, the policy and the value functions are usually randomly initialized ([238]). This leads to random actions during the initial stages of interaction with the process. However, such nature of this learning approach may compromise the safe operation of chemical processes and is often time-consuming. Instead, an explicit system model -which cannot be modified by the agent- can be used for agent interactions during simulation-based training.

Process safety is often satisfied by taking safe actions according to the predicted states by using a system model ([112, 239]), or expert data ([240]), which may not be available in all situations. On the other hand, data-driven schemes provide flexible solutions by introducing constraints in the reward or in the loss function to obtain a desired behaviour. These schemes have been used either for exploration ([72, 73]), providing smoother policy updates ([88]), avoiding undesired outcomes ([241]), or other problem-specific requirements. Even though constrained Markov decision processes have been extensively investigated ([103, 242, 243]), their application in real-time processes is still a challenge in the field, which will be addressed in this chapter by using a soft-constrained learning scheme ([241]).

In addition, nonlinearity is an inherent characteristic in most chemical processes. To achieve closed-loop control for such processes, the process model is linearized. However, an RL agent may obtain computationally feasible solutions (by using automatic differentiation methods, [244]) without linearization (as nonlinearities are introduced by means of activation functions, [245]).

For industrial continuous process control, proportional-integral-derivative (PID) controllers and model predictive controllers (MPC) are preferred either due to their simplicity or ability to handle constraints at both the low level and the supervisory level along with their optimality respectively. However, PID controllers require frequent tuning, and real-time optimization using MPC for nonlinear, dynamic, large-scale, multiple-input multiple-output (MIMO) systems could be computationally costly ([246]). Recent works prove that an RL based solution may overcome some of the challanges faced by these controllers when dealing with complex processes ([195, 247]).

In earlier works, the use of a long-short-term-memory- (LSTM-) based RL agent to control air conditioning systems was reported ([119]). Also, control of linear single-input single-output (SISO) and MIMO systems in simulation environments using RL was demonstrated ([183]). RL agents have been used to tune PID and directly control temperature of a simulated continuous stirred tank reactor (CSTR) with discrete state and action spaces ([184]). A model-based RL approach has been utilized to control a quadrotor ([248]) and more recently, a hierarchical RL based control of a gravity separation vessel is presented ([126]). [249, 250] used RL-based controllers for various chemical processes. Moreover, [187] demonstrated that RL can be used to achieve plant-wide control for a vinyl actate monomer process.

To deal with numerous practical issues, such as exploration, safety and sample efficiency, corresponding strategies will be proposed in this work. To best illustrate these strategies, online RL-based control of a MIMO, multi-modal, nonlinear, dynamic hybrid tank system ([251]) will be represented in both simulated and experimental environments. The exploration of state/action spaces and sample efficiency are improved without using any explicit knowledge of the learning process. That is, the agent is not forced to visit unseen parts of the environment by means of neither state-visitation frequency ([252]), nor the value/policy function. The environment here, consists of a continuous state/action space.

This work reports an end-to-end RL solution of a nonlinear process with continuous state and action spaces by:

1. using meta-heuristic model parameter optimization by using experimental data for an experimental three-tank system with hybrid nonlinear dynamics (Step 1),

2. utilizing three different approaches on the simulation level to explore the state/action space (Step 2),

3. comparing the exploration performances of different agents using the extent of exploration (EoE),

4. *in silico* asynchronous advantage actor-critic (A3C/A-A2C) implementation to control the water levels in the tanks,

5. online learning (A2C) using the best *in silico* policy on the real process with a socket connection system ([253]) (Step 3),

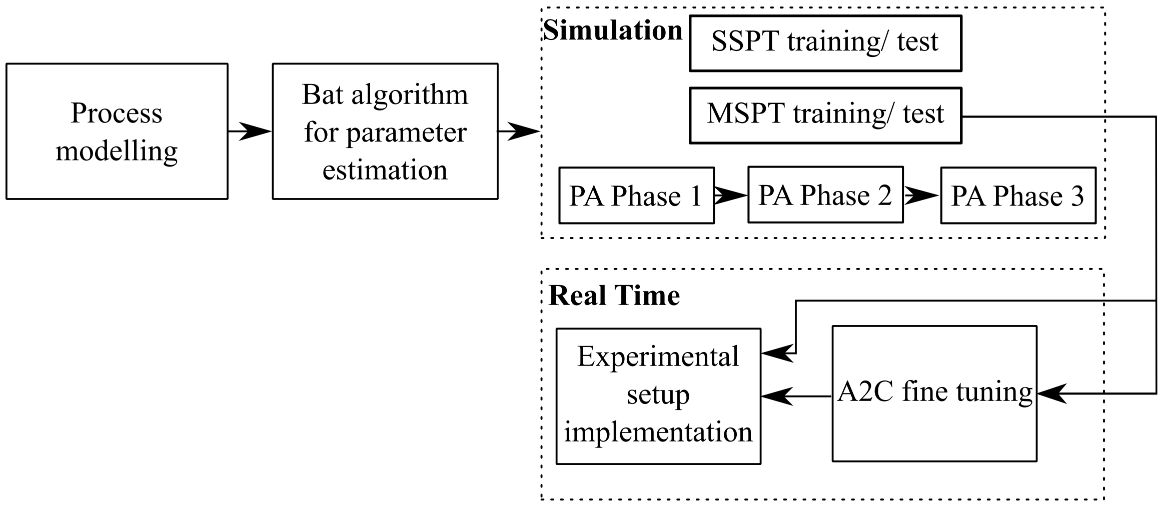


Figure 4.2: Flowchart for the end-to-end implementation. SSPT refers to single set point trajectory, MSPT represents multiple set point trajectory, PA stands for the proximal agent.

6. introducing a soft-constrained scheme to maintain the states within the safe operational range.

This summary is illustrated in Figure 4.2.

This chapter is organized as follows. In Section 4.2, RL and Markov decision process (MDP) are introduced. A soft-constrained learning scheme and a novel performance criterion, extent of exploration (EoE), are shown in Section 4.3. The hybrid tank's first principles model is described together with the related parameter estimation using the metaheuristic bat algorithm in Section 4.4. Results and discussion are presented in Sections 4.5 and 4.6 respectively. Conclusions are summarized in Section 4.7.

4.2 Reinforcement Learning

Reinforcement learning is a nature-inspired machine learning paradigm, where an agent learns a policy that maximizes a reward in a dynamic environment. The learning is based on "intelligent" trial and error, and the agent considers the future impacts of its actions. These properties distinguish RL from other machine learning methods and make it suitable for sequential decision-making under uncertainty ([169]).

4.2.1 Markov Decision Processes (MDPs)

An RL problem can mathematically be represented using an MDP, $M = \langle X, U, R, P, \gamma \rangle$, where $x \in X$, $u \in U$, $r \in R \subset \mathbb{R}$ are the **state**, **action** and **reward** respectively. $P(x', r|x, u)$ defines the state transition probabilities (*i.e.*, process model), which are unknown to the agent for model-free RL. $\gamma \in [0, 1)$ is the discount factor, and $\pi(u|x)$ is the policy that maps the states to the actions that define the behaviour of the agent.

The agent starts at a state x and chooses an action $u \sim \pi(u|x)$, which yields a scalar reward, r, and makes the agent transition to a successor state, $x' \sim P(x', r|x, u)$. Following this sequence (*i.e.*, x, u, r, x), the agent takes a new action and the process continues. During this process, the agent's aim is to learn a policy, π, that maximizes the discounted future reward ([35]), G_t, from time step t, which is given in equation (4.1).

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4.1}$$

The 'effect of the future consequences' by the agent's current actions can be adjusted by changing γ. The state-value, $v_\pi(x)$, and the action-value, $q_\pi(x, u)$, can be calculated using Bellman expectation equations (4.2)-(4.3).

$$v_\pi(x) = \mathbb{E}_\pi \left[G_t | X_t = x \right], \qquad \forall x \in X \tag{4.2}$$

$$q_\pi(x, u) = \mathbb{E}_\pi \left[G_t | X_t = x, U_t = u \right], \quad \forall x, u \in X \times U \tag{4.3}$$

Note that equation (4.2) does not include the action information, thus cannot be used alone to find the best policy. However, its approximated value is used as a baseline in actor-critic algorithms as will be discussed in Section 4.3, and it may be used to modify the properties of the Markov decision process also will be discussed in Section 4.3.2. After the value functions are estimated for each state, the optimal value functions can be found using equations (4.4) and (4.5).

$$\begin{aligned} v_\pi^*(x) &= \max_\pi v_\pi(x), \forall x \in X & (4.4) \\ q_\pi^*(x,u) &= \max_\pi q_\pi(x,u), \forall x, u \in X \times U & \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi^*(X_{t+1}) | X_t = x, U_t = u] & (4.5) \end{aligned}$$

Note the relationship between v^* and q^* that is shown in equation (4.5). Following that, the optimal policy, π^*, can be found as

$$\pi^*(x) = \arg\max_u q_\pi^*(x,u) \tag{4.6}$$

4.3 Formulating control as an RL problem

Value-based methods use value functions to obtain optimal policies, whereas policy-based methods directly optimize the policy, $\pi(u|x,\theta)$. In the latter case, a neural network, parameterized by θ is optimized using gradient ascent on $\mathbb{E}[G_t]$. However, stability, sample efficiency and variance are the main challenges of these methods. Actor-critic methods ([55, 69]), combine policy-based and value-based methods to address these problems by employing two networks that are responsible for 1) choosing actions (*i.e.*, **actor network, parameterized by** θ) and 2) estimating the value function (*i.e.*, **critic network, parameterized by** ω**)**. The networks are trained consecutively to yield better estimates of the policy and the value functions. Several algorithms have been proposed to solve discrete space control problems [40], which are not suitable for continuous space problems. Unlike those methods, the asynchronous advantage actor-critic (A3C) algorithm ([73]) can solve continuous space problems and it consists of multiple workers (with their own actor and critic networks) that asynchronously update a global network. Unlike the Monte Carlo type of approaches that calculate the return for the entire episode and update the global network at the end of each episode, the A3C algorithm updates the global network using k samples (*i.e.*, state, action, reward) via rollout. The value function is estimated at every k steps with a fixed policy. This scheme decreases the variance of the estimates ([73]) compared to "Vanilla" policy gradient algorithm ([35]).

Initially, the global network parameters (*i.e.*, θ_G and ω_G) are sampled from a normal distribution with mean zero and variance one. The global and the worker networks share the same parameters (*i.e.*, $\theta_G = \theta_L$, $\omega_G = \omega_L$), where the subscripts L and G represent the worker and the global networks, respectively. Following this initialization, the workers start interacting with their own environments independently. They generate their own data and make a k-step policy evaluation by using equation (4.7). At the end of k steps (*i.e.*, also called buffer), the global critic and actor-network parameters are updated using equations (4.8) and (4.9).

$$\delta\left(x_t|\omega_L\right) = \sum_{i=0}^{k-1}\left(\gamma^i R_{t+i}\right) - V\left(x_t|\omega_L\right) \tag{4.7}$$

$$d\omega_G \leftarrow d\omega_G + \alpha_c \nabla_{\omega_L}\delta\left(x_t|\omega_L\right)^2 \tag{4.8}$$

$$d\theta_G \leftarrow d\theta_G + \alpha_a \nabla_{\theta_L}\delta\left(x_t|\omega_L\right)\ln\pi\left(u_t|x_t,\theta_L\right) + \beta\pi\left(u_t|x_t,\theta_L\right)\ln\pi\left(u_t|x_t,\theta_L\right) \tag{4.9}$$

where t represents a discrete time step that the agent is in, β is a fixed weight for the entropy term (*i.e.*, $\pi\ln\pi$) and adjusts the degree of exploration, and k is the buffer size. Initially, $d\omega_G = d\theta_G = 0$.

The states constitute the observable process variables, the actions are the manipulated variables, and the reward is a function of the states and the actions as will be explained in Section 4.4.

4.3.1 Constraint-free Learning

This learning scheme was carried out using three schemes to study the agents' exploration abilities. In each scheme, the agent consists of a global and four local networks trained as follows:

Single set point trajectory (SSPT): The control goal of this scheme is to train each worker to track a single set point trajectory.

Multiple set point trajectory (MSPT): The control goal of this scheme is to train each worker to track a different set point trajectory in order to cover most of the state space.

Proximal agent (PA): Inspired by [235], a *proximal agent* consisted of four local networks (workers) was trained gradually. The motivation is to (1) utilize the information obtained in the earlier stages of the training to speed up the training, (2) increase the complexity of the control task gradually to improve generalization and yield a better control performance. First, every worker is trained on a single pair of set points (*Phase 1*), followed by training on single set point trajectories (*Phase 2*), and lastly on four unique set point trajectories (*Phase 3*).

It should be noted that this concept is different from the proximal policy optimization (PPO) algorithm ([79]).

4.3.2 Soft-constrained Learning

Obtaining a constrained nonlinear controller for a nonlinear process is challenging. This involves a multitude of optimization procedures, rendering it infeasible for lower-level implementation ([254]). At higher levels, in real-time optimization, the problem persists even with offline optimization and it becomes cumbersome owing to the complex plant dynamics. An offline optimized online strategy with soft constraints being considered during offline training is proposed. This scheme greatly improves constraint satisfaction on observable states while remaining model-free.

Here, the optimization (4.4) is converted into a constrained problem as shown in equation (4.10)

$$v_\pi^*(x) = \max_\pi v_\pi(x), \forall x s.t.\ J_c^\pi \leq \xi \tag{4.10}$$

where $J_c^\pi = \mathbb{E}[C(x)]$ represents expectation of constraint C, and ξ is the penalty threshold. Instead of solving this problem as a hard-constrained optimization problem, equation (4.10) can be relaxed as shown in (4.11).

$$v_\pi^*(x) = \max_\pi \left[v_\pi(x) - \Lambda\left(J_{co}^\pi - \xi\right)\right], \forall x \tag{4.11}$$

where Λ is a dynamic weight and is used as a Lagrange multiplier with an initial value of zero. The update rule for the dynamic weight is given by equation (4.12).

$$\Lambda_{new} = \Lambda_{old} + \alpha_{co} \sum_{episode} |C_1 + C_2| \times \xi \tag{4.12}$$

where $\alpha_{co} = 1 \times 10^{-6}$ is the learning rate for the weight of the constraints that is empirically selected to minimize the mean squared error (MSE) during learning. C_1 and C_2 represent the constraints on the pumps (p_1 and p_2). Their relationship is given as:

$$C_i = \begin{cases} |p_i - 1|, & \text{if } p_i > 1 \\ |p_i|, & \text{otherwise} \end{cases}, \text{for } i = 1, 2 \tag{4.13}$$

As mentioned below the summation term in equation (4.12), Λ is updated at the end of each episode using all constraint breaches as shown in equation (4.13). Convergence analysis of the method and the related proofs have been provided in [241].

4.3.3 Extent of Exploration

The Jaccard index or intersection over union (IoU) is used to measure similarity between two sets and is defined in equation (4.14).

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.14}$$

where the numerator and the denominator correspond to the intersection and union of the sets respectively. This index has been used in computer vision ([255]) and keyword similarity assessment ([256]). Inspired by this concept, an agent's exploration performance can be benchmarked using **extent of exploration** that involves discovered, against the total operational range of state/action spaces considered. Because the total operational range is limited by physical constraints, here, an empirical limit is considered. Therefore the total operational range is assumed to be a feasible set and it can be normalized between zero and one.

In general, EoE can be defined as given in equation (4.15).

$$\text{EoE} = \frac{D}{T} \tag{4.15}$$

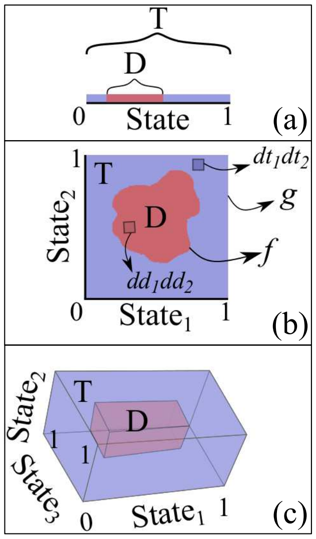


Figure 4.3: EoE for a) 1-D, b) 2-D, c) 3-D state spaces

where

$$D = \int \cdots \int_0^1 f\left(d_1, \ldots, d_n\right)\, dd_1 \ldots dd_n \in \mathbb{R}^n \tag{4.16}$$

is the discovered portion of the total feasible operation range, T,

$$T = \int \cdots \int_0^1 g\left(t_1, \ldots, t_n\right) dt_1 \ldots dt_n \in \mathbb{R}^n \tag{4.17}$$

for n-dimensional state/action spaces. Infinitesimal discrete elements of the states/ actions for arbitrary functions, f and g, are represented by dd_i and dt_i (for $i = 1, ..., n$) respectively. Practically, D and T can be calculated using the sum of finite points obtained from the process, too. However, it should be noted that this would affect the precision of the calculation. In this work, the resolution of EoE calculations for action and state spaces have been empirically determined as $1 \times 10^{-5}\%^{-1}$ and $1 \times 10^{-5} cm^{-1}$, respectively.

For one dimensional state spaces, D and T represent segments (Figure 4.3-a). For a two-dimensional (2-D) state-space, they are 'areas' (Figure 4.3-b). Likewise, they correspond to 'volumes' for 3-D state-spaces (Figure 4.3-c). For the sake of simplicity, the figure illustrates the state spaces only, however, the calculation is the same for action spaces. EoE is bounded between 0 (no exploration) and 1 (full exploration). It provides a scalar representation of the coverage of the state/action space.

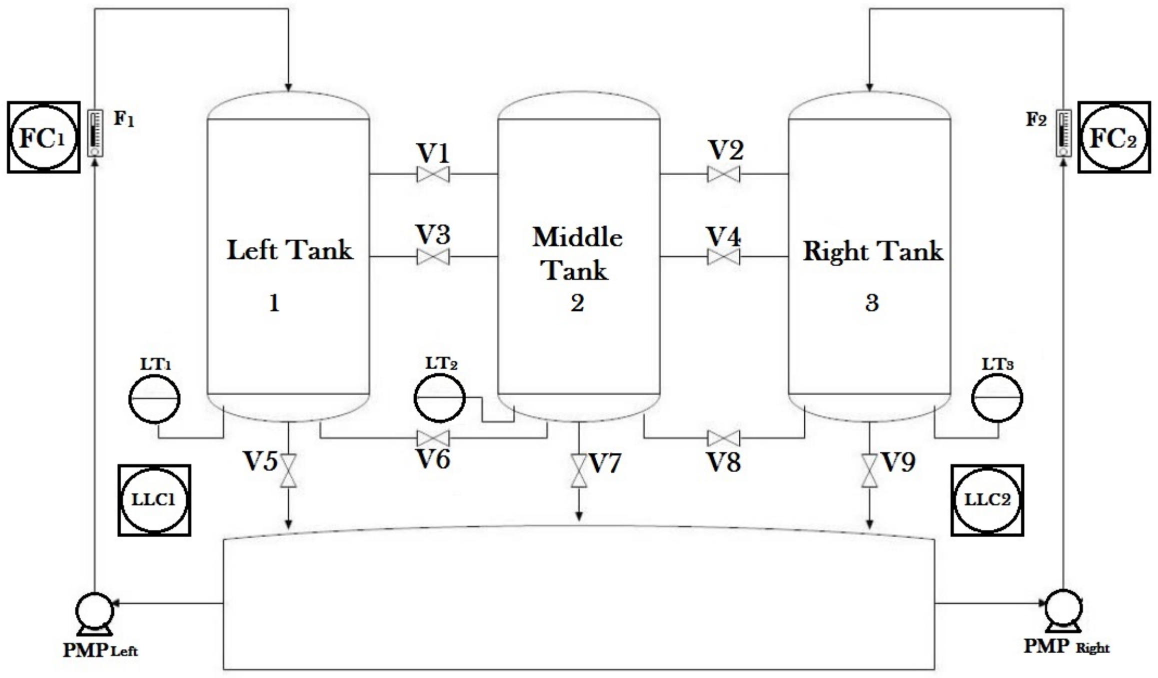


Figure 4.4: Piping and Instrumentation Diagram (P&ID) of the Hybrid Three-Tank.

4.4 The Hybrid Tank System

Process Description and Functionality: As shown in Figure 4.4, the hybrid three-tank (HTT) system ([251]) constitutes of three identical cylindrical tanks installed at the same level, a storage tank below, and two pumps whose outflows (Q_{p_1} and Q_{p_2}) enter tanks 1 and 3 respectively. Valves V3 (for tanks 1 and 2) and V4 (for tanks 2 and 3) control the interactions at the mid level at 15.3 cm. Valves V1 (for tanks 1 and 2) and V2 (for tanks 2 and 3) control the interactions at the top level at 30.6 cm. All these valves are kept open (enabling interaction) throughout the reported simulation and experiments.

There is also a provision for interaction in the bottom level of tanks through valves V6 (tanks 1 and 2) and V8 (tanks 2 and 3) respectively. However, these interactions are blocked by closing the valves.

All three tanks have a discharge pipe each at the bottom to drain water into the storage tank. Each pipe is equipped with a solenoid valve and the tanks are equipped with overflow protection leading into the storage tank.

Differential pressure transmitters (DPTs) are used to measure the water levels in the three tanks.

System Equations: Considering the system configured in Figure 4.4. The first principle model of the HTT, based on mass balance is provided in equations (4.18)–(4.20) whose variables and the parameters are all defined in the nomenclature provided in Table 4.1.

Table 4.1: Nomenclature and the variables used in the differential equations.

Symbol	Definition	Value/unit
g	Gravitational constant	983.991 $\frac{cm}{s^2}$
$H_{1,2,3}$	Water level in tank 1, 2, and 3	cm
Q_{Tank_i}	Total flow rate into tank i	$\frac{cm^3}{s}$
$Q_{p_{1,2}}$	Flow rate of left and right pump	$\frac{cm^3}{s}$
$p_{1,2}$	Speed of left and right pump	%
$K_{1,2}$	Pump coefficient of left and right pump	$\frac{cm^3 s^{-1}}{\%}$
Q_{V_n}	Flow rate through valve n	$\frac{cm^3}{s}$
A_{Tank}	Cross-sectional area of tank	180.54 cm^2
A_V	Cross-sectional area of valve	0.385 cm^2
$H_{V_{1,2}}$	Height of valve 1 and 2 center from the tank	30.6 cm
Ψ, $\widetilde{\Psi}$, $\overline{\Psi}$	Experimental, simulation, mean experimental data	
$H_{V_{3,4}}$	Height of valve 3 and 4 center from the tank	15.3 cm
$C_{V_{1,2}}$	Drain coefficient of valves 1 and 2	0.26
C_{V_3}	Drain coefficient of valve 3	0.29
C_{V_4}	Drain coefficient of valve 4	0.33
C_{V_5}	Drain coefficient of valve 5	0.78
C_{V_7}	Drain coefficient of valve 7	0.69
C_{V_9}	Drain coefficient of valve 9	0.82

$$
\begin{aligned}
\frac{dH_1}{dt} &= \frac{Q_{Tank1}}{A_{Tank}} \\
Q_{Tank1} &= Q_{V_1} + Q_{V_3} + Q_{V_5} + Q_{p_1} \\
Q_{V_1} &= C_{V_1} \times A_V \times sign(max(H_2 - H_{V_1}, 0) \\
&\quad -max(H_1 - H_{V_1}, 0)) \\
&\quad \sqrt{2g\ |max(H_2 - H_{V_1}, 0) - max(H_1 - H_{V_1}, 0)|} \\
Q_{V_2} &= C_{V_2} \times A_V \times sign(max(H_2 - H_{V_2}, 0) \\
Q_{V_3} &= C_{V_3} \times A_V \times sign(max(H_2 - H_{V_3}, 0) \\
&\quad -max(H_1 - H_{V_3}, 0)) \\
&\quad \sqrt{2g\ |max(H_2 - H_{V_3}, 0) - max(H_1 - H_{V_3}, 0)|} \\
Q_{V_4} &= C_{V_4} \times A_V \times sign(max(H_2 - H_{V_4}, 0) \\
&\quad -max(H_3 - H_{V_4}, 0)) \\
&\quad \sqrt{2g\ |max(H_2 - H_{V_4}, 0) - max(H_3 - H_{V_4}, 0)|} \\
Q_{V_5} &= -C_{V_5} \times A_V \times \sqrt{2 \times g \times H_1}
\end{aligned}
\tag{4.18}
$$

$$
\begin{aligned}
\frac{dH_2}{dt} &= \frac{Q_{Tank2}}{A_{Tank}} \\
Q_{Tank2} &= -Q_{V_1} - Q_{V_2} - Q_{V_3} - Q_{V_4} + Q_{V_7} \\
Q_{V_7} &= -C_{V_7} \times A_V \times \sqrt{2 \times g \times H_2}
\end{aligned}
\tag{4.19}
$$

$$
\begin{aligned}
\frac{dH_3}{dt} &= \frac{Q_{Tank3}}{A_{Tank}} \\
Q_{Tank3} &= Q_{V_2} + Q_{V_4} + Q_{V_9} + Q_{p_2} \\
Q_{V_9} &= -C_{V_9} \times A_V \times \sqrt{2gH_3} \\
&\quad -max(H_3 - H_{V_2}, 0)) \\
&\quad \sqrt{2g\ |max(H_2 - H_{V_2}, 0) - max(H_3 - H_{V_2}, 0)|} \\
Q_{p_1} &= K_1 \times p_1, \qquad Q_{p_2} = K_2 \times p_2
\end{aligned}
\tag{4.20}
$$

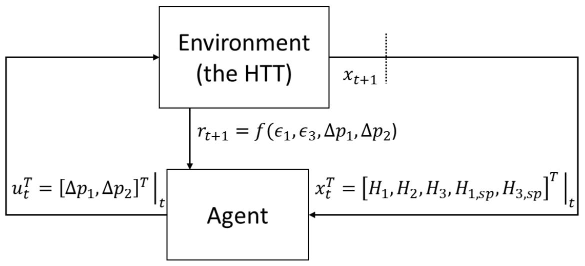

Environment
(the HTT)
x_{t+1}
$r_{t+1} = f(\epsilon_1, \epsilon_3, \Delta p_1, \Delta p_2)$
$u_t^T = [\Delta p_1, \Delta p_2]^T \big|_t$
Agent
$x_t^T = [H_1, H_2, H_3, H_{1,sp}, H_{3,sp}]^T \big|_t$

Figure 4.5: RL infrastructure. The agent observes the water levels, changes the pump speeds and receives the rewards as a function of the performance metrics.

where pump coefficients $K_1 = 1.470$ and $K_2 = 1.435$ were calculated experimentally to convert percentage pump speeds p_1 and p_2 for pumps 1 and 2 respectively. These percentage pump speeds correspond to the actions taken by the agent.

Definition of the state, action, and reward: The **state** vector for the non-constrained control is defined as:

$$x_t^T = [H_1, H_2, H_3, H_{1,sp}, H_{3,sp}]^T \tag{4.21}$$

For the constrained control, to capture the system dynamics effectively, this vector is augmented as:

$$x_{aug,t}^T = [H_1, H_2, H_3, H_{1,sp}, H_{3,sp}, p_1, p_2]^T \tag{4.22}$$

where H_i ($i \in \{1, 2, 3\}$) are the water levels in the tanks as described in the nomenclature. $H_{j,sp}$ ($j \in \{1, 3\}$) are the set points in tanks 1 and 3 respectively.

The **actions** could be defined either in position form (*i.e.*, the output of the agent is the value of the manipulated variable) or in velocity form (*i.e.*, the output of the agent is the change in the value of the manipulated variable). However, position form may cause chattering, oscillations, and aggressive control, especially if the process drifts. The velocity form, on the other hand, provides a smoother transition between the states during learning and is more robust to the drifts. Thus it is used in this work. The actor network outputs two normal distributions, and the actions are sampled from them $\big($*i.e.*, $\Delta \mathrm{p}_i \sim \mathcal{N}(\mu_i, \sigma_i)$ for i = 1, 2$\big)$. The sequential decision making process is summarized in Figure 4.5.

The immediate **reward**, at time step t, used in equation (4.7) is provided in equation (4.23).

$$R_t = -\lambda\left(\epsilon_1+\epsilon_3\right) - \eta\left(\left\|\Delta p_1\right\| + \left\|\Delta p_2\right\|\right) \tag{4.23}$$

where $\epsilon_{1,3} = \left\{\left\|H_1 - H_{1,sp}\right\|, \left\|H_3 - H_{3,sp}\right\|\right\}$ are the deviations of the levels from their corresponding set points for tanks 1 and 3 respectively. Δp_i represents the change in the pump speed for $i = 1, 2$. $\lambda = 0.05$ and $\eta = 0.25$ are the weighting factors that are empirically selected to minimize the MSE during learning. This reward function is similar to the quadratic cost function (QCF), which is widely used in classical control, as shown in equation (4.24) ([34]).

$$\begin{aligned} QCF = &\left(x_t - x_{t,sp}\right)^T \lambda \left(x_t - x_{t,sp}\right) \\ &+ \left(u_t - u_{ref}\right)^T \eta \left(u_t - u_{ref}\right) \end{aligned} \tag{4.24}$$

where $x_{t,sp}$ is the set point, and u_{ref} is a reference for the input. Equations (4.23) and (4.24) minimize the deviation from the set point and the control effort. The similarity between these functions reemphasizes the connection between RL and the control theory.

Identifying System Parameters: In order to obtain a high-fidelity model of the process represented in equations (4.18), (4.19), (4.20), the drain coefficients are estimated (C_{V_i}) using a metaheuristic bat algorithm with the procedure as described in the literature ([257, 258]) using real process data. The optimal coefficients are estimated using a zero derivative scheme to obtain the L1 norm of the prediction error for all three levels. This is shown in equation (4.25).

$$C_{V_i} = \underset{C_{V_i}}{\arg\min}\left(1 - \frac{\left\|\sum_{k=1}^{N} \Psi_k^T - \widetilde{\Psi}_k^T\left(C_{V_i}\right)\right\|_1}{\overline{\Psi}_k^T}\right) \tag{4.25}$$

where $N = 1441$ is the number of samples, $\Psi_k^T = \left[H_1, H_2, H_3\right]_k^T$, $\widetilde{\Psi}_k^T = \left[\widetilde{H}_1, \widetilde{H}_2, \widetilde{H}_3\right]_k^T$, $\overline{\Psi}_k^T = \left[\overline{H}_1, \overline{H}_2, \overline{H}_3\right]_k^T$ represent experimental data, simulation data, and the mean of the experimental data at the k-th time step respectively. The simulation data refers to data calculated from simulation of the process through equations (4.18), (4.19) and (4.20). The model has been validated against process measurements as indicated in Figure 4.6.

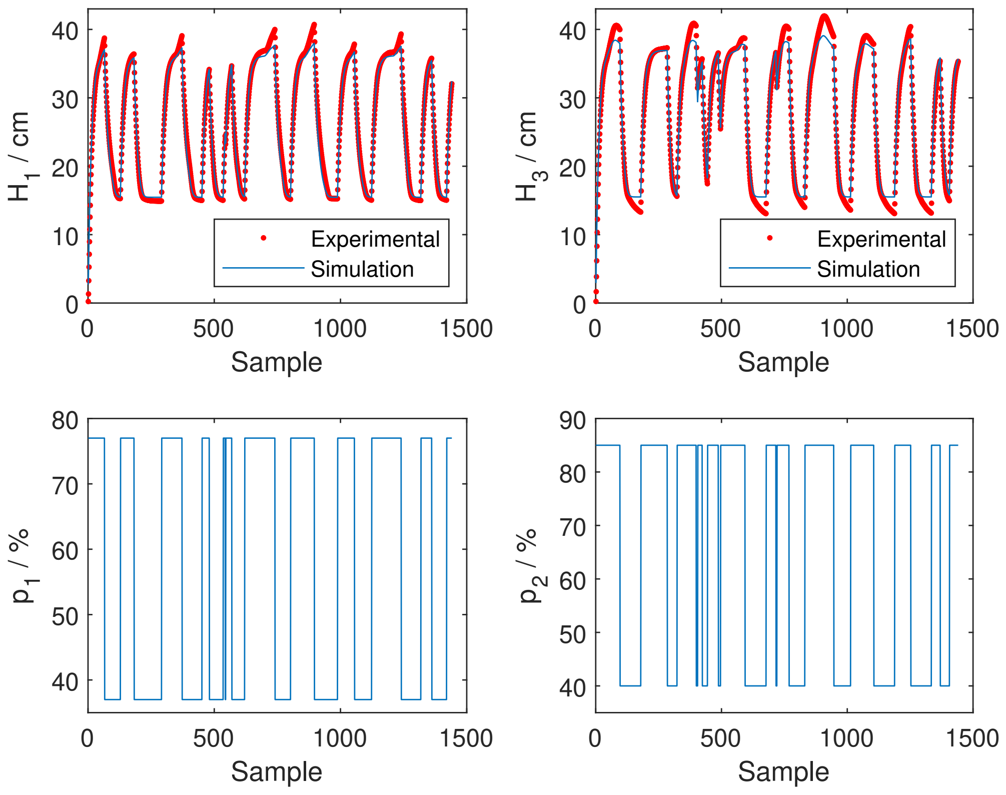


Figure 4.6: First principle model validation.

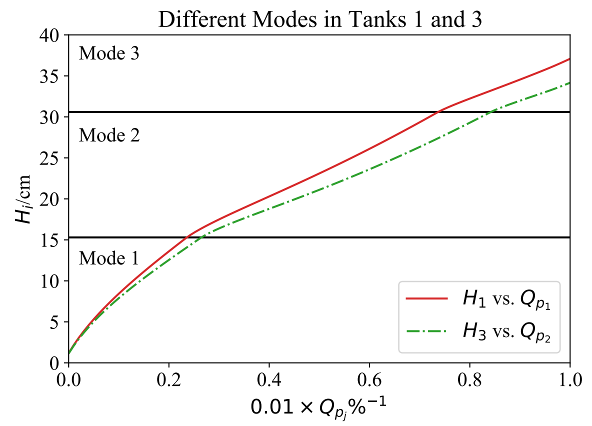


Figure 4.7: The change in the liquid level (H_i) and the pump speed (Q_{P_i}) relationship (the slopes) reflects the multi-modality of the HTT system.

Multi-Modality and nonlinearity of the System: Valves V1-V4 (shown in Figure 4.4) connect the tanks and change the operation mode based on the liquid level. Consequently, there are three major modes in tanks 1 and 3 as shown in Figure 4.7 with the H_i and Q_{p_i} relationship. Controlling a process with such behaviour may require an adaptive controller, which introduces further design challenges. Alternatively, RL can generalize the solution to be robust to such variations once trained.

Similarly, as shown in equations (4.18)–(4.20) the system has some nonlinearities, which may require an additional linearization step while using a classical control methodology. However, the model-free nature of the RL agent inherently addresses such nonlinearities owing to the generalization capabilities of neural networks and the actor-critic algorithm ([232]).

4.5 Experimental Validation

This section discusses the implementation details for the discussed schemes, illustrates the common learning and control metrics that are used in both simulated and experimental environments, and represents the EoE by using heatmaps of the state & action spaces. Moreover, a robustness test is performed to showcase the potential of an RL agent in coping with uncertainties.

4.5.1 Implementation Details

Even though the information given in Sections 4.3 & 4.4 is general, the performance of the discussed algorithms may vary from one problem to another, depending also on the implementation steps (such as the hardware/software that is being used). This subsection explains the implementation in detail for the sake of clarity.

4.5.1.1 Hardware and software

The training program was developed by using Python 3.7, and Tensorflow 1.14.0 was used to instantiate and train the neural networks. The training was conducted on a Lambda deep learning workstation with an Intel Core i9-9820X CPU (10 CPU cores) at 3.30 GHz, 64 GB DDR4 RAM at 2666 MHz under 64-bit Windows 10 operating system.

4.5.1.2 Experimental setup

Real-time communication between the RL agent and the HTT was established using a socket connection to an Opto22 server ([253]). This server acted as the data acquisition system through which the process measurements were transmitted. A block diagram representation of the experimental setup is shown in Figure 4.8.

4.5.1.3 Agent networks

The agent consists of an actor and a critic neural networks. The neural network architecture for the actor is shown in Figure 4.9. The actor constitutes of an input layer which is fed by a normalized input state vector, a dense hidden layer with 200 neurons, and an output layer from which the normalized increment of the pump flow rate distribution (the action) is obtained. A sample from this distribution is the change in pump speed. This value is converted into flow rate and is added to the current flow rate.

Similarly, the critic comprises of an input layer, a hidden layer with 100 neurons, and an output layer. The critic estimates a scalar $V(x_m|\omega)$ from its normalized input.

A sample time of 5 seconds is utilized. Based on the update rules provided in equations (4.8) and (4.9), the critic is updated first followed by the actor. This is

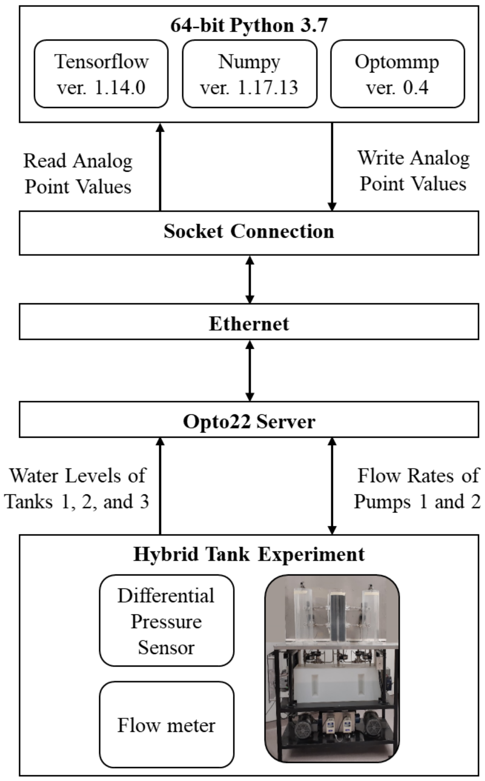


Figure 4.8: Experimental setup. The connection between the HTT system and the algorithm was established by using an Opto22 Server.

because the critic receives measurements from the environment and defines a goal for the actor. Then the actor improves its behaviour according to this goal without diverging. Different strategies, such as the *shared parameters* ([259]), can be used for other control tasks.

Both networks are updated using the RMSProp ([260]) optimizer. The hyper-parameters for the two networks are provided in Table 4.2.

As indicated previously in Section 4.3, three different exploration schemes are utilized to train an agent from a random initialization point, namely SSPT, MSPT, and PA. Each agent encompasses the asynchronous infrastructure of four local networks and a global network comprise this agent as shown in Figure 4.1. The actor-critic network pairs in each worker are initialized from a random position in their parameter space. Training constitutes of episodes, each of which constitutes of a five hour closed-loop simulation based on the first principle model of the HTT (shown in equations (4.18)–(4.20)). With a control objective of tracking set points in tanks 1 and 3,

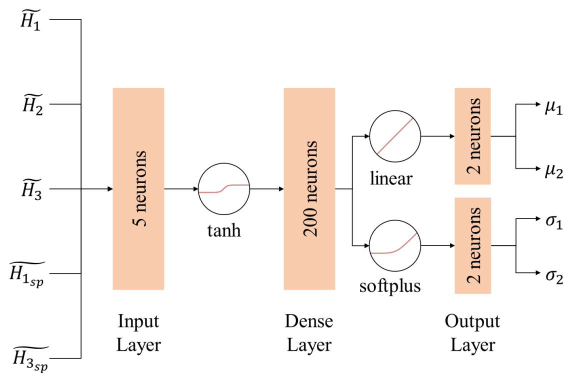


Figure 4.9: Actor architecture.

Table 4.2: Hyper-parameters for learning

Hyper-parameter	**Title**	**Value**
γ	Discount factor	0.95
α_a	Actor learning rate	1×10^{-4}
α_c	Critic learning rate	1×10^{-3}
α_{co}	Constraint learning rate	1×10^{-6}
β	Entropy factor	0.011
$\max\lvert\nabla\rvert$	Maximum gradient norm	10
k	Maximum samples in buffer	100
$\max\lvert\Delta p\rvert$	Maximum action bound	5%
$\min\sigma$	Minimum standard deviation	0.0001
$\max\sigma$	Maximum standard deviation	0.1

the agent is deployed. The first scheme of exploration constitutes of the utilization of the SSPT scheme. During each episode, the tracking scheme assigns the same varying set points to all workers, every 10 minutes. This is repeated for the MSPT scheme, the second scheme, except that unique set point trajectories are employed for each worker and are updated using similar time intervals. The third scheme (PA) trains all workers gradually, one-by-one. Here, the policy is randomly initialized and trained by using a single set point (Phase 1). The resulting policy is subject to further learning by using an SSPT scheme (Phase 2) and then an MSPT scheme (Phase 3).

4.5.1.4 Constrained Learning

Apart from the SSPT, MSPT, and PA settings that did not constitute of any constraints, a new agent with the augmented state vector and reward function is trained. The network parameters are randomly initialized before training the agent. This agent utilizes a dynamic Lagrangian multiplier, Λ, as described in Section 4.3.2. This multiplier adjusts the reward signal with respect to the degree of constraint violation, and provides safer learning, which is of concern in real applications.

4.5.2 Simulation Results

The episodic rewards for the PA scheme are shown in Figure 4.10. The initial episodic rewards in each case depict the gradual learning that the policy has achieved based on each phase of the learning employed. As seen in Figure 4.10, the cumulative reward for Phase 1 of the PA agent (SSP) converged to a larger (better) value than any other training phase. This is because the given set point remains constant throughout the duration of the episode, which makes the control problem less challenging. Each subsequent phase has a larger initial episodic reward.

A broader comparison of all proposed schemes using episodic rewards is depicted in Figure 4.11.

It can be observed that the PA scheme is immediately effective and reaches 90% of its maximum performance quicker than the other schemes. Though it indicates that this scheme is more sample efficient than the other schemes, learning quickly may reduce the generalization ability, hence affect its control performance.

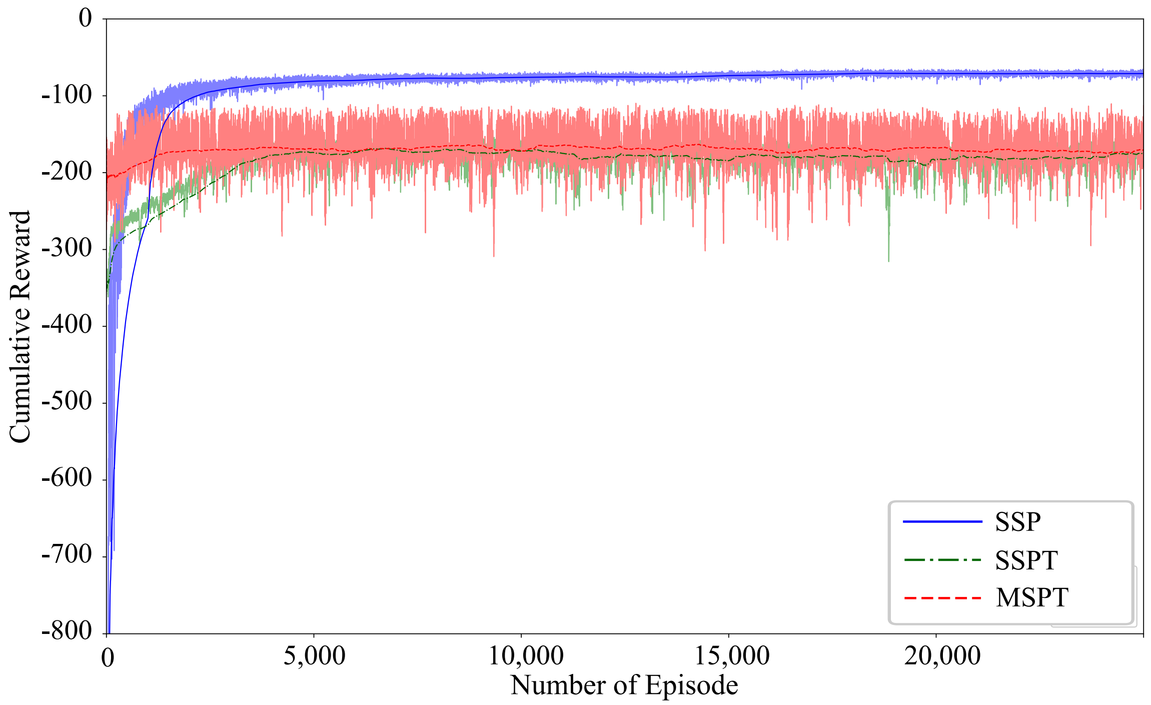


Figure 4.10: PA scheme episodic cumulative rewards.

Figure 4.12 provides a spatial heat map of the state/action space for the three schemes. The space coverage of the SSPT scheme is limited because the control task here involves a single set point trajectory. Although the MSPT and the PA schemes have similar heatmaps, the MSPT scheme explores more than the other schemes. Note that the grid lines appear in the state space due to the interaction between the tanks. The EoE of each scheme is provided in Table 4.3. The table shows that the MSPT scheme yielded the highest exploration value despite the PA scheme also involved an MSPT training phase. This is because the network parameters in the MSPT scheme were initialized randomly, whereas each consecutive phase of the PA training constituted of the network parameters obtained at the end of the previous phase, which limited the exploration ability of the agent for this problem.

Amongst the schemes proposed, the MSPT from random initialization point shows the highest EoE as the result of an increased entropy in the policy. This is seen in Figure 4.12-b and c.

Another interesting observation in Figures 4.12-d, e and f is the appearance of grid line type heat maps. These distinct features represent the modal variation in

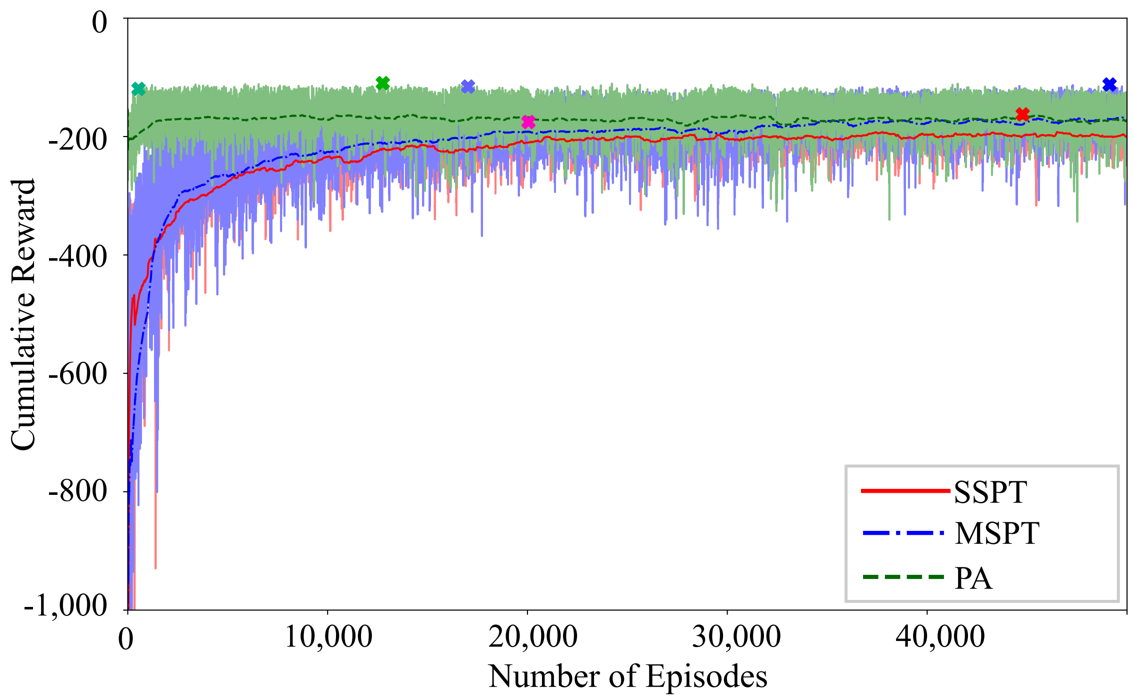


Figure 4.11: Episodic cumulative rewards - three schemes. Each scheme is associated with two X marks, the first mark represents **90% of the maximum rewards** and second mark **the maximum rewards** throughout the training respectively. The shaded areas represent the actual values, and the lines represent the moving average of the actual values. The proximal agent converges to 90% of its maximum performance quicker than the other schemes.

Table 4.3: EoE of different schemes. Though the PA scheme reaches its maximum performance quicker than the other schemes, the MSPT scheme explores the most.

Training Session	**Action Space**	**State Space**
SSPT	0.19944	0.25670
MSPT	**0.38709**	**0.50827**
PA Phase 1	0.08418	0.08030
PA Phase 2	0.08686	0.11773
PA Phase 3	0.26038	0.38118

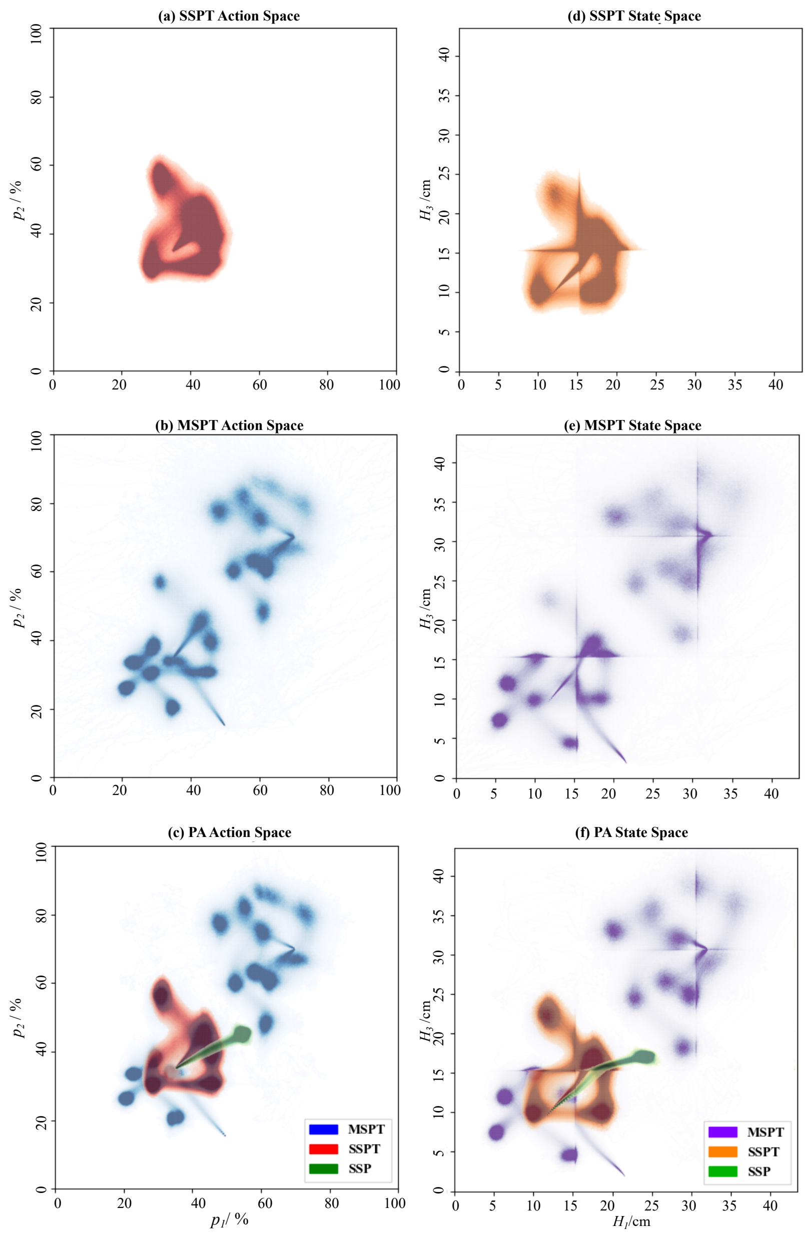


Figure 4.12: Heat maps obtained from different training schemes. The MSPT scheme results in the highest exploration (b), (e) as indicated by the EoE in Table 4.3. Grid lines that appear in the state space (*i.e.*, d, e, and f) are a result of the multi-modal structure of the system, owing to the interaction between the tanks.

Table 4.4: Performance metrics from validation - Simulation

		SSPT	**MSPT**	**PA**
MAE / cm	H_1	1.571	0.697	0.686
	H_3	4.137	0.744	1.204
MSE / cm^2	H_1	4.450	2.125	1.958
	H_3	24.205	2.299	3.167
Variance	p_1	0.385	0.404	0.275
	p_2	0.746	0.328	0.377

the process, owing to the interaction the tanks 1 and 3 have with the tank 2. Exploration of these regions to improve the policy further emancipates the generalization of the policy. This aspect fortifies such schemes for multi-modal processes, which are prominent in chemical process systems.

Based on the three training schemes, three policies were obtained. To validate these policies, a servo tracking for a set point trajectory that is distinct from training is utilized. Based on repetitive simulations, the average performance metrics (MSE, mean absolute error (MAE), and variance in manipulation) were obtained and provided in Table 4.4.

Table 4.4 shows that average MAE and MSE values for MSPT scheme are lower than that of PA scheme. Although training an agent using three phases that improve successively may lead to a faster convergence as shown in Figure 4.11, this scheme does not necessarily improve the control performance. In fact, for this problem, training an agent using MSPT scheme provided a better control in terms of the error metrics as shown in Table 4.4 and in Figure 4.13. The SSPT scheme deviated from the set points significantly in both tanks. Although the PA scheme converged to its maximum performance quicker than the other schemes, it could not remove the offset completely. Moreover, MSPT showed the highest variation in the pump speed to minimize the deviation from the set points. These are because the SSPT and the PA schemes did not explore the state/action spaces as much as the MSPT scheme did. EoE values of these schemes are shown in Table 4.3.

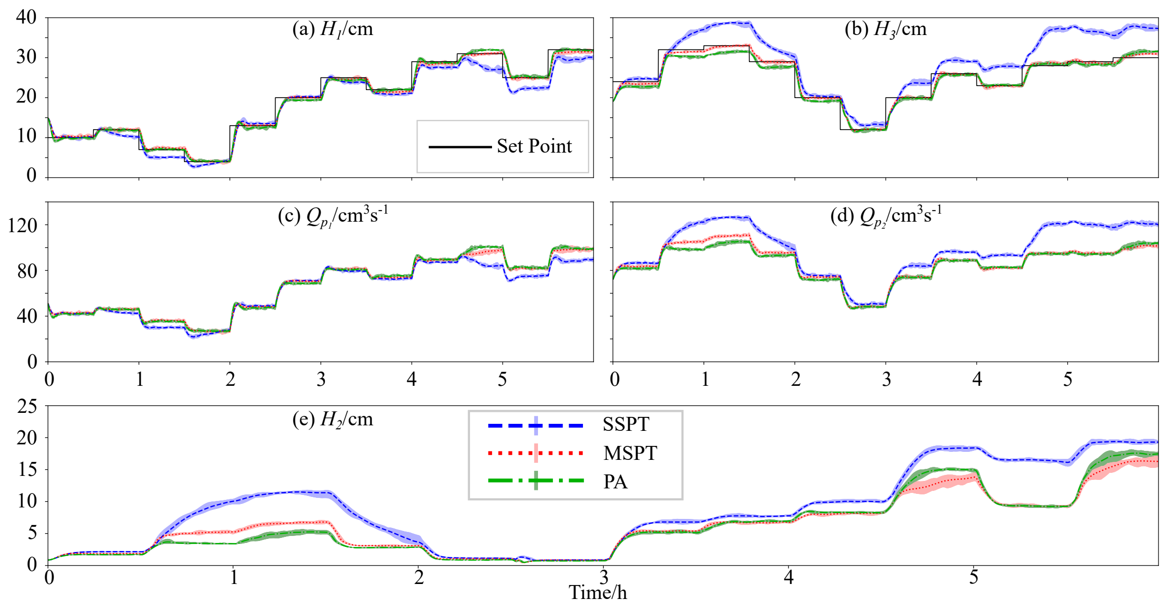


Figure 4.13: Validation of final policies from the three training schemes - Simulation. Though the PA scheme converges to 90% of its maximum performance quicker than the other schemes (as shown in Figure 4.11, it yields offsets during control. The SSPT-based agent tends to diverge from the set points in both tanks. On the other hand, MSPT agent shows the best controlling performance in terms of the deviation from the set point.

4.5.3 Robustness Test - Simulated

Though the agent is able to control the water levels in the tanks, it is important to observe the robustness of the resulting policy to the process uncertainties after learning. The tested policy is obtained from the MSPT scheme, because it yielded the highest EoE. It is tested by inducing a ±10% variation to the pump coefficients (*i.e.*, K_1 and K_2). The agent has been exposed to this variation through the test but not during learning. The visualization of servo tracking under such uncertainties is provided in Figure 4.14. The performance metrics are provided in Table 4.5.

Overall, the learned MSPT policy was able to control the levels without significant errors, oscillations or overshoots. These show that even though the agent did not experience such changes during the training, its policy was able to adapt to the perturbation in the action space.

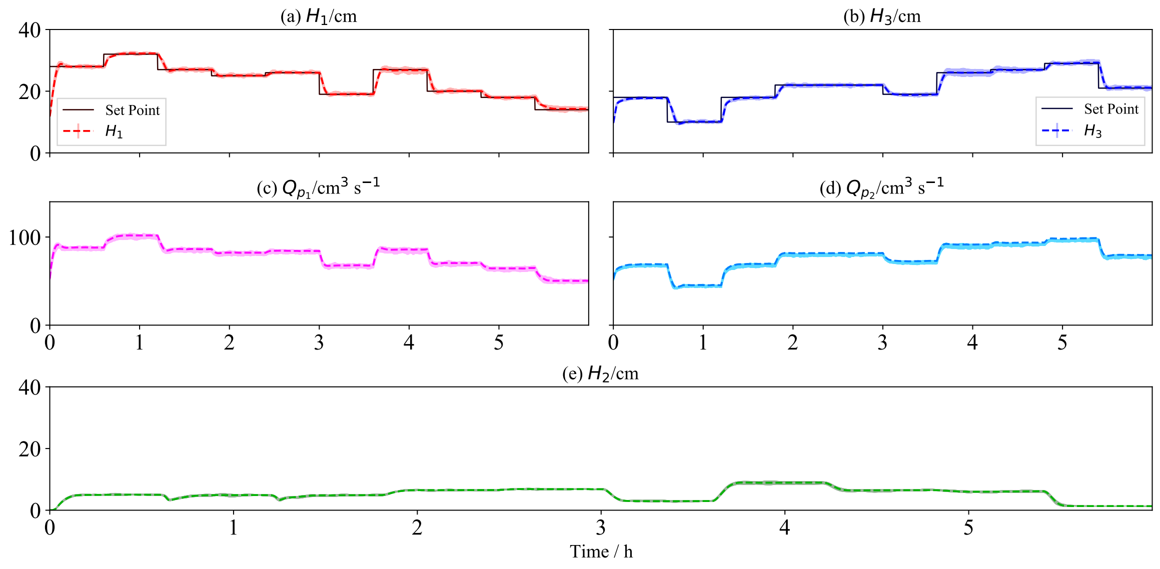


Figure 4.14: Robustness test using the MSPT policy - Simulation. ±10% variation is induced to the pump coefficients at the beginning of the test. The MSPT agent successfully adapted to these changes without any significant deviations.

Table 4.5: Performance metrics from robustness test of MSPT policy- Simulation. The agent was able to control the pumps without any significant errors.

		Ideal	p_1 **+10%**	p_1 **-10%**	p_2 **+10%**	p_2 **-10%**
MAE/cm	H_1	0.558	0.582	0.637	0.608	0.573
	H_3	0.570	0.613	0.636	0.516	0.567
MSE/cm^2	H_1	2.362	2.263	2.559	2.463	2.352
	H_3	1.805	1.827	1.870	1.596	1.918
Variance	p_1	0.0518	0.0627	0.0545	0.0603	0.0571
	p_2	0.0542	0.0594	0.0502	0.0498	0.0475

4.5.4 Real-time Experimental Results

Owing to its performance, the policy obtained from the simulation using MSPT scheme is utilized for the online learning. Since only one instance of the HTT real-time experiment is available in reality, asynchronous learning is limited to learning through simulation. Its single agent variant, the A2C algorithm is utilized for online learning. It consists of one global policy and only one worker (constituting of a policy and the HTT experiment for the environment). The experimental setup is depicted in Figure 4.8.

During online learning, the behavioral policy is randomly initialized and generates losses based on its interaction with the HTT experiment. To ensure safe operation

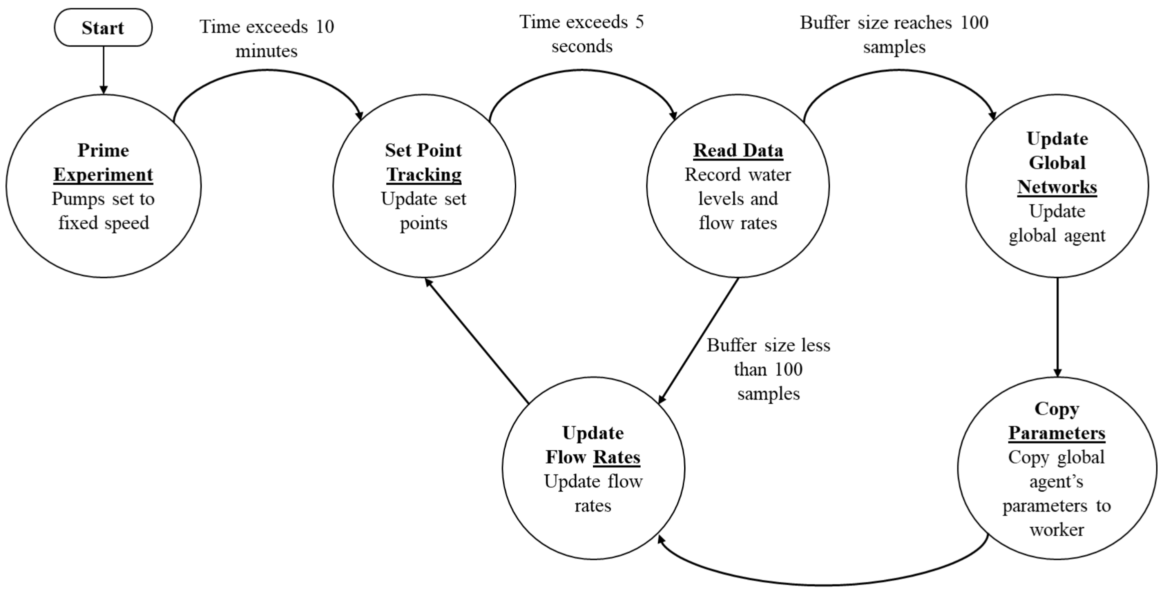


Figure 4.15: Finite state machine design of real-time online learning application in Python©

Table 4.6: Results before/after online A2C learning.

		Before	**After**
MAE/cm	Tank 1	0.632	0.506
	Tank 3	0.746	0.641
Variance	Pump 1	1.085	1.109
	Pump 2	1.079	1.103

and avert communication failures or faults, a Mealy-Moore finite state machine (FSM, [261]) based application is designed. This is shown in Figure 4.15. Here, each circle indicates a state of operation while the arrows indicate the conditions for transition from one state to another. Also, the tasks carried out during transition or within the state are depicted.

During online training, the episodic training is again carried out. Multiple sequential episodes constituting of two hours each are carried out on the experimental setup with multiple inter-episodic updates of the policy using rollout. The servo tracking results from the entire 32 hours of online learning are shown in Figure 4.16, and the related metrics are provided in Table 4.6.

It can be observed that during the online learning, the agent continues to be reliable and does not produce abnormal actions which could otherwise throw the

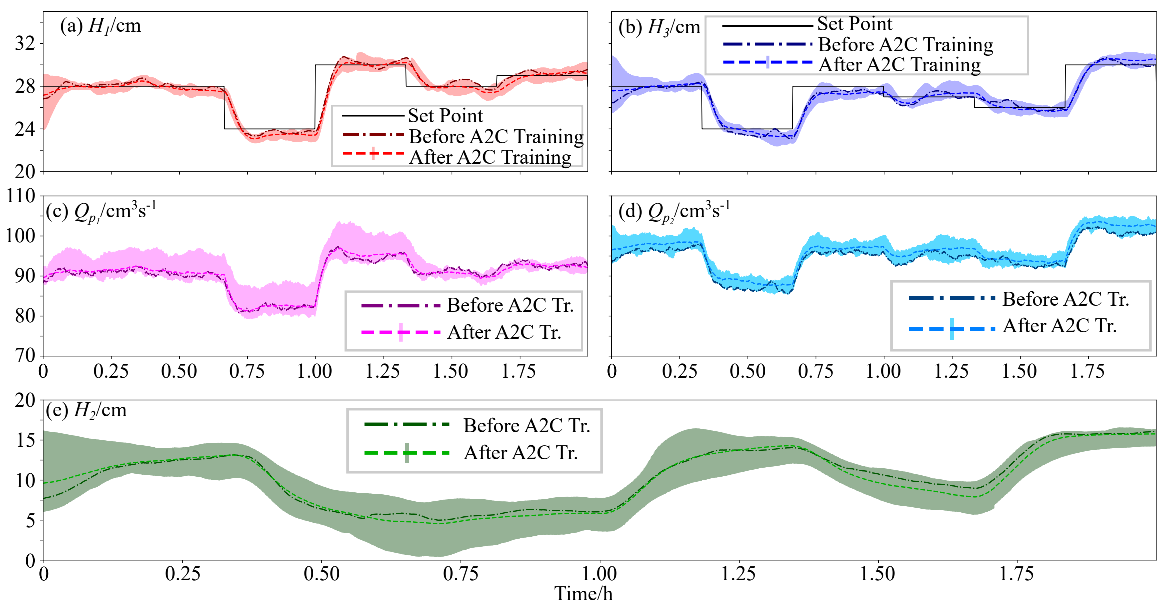


Figure 4.16: Online learning using A2C scheme on experimental setup.

closed loop process into instability. The agent also reduces the average tracking error for both tanks by taking more aggressive actions. This is because the agent adapts to the behaviour of the real system, which involves measurement uncertainties unlike the deterministic system model.

4.5.5 Robustness Test - Experimental

In addition to the simulated robustness test shown in Section 4.5.3, unmeasured disturbances are introduced at different time steps by opening valves V6 and V8 (shown in Fig. 4.4) to demonstrate the effectiveness of the agent. Additional Gaussian noise, $\mathcal{N}(0, 0.04)$, is added to the sensory measurement in order to enhance the uncertainties and make the control problem more challenging. As shown in Figure 4.17, the agent is robust to such environmental challenges.

4.5.6 Soft-constrained Learning

The previous result shows in Figure 4.13 that the agent can control the water levels in a MIMO system. However, during learning in those schemes the agent may try to maximize the reward without considering physical limits such as the maximum height in the tank, and operational limits on the pumps' speed. Introducing hard-state-

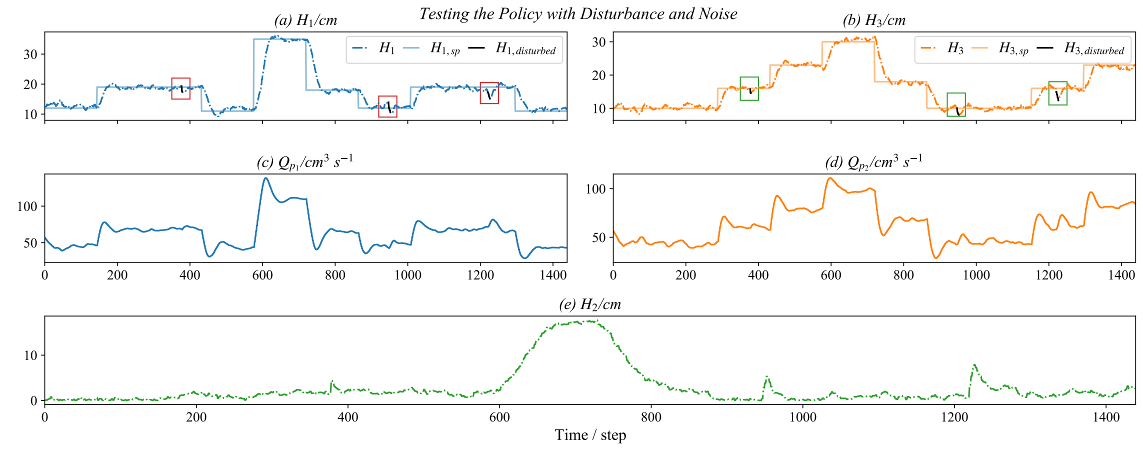


Figure 4.17: Disturbance and noise test using the MSPT policy - Experimental. The disturbances are introduced by simultaneously opening valves V6 and V8 for 15, 30 and 35 seconds respectively. Additional noise, $\mathcal{N}(0, 0.04)$, is added to enhance the sensory uncertainties throughout the experiment. The results show that the agent is robust to such environmental conditions.

constraints may prevent the agent from learning or experiencing the consequences of visiting the states beyond these constraints. Soft-constraints, on the other hand, sacrifice this 'never-visit' condition while preserving feasibility and letting the agent learn these states. Moreover, the dynamic weight factor (*i.e.*, Λ in equation (4.11)), makes the agent be aware of how far it is from the limits.

Figure 4.18 shows that the set points are tracked in a stable manner by the soft constrained scheme. Also, the statistical results over ten simulations indicate that the agent, on average, violates the constraints 25 times in the first episode of the training phase. Meanwhile, it avoids taking actions that lead to unsafe states after training, with no violations during validation. These show that the agent learns not to violate the constraints while maintaining a satisfactory control performance. This scheme allows the introduction of constraints on any observable state. Such an adaptation would promote constrained real-time optimization based on state-specific operational regions using a model-free approach. Hence, this scheme becomes an important augmentation to obtain a model-free constrained policy.

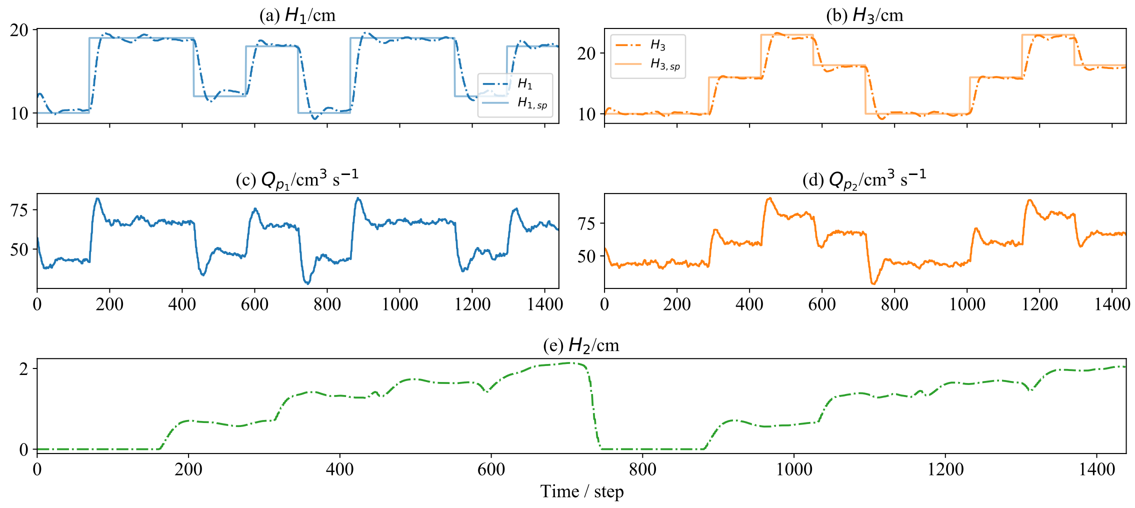


Figure 4.18: Constrained scheme - test.

4.6 Discussion

Alternative to the A3C scheme, various model-free algorithms such as deep deterministic policy gradient (DDPG, [74]), soft actor-critic (SAC, [72]), twin delayed DDPG (TD3, [76]) can be used to solve the same control problem. These algorithms differ in terms of their action selection and policy update rules.

For example, DDPG can be used to learn external policies (*e.g.*, a policy defined by a human expert), which may provide safer control initially. SAC can be used to enhance the exploration because it maximizes the entropy of the policy similar to the A3C algorithm. TD3, on the other hand, uses two neural networks for policy evaluation (value estimation), and uses the worst-case value estimate during learning. This idea takes its roots from the *worst-case performance under uncertainty* methodologies in the robust control literature ([262, 263]). TD3 also keeps track of the network parameters to ensure learning stability.

A3C, however, achieves similar control performance compared to the above-mentioned algorithms with fewer hyper-parameters, promotes exploration and allows the user to parallelize the simulation environments to reduce the training duration.

In addition to this low-level control application, the scheme can be utilized in higher levels of the control hierarchy ([264]) by adjusting the interaction frequency of the agent. This may increase the efficiency of plant-wide optimization strategies while preserving safety and process stability.

4.7 Conclusion

Since approximation-based reinforcement learning requires interactions starting with a random control policy, its direct real-time implementation on processes may be unsafe because of taking possibly detrimental initial actions. Furthermore, thorough state/action space exploration is necessary to obtain a truly generalized control policy. This becomes more important while dealing with nonlinear/multi-modal chemical processes. This chapter proposes use of three different progressive reinforcement learning schemes to learn the control policy using a first principles model of a hybrid three-tank system. The control problem is then reformulated into a constrained scheme to obtain a safer control policy. It presents the control related results and also studies the extent of exploration (EoE) as a measure of the state/action space exploration illustrated in the form of heatmaps. The best performing scheme is successfully implemented online on the real-time hybrid three-tank system through Opto-22 socket connection. The results obtained demonstrate the efficacy of pre-training using simulation in finding an initial stable policy for online, real-time learning, paving way to the possibility of implementation on industrial processes parsimoniously.

Chapter 5

Reinforcement Learning with Constrained Uncertain Reward Function Through Particle Filtering [*]

This chapter presents the constrained filtering approach considering the reward/cost properties in process control. Advancements in computational sciences have stimulated the use of an abundant amount of data in control and monitoring. Recent studies have reemphasized that the performance of the data-driven control significantly depends on the data quality. This quality is affected by uncertainties such as process and measurement noises. This study addresses a type of noise commonly seen in industry, and shows how it degrades the performance of a deep reinforcement learning (RL) agent. Then, a novel filter is proposed to reduce the effect of this noise when it causes skewed probabilistic distributions in the reward functions. We demonstrate that the RL policy can be improved by using a constrained filter with a combination of the optimal filtering and RL concepts. The proposed algorithm is applied to a pilot-scale separation process that resembles an industrial separation vessel. The experimental results demonstrate the proposed algorithm can improve the process operation efficiency.

[*]This chapter has been published as **O. Dogru**, R. Chiplunkar and B. Huang, "Reinforcement Learning With Constrained Uncertain Reward Function Through Particle Filtering," in IEEE Transactions on Industrial Electronics, vol. 69, no. 7, pp. 7491-7499, July 2022, doi: 10.1109/TIE.2021.3099234.

5.1 Introduction

Rapid developments in technology and data availability in different domains have caused a paradigm shift in research as well as in the process industry. Complex control problems today are often solved by black-box methods that are data-driven. This is because the state-of-the-art techniques have made it possible to process high-dimensional data in real-time. Despite the practicality of these techniques, this data-driven era has its challenge: data reliability.

Similar to the classical control techniques [34], the learning-based methods aim to find the best set of parameters, which can describe the data accurately. These parameters can be used either to reconstruct data under different operating conditions [144] or to map an input data to its label [142]. In system identification and control [29, 172], such supervised and unsupervised learning techniques are used to develop time-series models, fault detection monitors and control [265, 266]. Reinforcement learning (RL), on the other hand, formulates these control/monitoring tasks as a sequential decision-making process and builds a model adaptively [35, 37]. Here, an *agent* tries to improve a performance metric by interacting with an *environment* as shown in Fig. 5.1-a. In addition to their impressive performance in playing games [39, 73], RL algorithms have also shown remarkable progresses in the control field [40, 119, 126, 187, 267–273].

Independent of the type of methods, the learning may be considered solving an optimization problem with an associated *reward* function. The reward can be binary, multinomial, continuous variable, or a combination of them [274]. For complex problems, it may be challenging to design a proper reward, especially in the presence of uncertainties. These uncertainties may be due to physical conditions that affect the sensory readings, human error, or adversarial attacks [275, 276]. Such uncertainties have been reported to degrade the model/controller performance [275], and need to be taken care of rigorously.

This study considers a type of uncertainty, which is caused by the sensors that are utilized in the reward function. When the noise is Gaussian and the system is linear, the Kalman filter (KF) provides the optimal estimate for the target variable

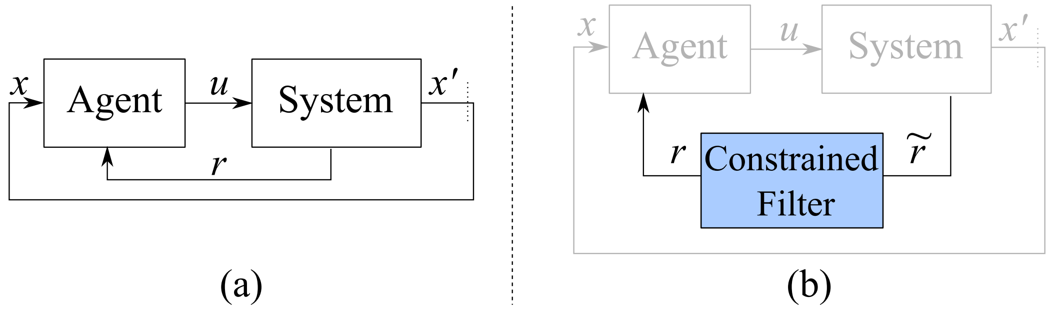


Figure 5.1: **(a)** Schematic of the generic RL algorithms. The agent observes a state and takes an action that results in the next state and the reward. **(b)** The proposed filtering scheme. The system emits a noisy reward, $\tilde{r}$, instead of the noise-free r. The constrained filter removes the noise by considering the reward boundaries and outputs an estimate of the true reward.

[31, 277]. However, non-Gaussian and nonlinear problems require more advanced solutions such as the nonlinear and constrained state estimation methods [278–281]. Some of these methods, (*e.g.*, the PDF truncation mentioned in [279]) result in biased estimates. Besides these methods, several studies have considered the robustness in RL by developing the system model or augmenting the reward to account for uncertainties [282]. However, these methods are either too general or do not take the constraints on the observation probability distributions into account. Ignoring this important information can deteriorate the performance and robustness of the estimation.

This study, on the other hand, utilizes a novel constrained particle filtering [283–286] scheme, targeting a type of reward that is commonly used in control applications. This is necessary in order to achieve an unbiased policy, improve sample efficiency, and prevent numerical issues while updating the parameters (*e.g.*, when using the stochastic gradient descent). The proposed scheme formulates the reward as a dynamic variable and considers constraints on both its transition probability and its observations to achieve certain properties of the reward functions such as non-negativity etc. In addition to reducing the variance of the estimation, this method maintains robustness. The contributions of this chapter are as follows:

- formulating the reward value as a dynamic process with its transition and observation probabilities,

- estimating the constrained reward by using a state and observation-constrained filter,
- demonstrating the importance of the proposed method by using an experimental case study,
- showcasing the robustness of the proposed method in the presence of noise and outliers,
- applying the constrained filter in an asynchronous scheme by using multiple workers [73],
- quantitatively and qualitatively analyzing the proposed scheme by considering the filtering, learning and tracking aspects in an RL setting through a pilot-scale experiment.

The remainder of this chapter is organized as follows: A detailed background information is provided in Section 5.2, the proposed method is described in Section 5.3, the results are discussed in Section 5.4, and the concluding remarks are presented in Section 5.5.

5.2 Background

An RL agent learns an optimal policy by interacting with the environment and receiving a reward signal [34, 35]. This signal is obtained by sensors and may contain noise inherently. This study proposes a method to obtain robust policies for detecting an interface when the reward uncertainty has a skewed distribution. In this section, the fundamentals of RL, actor-critic RL, optimal control, and state (reward) estimation are presented.

5.2.1 Markov Decision Process (MDP)

Generally, a control problem can be represented as an MDP that is denoted as $\mathcal{M} = \langle \mathcal{X}, \mathcal{U}, \mathcal{R}, p, \gamma \rangle$, where $\langle \cdot \rangle$ denotes a tuple. At a discrete time, t, the agent observes a *state*, $x_t \in \mathcal{X}$, and takes an *action* $u_t \in \mathcal{U}$. The system, whose model is represented as $p(x', r|x, u)$, moves the agent to the next state while emitting a *reward* signal $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$. The model-free algorithms assume that this model is unknown to the agent. The discount factor, $\gamma \in [0, 1]$, is a weight that determines the contribution of the future rewards. From a digital signal processing point of view, it behaves like a low-pass filter. The agent's goal in an MDP is to *learn* how to maximize the discounted return (5.1).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{5.1}$$

where the capital letters indicate that the reward is a stochastic variable.

During its interactions with the environment, the agent samples its actions from a stochastic policy, $\pi(u|x)$, whose performance is measured by using a value function. The type of value function can be selected based on the policy evaluation technique. The learning can be performed by recursively solving the Bellman equations (5.2)-(5.3) [37].

$$\begin{aligned}
v_\pi(x) &= \mathbb{E}_\pi \left[G_t | X_t = x\right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | X_t = x\right], \forall x \in \mathcal{X} \\
&= \sum_u \pi(u|x) \sum_{x'} \sum_r p(x', r|x, u) \\
&\qquad\qquad \times \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | X_{t+1} = x'\right]\right] \\
&= \sum_u \pi(u|x) \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma v_\pi(x')\right]
\end{aligned} \tag{5.2}$$

$$\begin{aligned}
q_\pi(x, u) &= \mathbb{E}_\pi \left[G_t | X_t = x, U_t = u\right], \forall x, u \in \mathcal{X} \times \mathcal{U} \\
&= \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma \sum_{u'} \pi\left(u'|x'\right) q_\pi(x', u')\right]
\end{aligned} \tag{5.3}$$

where $\mathbb{E}[\cdot]$ is the expectation of a random variable. These equations assume the Markov property, where the future values depend only on the current values, and are independent of the past given the current values.

Following the previous step, the optimal value functions can be found by using equations (5.4) and (5.5).

$$\begin{aligned} v^*(x) &= \max_{\pi} v_\pi(x), \forall x \in \mathcal{X} \quad (5.4) \\ q^*(x,u) &= \max_{\pi} q_\pi(x,u), \forall x, u \in \mathcal{X} \times \mathcal{U} \\ &= \mathbb{E}\left[R_{t+1} + \gamma v^*(X_{t+1}) | X_t = x, U_t = u\right] \quad (5.5) \end{aligned}$$

Finally, the optimal (also called the *greedy*) policy can be obtained by using equation (5.6).

$$\pi^*(x) = \arg\max_{u} q^*_\pi(x,u) \quad (5.6)$$

However, the system model, $p(\cdot)$, may not be available, and hence cannot be used in equations (5.2) and (5.3). In addition, finding exact values of $v(x)$ or $q(x,u)$ may not be feasible in large/continuous space control problems. Approximate value functions, $V(x|\omega)$ or $Q(x,u|\omega)$, can replace the exact solutions to overcome these challenges, where ω denotes the parameters of the value functions. Section 5.2.2 explains how to utilize these functions to learn optimal policies.

5.2.2 Actor-Critic Reinforcement Learning

The value functions mentioned in the previous section are also called a *critic*, by which the optimal policy is obtained indirectly [38,39]. Alternatively, a policy can be parameterized as $\pi(U_t|X_t, \theta_t)$, and directly optimized. This technique is called policy gradient, and it optimizes the parameters as shown in equation (5.7).

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(U_t = u_t | X_t = x_t, \theta_t)}{\pi(U_t = u_t | X_t = x_t, \theta_t)} \quad (5.7)$$

where α is the learning rate, and θ represents the parameters of the actor [35].

As an extension of the policy gradient and value-based techniques, actor-critic algorithms address estimation variance and slow learning problems by combining both techniques. Several actor-critic algorithms have been proposed [73, 74, 76] and applied to various control problems [120,125], which will not be detailed in this study. Nevertheless, one common feature of these methods is that they minimize a variant of the TD-error (δ) to improve the critic (equation (5.8)) and the actor (equation (5.9)) networks [76, 129].

$$d\omega_{new} \leftarrow d\omega_{old} + \alpha_c \nabla_\omega \delta(\cdot|\omega)^2 \tag{5.8}$$

$$d\theta_{new} \leftarrow d\theta_{old} + \alpha_a \delta\left(\cdot|\omega\right) \nabla_\theta \ln \pi\left(\cdot, \theta\right) \tag{5.9}$$

$$\text{with } \delta = y - h \tag{5.10}$$

where α_c and α_a represent the learning rates of the critic and the actor, respectively. $d\cdot$ denotes the increment of the parameters. The networks are updated by using the backpropagation [73]. y represents an observed value function, and h corresponds to a predicted value function. For the asynchronous advantage actor-critic (A3C) algorithm, these can be calculated as:

$$y = r_{t+1} + \gamma V(x_{t+1}|\omega') \tag{5.11}$$

$$h = V(x_t|\omega) \tag{5.12}$$

where ω' indicates that the parameters are the value determined in the previous time instant, and ω stands for the parameters to be updated [129]. Note that the only information that comes from the environment is the reward signal, r_{t+1}, and the value function is predicted through the deep networks. Updating the network parameters as such improves the learning stability [76]. Nevertheless, the uncertainties in the reward value will still affect the value functions, TD-error, the policy, and finally, the control performance.

This study investigates a type of practical reward functions and its uncertainties, as will be discussed in the next section.

5.2.3 The Reward Function

The reward in RL is a performance metric that may be either binary (*e.g.*, whether a goal is achieved or not) or continuous (*e.g.*, the mean square error), and can be obtained either sparsely (at the end of an episode) or frequently (at every sampling instant). Practically, it may give information about ”product quality”, ”deviation from an operational point”, ”control effort”, ”how long an agent has survived”, ”whether the agent achieves a goal or not” *etc.*

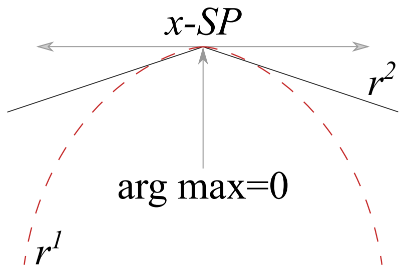


Figure 5.2: Geometric representation of various types of reward. Note that both equations are concave, and have a maximum value of 0 when there is no deviation.

From the optimal control point of view, the reward may be formulated as the quadratic cost function as commonly used in model predictive control (MPC) or linear quadratic regulator (LQR). A simplified version of the quadratic cost is defined as shown in equation (5.13)[†].

$$r_t^1 = -\left(x_t - SP_t\right)^T Q_t \left(x_t - SP_t\right) \tag{5.13}$$

where SP represents the desired/reference/setpoint value, Q is a square weight matrix that determines how much a particular state will contribute to the reward function, and $(\cdot)^T$ represents the transpose operation.

One advantage of using the quadratic form is that the reward function becomes twice differentiable, which is beneficial for Jacobian/Hessian-based optimization techniques. It is also smooth and concave, which helps to find the optimal values. However, a quadratic reward can be sensitive to uncertainties in sensory measurements, particularly uncertainties with large magnitudes. Therefore, using a lower-order reward may be more beneficial in the presence of uncertainties. In this study, the L1-norm (equation (5.14)) is used to make the agent more robust to outliers and noise.

$$r_t^2 = -\left|x_t - SP_t\right|_1 \tag{5.14}$$

Note that both equations (5.13) and (5.14) have a maximum value of 0[‡] theoretically, and are shown geometrically in Fig.t 5.2.

[†]This is the weighted and squared L2-norm. In fact, the quadratic cost function does not have a negative sign in control due to the minimization operation. It can also include an action term. To connect the reward used in RL & the cost used in optimal control, and for the sake of simplicity, here it is shown as a negative function. See [29, 172] for more information.

[‡]That is, when $x = SP$

The noise is inherent and cannot be avoided regardless of what norm to be used. These challenges motivated this work to develop a type of Bayesian filter for estimating the reward in the presence of uncertainties. The theory behind the proposed filtering method is explained in Section 5.2.4.

5.2.4 State (Reward) Estimation

A state in a system completely represents the internal status of the system at any time [31]. In real systems, the true values of the states may not be known due to uncertainties, and some of the states may not be observed due to hardware limitations. These issues make the system/control design challenging in practice. State estimation, in this context, refers to optimally estimating a state by using some uncertain observations. Inspired by the state estimation methodology, the goal of this work is to estimate the *true/hidden reward* given its noisy observations.

Let $p(r_t|r_{t-1})$ and $p(\tilde{r}_t|r_t)$ represent the transition and the observation probability distribution functions (PDFs) of the reward respectively. The objective is to estimate $p(r_t|\widetilde{\mathbf{R}}_\mathbf{t})$, where $\widetilde{\mathbf{R}}_\mathbf{t} = [\tilde{r}_1, \tilde{r}_2, ..., \tilde{r}_t]^T$ is the noisy reward observations up to time, t. The estimation could be performed by using a Kalman filter if $p(r_t|r_{t-1})$ and $p(\tilde{r}_t|r_t)$ are Gaussian PDFs. However, in typical RL tasks, these PDFs are skewed. This property introduces tractability challenges, which require advanced solution methods, as will be discussed in Section 5.3.

5.3 Interface Tracking Under Uncertainty

To give a practical background of this work, separation processes are a key element in the oil sands industry because both the product quality and the environmental impact depend on their efficiency [126, 137]. One of the main separation processes is through a primary separation vessel (PSV), where bitumen is separated from impurities such as water and solids [287]. This physical process involves complex interactions due to operational conditions and feed properties. It is controlled by measuring the interface level (by utilizing a pressure sensor) and manipulating the process flow rate, such that this level is kept at a particular point to optimize the product quality. Therefore, it is

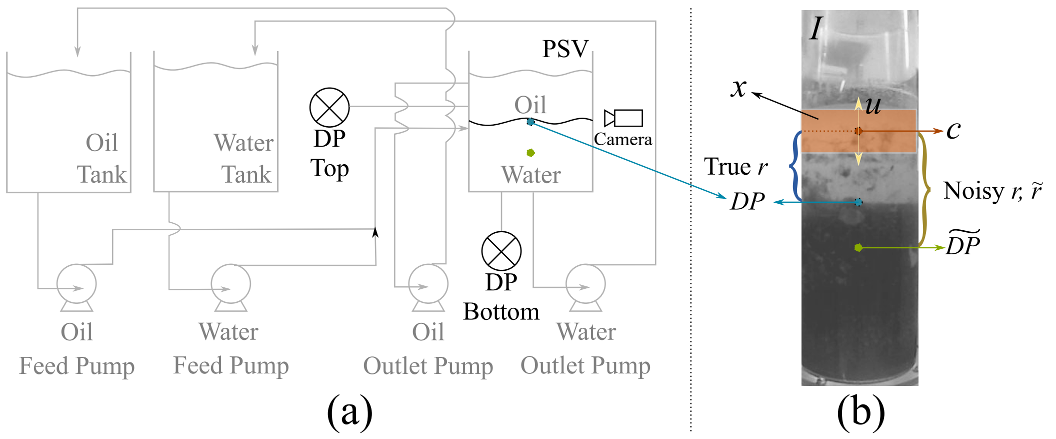


Figure 5.3: **(a)** The pilot-scale PSV setup. A mixture of oil and water is fed into the PSV tank, while the level is monitored by using two differential pressure (DP) sensors as well as a camera. The true level is indicated with the blue dot, whereas its noisy measurement is shown as the green dot below. **(b)** An image, I_t obtained by using the camera. The agent observes the pixels, x_t, inside the orange box. Then, it takes an action, u_t, which either moves the box up/down by 1 pixel or does not move. Following this, it observes x_{t+1} from I_{t+1}, and calculates the reward ($\tilde{r}_{t+1}$ by using $\widetilde{DP}_{t+1}$, instead of DP_{t+1}). The time steps are not shown in the figure for the sake of simplicity.

crucial to monitor the level accurately. The pilot-scale system to mimic this process is shown in Fig. 5.3-a.

The camera shown in Fig. 5.3-a obtains an image, I_t, in which a virtual moving-box (with a center, c_t) is drawn as shown in Fig. 5.3-b. The pixels within this box are cropped and sent to the RL agent as the state, x_t. The agent moves the box to c_{t+1} by sampling an action from its policy, *i.e.*, $u_t \in \{+1, 0, -1\}$px. $\sim \pi(u_t|x_t, \omega)$. The camera obtains the next image, I_{t+1}, from which the next states, x_{t+1}, are cropped. Parallel to this, the DP sensors measure the level and send it to the agent for the reward calculation, as shown in equation (5.15).

$$\tilde{r}_{t+1} = -|\widetilde{DP}_{t+1} - c_{t+1}| \tag{5.15}$$

where c_{t+1} represents the center of the box after the action is taken, and $\widetilde{DP}_{t+1}$ is the corresponding level measurement from the pressure sensors. Note that this reward representation is equivalent to equation (5.14), and the reward is maximum when the center of the box and the DP sensor measurement overlap, *i.e.*, when $\widetilde{DP}_{t+1} = c_{t+1}$. Note that the DP sensors are only used in the training phase as a reference of the

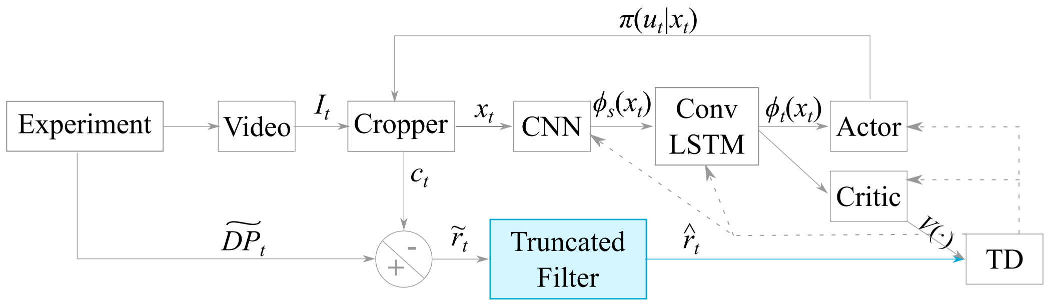


Figure 5.4: A flowchart of the proposed method. The gray boxes indicate a general RL scheme without filtering. The proposed method is highlighted as the blue box. The solid and the dashed lines indicate the data and the gradient flows respectively. CNN and ConvLSTM represent convolutional neural and convolutional long-short term memory networks respectively. The images, I, are obtained from the experiment by using a camera together with their corresponding noisy pressure sensor measurements, $\widetilde{DP}$. The agent observes a cropped image, x_t, moves the cropper box according to its policy. ϕ_s and ϕ_t represent the spatial and temporal features obtained from the neural networks respectively. The noisy reward is calculated according to equation (5.15). Then, the reward is filtered according to equations (5.19) and (5.20). TD-error is calculated by using this filtered reward estimate.

level.

In the ideal case, the level measurements should be noise-free, as shown with a blue dot in Fig. 5.3-b. However, in real applications, there is inherent noise, and the pressure sensors provide a noise-corrupted $\widetilde{DP}_{t+1}$ instead of DP_{t+1}. Common filtering techniques to estimate the true value of a random variable given its noisy measurements include but are not limited to Kalman filtering (and its variants) [31, 288, 289], average filtering [290], *etc.*, which do not consider the constraints and skewness of the distribution function (such as non-positiveness of a quadratic reward) that may exist on both the reward and observations. This study considers the interface tracking problem as an MDP that satisfies the Markov property for the states and the reward. The following random-walk type of Markovian evolution of the reward function is considered in this study:

$$r_{t+1} = r_t + e_t \tag{5.16}$$

$$\tilde{r}_t = r_t + \nu_t \tag{5.17}$$

$$s.t.\ r_{t+1}, r_t, \tilde{r}_t \leq 0 \tag{5.18}$$

where e and ν represent uncorrelated lumped state and the observation noises owing to uncertainties such as DP cell measurement errors, with variances σ_e and σ_ν respectively. Equation (5.16) shows that the consecutive rewards have similar values because the level in the PSV can be considered as a slow dynamic process [137].

The non-positiveness of the reward that would skew the uncertainty distributions (as shown in equation (5.15)) makes this estimation problem challenging, especially when the reward is close to zero (or maximum). To overcome these challenges, this study proposes a constrained estimation scheme by using a truncated normal distribution that captures skewness, where the reward transition is a truncated normal distribution with a PDF that is shown in equations (5.19) and (5.20) [291].

$$p(r_{t+1}|r_t) = TN(r_t, \sigma_e, -\infty, 0) = \frac{\phi(\frac{r_{t+1}-r_t}{\sigma_e})}{\sigma_e \Phi(\frac{-r_t}{\sigma_e})}{}_{\leq 0}(r_{t+1}) \tag{5.19}$$

$$p(\tilde{r}_t|r_t) = TN(r_t, \sigma_\nu, -\infty, 0) = \frac{\phi(\frac{\tilde{r}_t-r_t}{\sigma_\nu})}{\sigma_\nu \Phi(\frac{-r_t}{\sigma_\nu})}{}_{\leq 0}(\tilde{r}_t) \tag{5.20}$$

where TN denotes the PDF of a truncated normal distribution, ϕ and Φ are the PDF and cumulative density functions (CDF) of a standard normal distribution, and is the indicator function respectively. Equation (5.19) represents the true reward dynamics and equation (5.20) is the PDF of the observed noisy reward. The truncated normal distribution can represent the non-positiveness as shown in Fig 5.5-a. However, filtering cannot be performed analytically by using the PDFs in equations (5.19) and (5.20) since they are truncated Gaussian distributions. Thus, this study develops a particle filter (PF) to approximate the *a posteriori* distribution of the reward function.

As shown in Fig. 5.4, at each time instant, a frame and its corresponding DP sensor measurement are obtained from the experiment by means of a camera and the DP sensor, respectively. The agent observes a portion of the image, x_t and takes an action, u_t, which moves the cropping box as discussed earlier. The corresponding noisy reward is obtained by using equation (5.15), and is filtered through equations (5.19) and (5.20). The filtered reward estimate is then used to calculate TD-error, which is used to train the entire structure. The CNN and ConvLSTM layers are updated by using equation (5.21) [270].

$$\begin{aligned}\Psi \leftarrow & \ \Psi + 0.5 \times \alpha_c \nabla_{\Psi^L} \delta(\cdot)^2 \\ & + 0.5 \times \alpha_a \nabla_{\Psi^L} \delta(\cdot) \ln \pi(\cdot) + \Xi \pi(\cdot) \ln \pi(\cdot)\end{aligned} \tag{5.21}$$

where $\Psi = [\psi_{CNN}, \psi_{ConvLSTM}]$ represents the parameters of the CNN and the ConvLSTM layers. Ξ is a weight coefficient that adjusts a penalty to encourage action diversity. L represents the local networks. Additional explanation about the sampling procedures will be provided in Section 5.4.1. More details about the experimental procedure and the neural networks have been provided in [270].

5.4 Results and Discussion

The proposed algorithm should filter the reward function while respecting its constraints, compromising neither the learning nor the tracking abilities of the agent. This section explains the details of the implementation, compares the filtering performances of different techniques, and finally presents the results from both the learning and tracking point of view.

5.4.1 Implementation Details

Images from the pilot-scale experimental setup were obtained using D-Link DCS-8525LH camera at 15 frames per second (FPS). From the 15 FPS footage, a representative image for each second was obtained. Hence, 80 images from 80 consecutive seconds were obtained with necessary down-sampling [270]. These images were then paired with their corresponding pressure sensor readings.

Off-line training was performed after creating a continuous trajectory of the interface level by manually ordering 80 unique images with their corresponding pressure sensor readings. This trajectory was then repeatedly shown to the agent following an approach as proposed in [270]. At any time, the agent observed only the pixels within the cropping box. More details about the dynamic environment can be found in [270].

The detailed proposed algorithm is shown in Algorithm 5.1. As shown in the algorithm, first of all, the *vanilla* A3C algorithm [73] utilizes N-local networks and a

Algorithm 5.1 Interface Tracking Under Uncertainty by using the A3C algorithm

```
1. Input: $\alpha_a, \alpha_c, \gamma, T_{max}$
2. Global network parameter vectors $\omega^G$ and $\theta^G$, and global shared time-step $T \leftarrow 0$
3. Worker-specific parameter vectors $\omega^L$ and $\theta^L$, and worker time-step $t \leftarrow 1$
4. Randomly initialize $\omega^G$ and $\theta^G$
5. Instantiate $N$ cropping boxes that are located at $c_t^L$
6. Repeat **until** $T > T_{max}$
    • Reset gradients $d\omega \leftarrow 0$, $d\theta \leftarrow 0$
    • Sync. workers' parameters $\theta^L = \theta^G$, $\omega^L = \omega^G$
    • $t_{start} = t$
    • Receive an image $I_t$, observe $x_t = Crop(I_t, c_t^L)$
    • Repeat Until $t - t_{start} = t_{max}$
       (a) Sample an action from the policy $u_t \sim \pi(u_t|x_t, \theta_L)$
       (b) Move the cropping box to $c_{t+1}^L = c_t^L + u_t$
       (c) Receive $I_{t+1}$, observe $x_{t+1} = Crop(I_{t+1}, c_{t+1}^L)$
       (d) Calculate noisy $\tilde{r}_{t+1} = -|\widetilde{DP}_{t+1} - c_{t+1}^L|$
       (e) Filter the noisy reward, $\widehat{r}_{t+1} = PF^L(\tilde{r}_{t+1})$
       (f) Update $t \leftarrow t+1$, and $T \leftarrow T+1$
    • $V' = \begin{cases} 0, & \text{if } t - t_{start} = t_{max} \\ V(x_t|\omega^L), & \text{otherwise, bootstrap from last state} \end{cases}$
    • For $j = t-1$ to $t_{start}$
       – Calculate, $\delta \leftarrow \widehat{r}_j + \gamma V' - V(x_j|\omega^L)$
       – Accumulate gradients wrt. $\theta^L : \theta \leftarrow d\theta + \nabla_{\theta^L} \log \pi(u_j|x_j, \theta^L)\delta$
       – Accumulate gradients wrt. $\omega^L : d\omega \leftarrow d\omega + \nabla_{\omega^L}(\delta^2)$
    • Asynchronously update: $\theta^G \leftarrow \theta^G + \alpha_a\, d\theta$, and $\omega^G \leftarrow \omega^G + \alpha_c d\omega$
7. Output: $\pi$
```

global network, denoted as the superscripts L and G in Algorithm 5.1, respectively. These local networks interact with the video starting at different points and promote data diversity. Given this, the proposed scheme can be used with multiple workers. In this article, four local networks can result in rapid convergence, while not increasing the computational complexity drastically. During the test phase, a single worker is used as there is only one environment available.

Then, the $Crop(I, c)$ function takes in an image I_t and crops it at a scalar position c_t. The output matrix, x_t, has a smaller size compared to I_t. The cropping operation removes the redundant pixels naturally and reduces the computational burden significantly. Although a single image could be used during learning, the agent receives a new image after each action in testing or application. Therefore, the agent learns the dynamic features of the separation process (*e.g.*, the movement of the liquid, small variations in lighting, and sensory noise in the image), and generalizes it.

Furthermore, the filtering step in Algorithm 5.1 is a constrained one due to the non-positiveness of the reward, which is achieved through a particle filtering based on the dynamic model (5.16)-(5.18). The recursive steps of the PF method are shown in Algorithm 5.2. There are various PF methods available in the literature that vary depending on their resampling schemes (*i.e.*, steps 5 and 12 in Algorithm 5.2.) [292]. For example, the systematic re-sampling scheme is one of the computationally efficient methods, and the residual sampling has been reported to reduce the variance due to re-sampling [283]. In this study, a sequential importance resampling (SIR) particle filter with $N_p = 500$ particles is used, for which the multinomial re-sampling scheme [31] yields a lower mean squared prediction error compared to the other schemes. Note that this step does not change the proposed method conceptually, and hence, a different type of re-sampling scheme may also be used. Moreover, the particle weights can be updated autoregressively as shown in [292]. $\sigma_e = 0.016$ and $\sigma_\nu = 0.008$ in the algorithm correspond to the noise variance in the reward transition and the observation, respectively. These values result in the lowest MSE value empirically but may depend on the system dynamics.

Note that although this work focuses on estimating a scalar reward, the proposed method can be used for estimating higher dimensional rewards (as in [293]).

In this case, computational burden and the accuracy should be considered. Either less complex resampling techniques can be chosen or filtering can be parallelized on central/graphics/tensor processing units o reduce the computational time [294]. Regularization techniques like roughening or prior editing can be used to improve accuracy [31].

Since the proposed scheme involves both filtering, learning, and tracking, the corresponding results are shown separately in the following sections. Section 5.4.2 compares the proposed filter with the average filter [290] and the *vanilla* Kalman [277] filter.

Algorithm 5.2 Particle filtering with Multinomial Resampling, $\hat{r}_t = PF(\tilde{r}_t)$

```
1. input $\tilde{r}_t$
2. **if** t==1 **then** Sample $N_p$-particles from a truncated normal distribution, $P_1^{N_p \times 1} \sim TN(\tilde{r}_1, \sigma_e, -\infty, 0)$
3. **else**
    • Observe $\tilde{r}_t$
    • Predict: $P_t^{N_p \times 1} \sim TN(P_{t-1}, \sigma_e, -\infty, 0)$
    • Calculate the weight vector, $\alpha^{N_p \times 1} = \frac{\phi\left(\frac{\tilde{r}_T - P_T}{\sigma_\nu}\right)}{\sigma_\nu \Phi\left(\frac{-\tilde{r}_T}{\sigma_\nu}\right)}$
    • Normalize the weights, $\alpha = \frac{\alpha}{\max(\alpha)}$
    • Re-sample, $P_t = Resample(P_t, \alpha)$
4. **end if**
5. Calculate the sample mean of the particles, $\hat{r}_t = \bar{P}_t$
6. Output: $\hat{r}_t$
```

5.4.2 Results: Particle Filtering

The initial step in the proposed scheme is to apply the particle filter. As mentioned in the previous sections, this filter takes the constraints as well as the skewed distribution into account in both the reward transition and the observation uncertainties. To demonstrate the effectiveness of the proposed scheme, a set of noisy observations are also filtered by using the average filter (AVG) [290], the *vanilla* Kalman filter [31] and the truncated Kalman filter (TKF) [279].

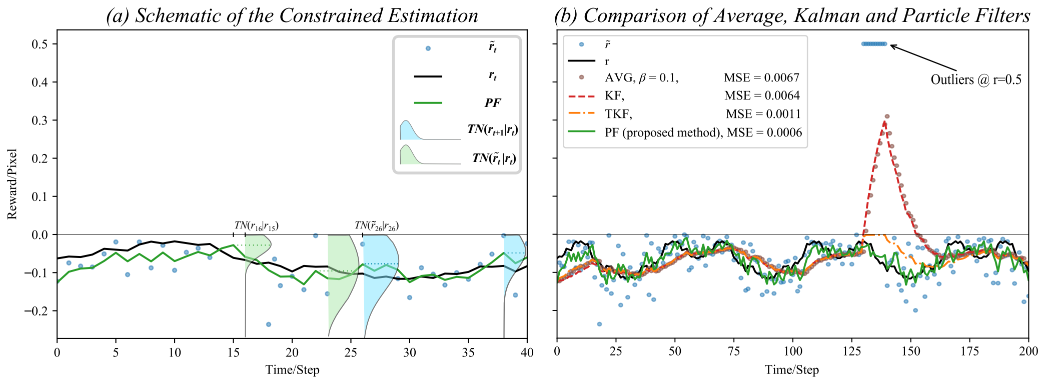


Figure 5.5: Proof of the "constrained filtering" concept. The blue dots are the noisy observations, $\tilde{r}$, the black curve is the true reward, r, the green line is the estimation of the PF. The horizontal line at $r = 0$ represents the constraint. **(a)** Schematic of the proposed method. The blue PDF represents the reward transition, and the green one is for the observation. Note that at any time step both the reward and the observation functions are constrained to zero. **(b)** Comparison of an average (AVG, the brown dots), the Kalman [31] (KF, the red dashed line), a truncated Kalman (TKF, the orange dash-dotted line) [279], and the proposed particle filter (PF, the green line), with MSE values of 0.0067, 0.0064, 0.0011 and 0.0006 respectively. Note the overall performance of the PF is better than the other algorithms. To demonstrate the effectiveness of the constrained filtering, some adversarial disturbance points (outliers) are introduced between $t = [130, 140]$. Note that when the observations are positive, the Kalman and the avg. filters make unconstrained estimations, whereas the TKF and PF estimations are bounded to zero. Moreover, TKF makes biased estimations since it truncates the distribution after estimation, whereas the PF truncation is considered during the estimation.

In the original implementation, the average filter utilizes the value function in the prediction step. In this comparison, its simplified version is used by ignoring the effect of the value function. The average filter is defined in equation (5.22).

$$\hat{r}_{t+2} = \hat{r}_{t+1} + \beta(\tilde{r}_{t+1} - \hat{r}_{t+1}) \tag{5.22}$$

where β is a fixed weight and $\hat{r}_1 = \tilde{r}_1$. In the example of this article, $\beta = 0.1$ yielded the lowest MSE.

During the comparison, in addition to the constrained observation noise, 10 positive adversarial observations or outliers are introduced at $t = [130, 140]$. As shown in Fig. 5.5-b, both the average filter and the Kalman filter violate the constraint, whereas the TKF and PF with truncated normal distributions avoid the constraint violation. In addition, TKF results in biased estimates because it truncates the estimation after the update step, which is not optimal. On the other hand, the proposed PF estimates the reward more accurately since it takes the constraints into account during both the prediction and the update steps as shown in equations (5.19) and (5.20). Note that the particles within the "adversarial attack" or outlier region have zero weights for the PF. The MSE values up to $t = 125$ for the AVG, KF, TKF and PF are 0.011, 0.012, 0.010, and 0.005, respectively. This indicates that the proposed method outperforms the abovementioned methods in terms of accuracy. Hence, it will be adopted in the following experiments.

5.4.3 Results: Learning with PF

One of the main challenges in deep RL is that the noise may deteriorate the policy performance [275]. Several heuristic schemes have been proposed to mitigate this problem [76,295]. However, the error introduced by the noise in the reward is naturally skewed and should be constrained in control problems. Moreover, the numerical issues related to the noise may also reduce the sample efficiency or even stop the learning completely. By considering these, the benefit of the proposed scheme is investigated from various perspectives.

Consider three policies, π_1-π_3, trained on the same visual but with different reward data. That is, π_1 was trained without any noise. π_2 was subject to two Gaussian noises

$\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0, 0.05)$ in the reward transition and the observation, respectively. While training π_3, the same noises as π_2 are used while the PF is used as a filter to reduce the noise effect.

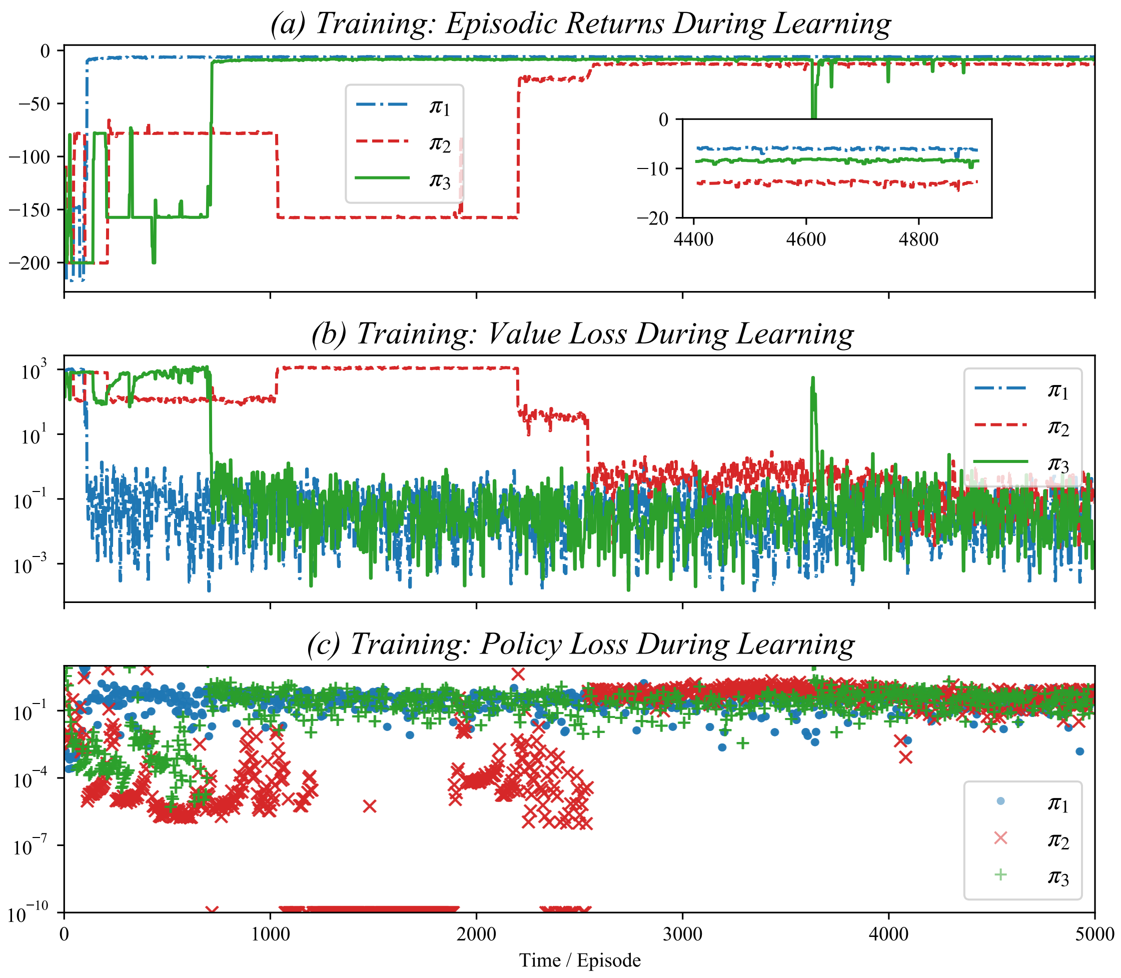


Figure 5.6: Performances of three policies during training. **(a)** Episodic return (equation (5.1)) is a common performance metric. In the ideal case, it should converge to zero, however, the maximum value is -5 in this problem due to the initial position of the cropping box. As shown in the figure, the noise corrupts π_2 initially and delays the learning of a good policy significantly. It also has a lower asymptotic return compared to the other policies. On the other hand, π_3 results in a closer asymptotic performance to π_1 due to the PF. **(b)** The A3C agent tries to minimize the TD-error by optimizing the critic loss shown in equation (5.8). The figure shows that π_1 and π_3 can minimize the value loss quickly, however, the noise in π_2 causes huge value losses, especially between episodes 1000 and 2000. **(c)** Since the policy is also optimized by using the TD-error, its loss should be stable to avoid numerical issues. π_1 and π_3 demonstrate similar performances in terms of the policy loss as expected. However, the excessive noise causes π_2 to become unstable between episodes 1000 and 3000.

As shown in Fig. 5.6, the noise deteriorates the learning performance significantly. It causes the agent to converge to an optimal policy notably later, increases the value loss remarkably, and makes the policy loss function unstable. On the other hand, the proposed PF approach reduces the variation originating from the DP sensor measurements and yields a better overall performance during the training.

5.4.4 Results: Interface Tracking

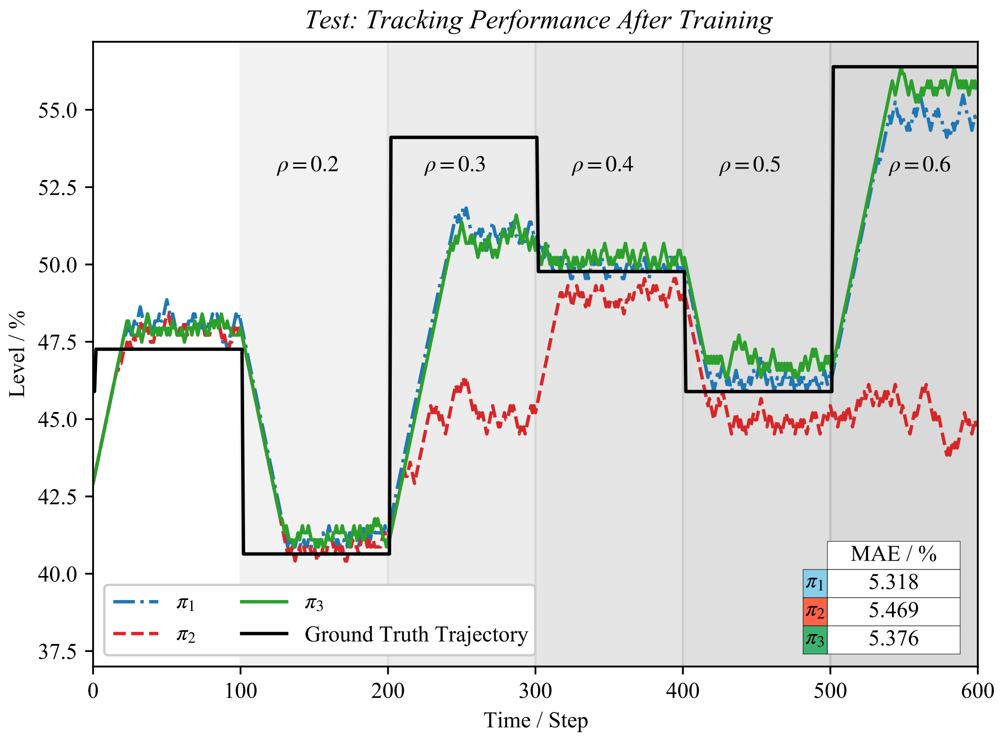
The ultimate goal of this study is to track the interface by using an RL agent. This section tests the agent's performance during the testing phase without further training.

During the test phase, the agent was subject to a discontinuous interface trajectory that consisted of previously unseen images. At this testing/application phase, the agent utilized only the image frames without relying on the DP sensor measurements. The agent was also subject to excessive noise and occlusion (whose magnitude is denoted as ρ) to make the tracking task more challenging. The experiment follows the same procedure as that of [270].

Fig. 5.7 shows that the policy trained in the presence of sensory noise (*i.e.*, π_2) tracks the interface successfully when the occlusion magnitude is below 30%. However, its inconsistent actions result in biased tracking afterwards. On the other hand, π_1 and π_3 track the interface more accurately up to an occlusion magnitude of 60%. π_3 demonstrates a slightly better performance at 60% occlusion, while the overall performance of π_1 is the best within the three policies in terms of the mean absolute error. However, policy π_1 is trained in the absence of any noise, which is not realistic. As a result, policy π_3 as proposed in this chapter demonstrates its superior and practical performance.

5.5 Conclusion

Even though data-driven techniques are becoming popular in industries, many RL schemes consider noise-free signals during training. However, noise is inherent in real processes, and its properties depend on the type of system of interest. Such challenges



	MAE / %
π_1	5.318
π_2	5.469
π_3	5.376

Figure 5.7: The tracking performance of the agent by using the three policies on a discontinuous trajectory. The black line represents the actual position of the interface, the coloured lines represent the center of the cropping box by using the corresponding policies. $\rho = 0.5$ indicates that 50% of the box is visually blocked between $t = [400, 500]$. Note that when the amount of occlusion is low, both policies perform similarly. However, when the amount of occlusion increases, π_2 starts behaving inconsistently and yields a biased tracking performance. On the other hand, π_1 and π_3 track the interface closely, and result in lower MAE values.

require robust solutions as represented in this work.

This study has focused on one of the common rewards in control systems, namely the L1-norm. The behaviour of this type of reward in the presence of noise was explained.

An estimation scheme for constrained states and observations was developed by using a particle filter. Furthermore, the proposed scheme was successfully integrated with a multi-worker RL agent by using noisy sensory data. The impacts of noise on the RL agent have been qualitatively and quantitatively studied, by evaluating the training and test performances through a pilot-scale experiment. Despite the studied reward being L1-norm, the proposed scheme can be extended to other norms.

Chapter 6

Skew Filtering for Online State Estimation and Control [*]

This chapter proposes a skew filtering scheme for online state estimation and control. Process optimization and control can become challenging when the measurements are affected by irregular noise. Classical approaches utilize Gaussian methods like Kalman filtering to alleviate the sensory noise. However, many industries involve skewed noise in their processes. While the closed skew-normal (CSN) distribution generalizes a Gaussian distribution with additional parameters, its dimension increases during recursive estimation, making it impractical. Even though there exist some techniques for the solution, they are typically either too complicated or inaccurate for higher-dimensional problems. This study proposes a novel online optimization scheme to reduce the dimensionality of a CSN distribution while considering the properties of the complete empirical distribution. Since the objective function used during the optimization step considers the geometry of the metric space, the proposed scheme achieves higher accuracy without sacrificing computational efficiency. After finding the reliable combination of objective function and optimizer, the proposed filter is applied to two real-time pilot-scale experiments. The results indicate that it is beneficial for recursive state estimation in the presence of skewed noise.

[*]This chapter has been submitted to *IEEE Transactions on Industrial Informatics* as **O. Dogru**, R. Chiplunkar and B. Huang, ”Skew Filtering for Online State Estimation and Control”.

6.1 Introduction

Industrial processes require safe, economically feasible operation while providing optimal production. To satisfy such requirements, sensory measurements are utilized in monitoring and control applications. However, these measurements are often affected by physical limitations that reduce the measurement quality. Various techniques have been developed to quantify and mitigate the noise such that the operational requirements are met efficiently and effectively. A common assumption in such techniques is that the state of interest and the sensory measurements exhibit Gaussian properties.

In the presence of Gaussian noise in both state and measurements, one can obtain an optimal estimation by using a Bayesian filter, or particularly a Kalman filter [296]. Because the Kalman filter consists of simple linear operations, the algorithm can be applied in both online and offline settings, which is favourable in various applications. Although assuming Gaussian behaviour provides effective solutions while estimating the variables of interest, the noise affecting state and measurements may not be completely Gaussian. In this case, more complex filters with additional assumptions and modifications may be needed.

This study focuses on a type of continuous probability distributions, where a Gaussian distribution is generalized with non-zero skewness. This property can be found in variables that range from truncated Gaussian to normal distributions. Some examples of variables with such properties can be found in the domains of actuarial science [297], aerospace engineering [298], biology [299], chemical engineering [300, 301], climatology [302], communication/electronics [32], defence [303], earth sciences [304], economics/finance [305], forestry/remote sensing [306], mathematics [307], process systems and control [14, 308], and statistics [309]. In the presence of noisy measurements, Bayesian estimation with distributions like skew-t [310] or Weibull can provide optimal estimations for these variables. Despite that these distributions can explain the skewness of the variables, they may not be sufficient to recover the actual distribution due to their low degree of freedom, or they may not have a closed-form solution in the estimation scheme, which increases the complexity and computational burden [311].

On contrary, a CSN distribution is a function with five parameters that generalizes the normal distribution. This generalization accounts for nonzero skewness as well as different location and scale information that can further adjust the shape of the distribution [312]. The CSN distribution is closed under the linear transformation of CSN variables, marginalization, and Bayesian inversion [313]. These properties are vital in developing recursive filtering schemes for linear dynamic systems characterized by skewed noises. [298] provides an overview of the filtering schemes for linear systems with CSN noise. Additionally, several methods such as ensemble filter [298], unscented Kalman filter [307], etc. have been developed for nonlinear systems with CSN noise. Although a CSN filter can provide effective estimations given the noisy observations, it suffers from increasing dimensionality during recursive Bayesian estimation that makes the filter impractical for online implementations. Despite several approaches that have been proposed to solve the dimensionality problem, they either introduce the skewness indirectly in the initial state [314] or consider partial information about the distribution of interest [300], which may yield suboptimal estimates.

This chapter, on the other hand, proposes a numerical optimization technique to overcome the increasing dimensionality problem without loss of generality. Since the proposed technique takes the entire CSN distribution into account at every time step, it provides optimal approximations to the actual high-dimensional distribution. In addition to online state estimation, the filter can also be used to develop advanced controllers such as reinforcement learning (RL) agents or model predictive controllers (MPC). The contributions of this study are as follows:

- Developing a dimensionality reduction technique for online skew state estimation,
- Theoretically and empirically comparing different objective functions and optimization tools for the proposed method,
- Utilizing the proposed filtering scheme in online RL and MPC applications,
- Quantitatively analyzing the proposed scheme by considering both filtering and control through two pilot-scale experiments.

In addition to these contributions, detailed background information is provided in Section 6.2, the proposed method is described in Section 6.3, the results are discussed in Section 6.4, and the concluding remarks are presented in Section 6.5.

6.2 Background

A process can be controlled by using different techniques such as learning-based or model-based controllers if the state of interest is available at all time steps. Since the practical control or monitoring schemes rely on noisy measurements, an accurate state estimation scheme is required to achieve satisfactory control performance. This study develops an optimal state estimator in the presence of skew measurement noise and applies it to real time state estimation and control problems.

6.2.1 Kalman Filtering (KF)

Physical processes may be represented by a discrete-time state-space model as shown in Eqns. (6.1) and (6.2).

$$x_{t+1} = A_t x_t + B_t u_t + E_t \epsilon_t \tag{6.1}$$

$$y_t = C_t x_t + D_t u_t + F_t \psi_t \tag{6.2}$$

where $x_t \in \mathbb{R}^n$, $u_t \in \mathbb{R}^q$, $y_t \in \mathbb{R}^p$ are the state, action, measurement respectively. t, ϵ_t and ψ_t, respectively, represent the discrete time step, state, and measurement noises. A_t, B_t, C_t, D_t, E_t, and F_t adjust the contribution of each element to the state/measurement and can be determined by using a proper system identification method [29].

In practice, the true value of the state of interest, x_t, may be unknown. Based on Bayes' theorem, KF can be used to calculate the estimated state, $\hat{x}_t$, optimally in the presence of Gaussian state and measurement noise. The filter, at each time step, estimates a mean and covariance for the state of interest given the noisy measurement. Note that if $C_t = 1$ and $D_t = 0$, then estimating $\hat{y}_t$ is equivalent to estimating $\hat{x}_t$. After simplifying the state space model as $D_t = 0$ and $E_t = F_t = 1$, the Kalman filter can be derived as follows [31]:

$$\hat{x}_t^- = A_{t-1}\hat{x}_{t-1}^+ + B_{t-1}u_{t-1} \tag{6.3}$$

$$P_t^- = A_{t-1}P_{t-1}^+A_{t-1}^T + \Sigma_{\epsilon,t-1} \tag{6.4}$$

$$K_t = P_t^- C_t^T (C_t P_t^- C_t^T + \Sigma_{\psi,t})^{-1} \tag{6.5}$$

$$\hat{x}_t^+ = \hat{x}_t^- + K_t(y_t - C_t\hat{x}_t^-) \tag{6.6}$$

$$P_t^+ = (I - K_tC_t)P_t^- \tag{6.7}$$

where $(\cdot)^-$ and $(\cdot)^+$ represent a priori and a posteriori estimates respectively. K_t is the Kalman gain, and $\Sigma_{\epsilon,t}$ and $\Sigma_{\psi,t}$ are the known state and measurement covariances.

6.2.2 Closed-Skew Normal (CSN) Distribution

The multivariate CSN generalizes the Gaussian distribution by introducing skewness into Gaussian models and was introduced in [312]. Consider two random vectors, $\tau \in \mathbb{R}^{n_2}$ and $\xi \in \mathbb{R}^{q_2}$. Joint probability density function (PDF) of τ and ξ can be written as:

$$\begin{bmatrix}\tau \\ \xi\end{bmatrix} \sim \mathcal{N}_{n_2+q_2}\left(\begin{bmatrix}\mu_\tau \\ \mu_\xi\end{bmatrix}, \begin{bmatrix}\Sigma_\tau \, \Gamma_{\tau\xi} \\ \Gamma_{\xi\tau} \, \Sigma_\xi\end{bmatrix}\right) \tag{6.8}$$

where μ_τ and μ_ξ are location vectors, and $\Sigma_\tau, \Gamma_{\xi\tau}$ and Σ_ξ are covariance matrices. Then, the CSN variable, x, can be defined as $x = [\tau|\xi \geq 0]$ with a distribution function given in Eqn. (6.9).

$$\begin{aligned} x \sim p(x) = p(\tau|\xi \geq 0) &= \frac{p(\xi \geq 0|\tau)p(\tau)}{p(\xi \geq 0)} \\ &= [1 - \Phi(0; \mu_\xi, \Sigma_\xi)]^{-1}[1 - \Phi(0; \mu_{\xi|\tau}, \Sigma_{\xi|\tau})]\phi(\tau; \mu_\tau, \Sigma_\tau) \end{aligned} \tag{6.9}$$

where Φ and ϕ are the cumulative distribution function (CDF) and PDF of an n_2 dimensional Gaussian, respectively. A multivariate CSN can be parameterized as shown in Eqn. (6.10).

$$\begin{aligned} &\mathrm{CSN}_{n_2,q_2}(\mu, \Sigma, \Gamma, \nu, \Delta) \\ &= [\Phi_{q_2}(0; \nu, \Delta + \Gamma\Sigma\Gamma^T)]^{-1}\Phi_{q_2}(\Gamma(x - \mu); \nu, \Delta)\phi_{n_2}(x; \mu, \Sigma) \end{aligned} \tag{6.10}$$

where Γ is the skewness, μ and ν are the location, and Σ and Δ are the scale parameters, respectively. It can be noted that setting $\Gamma = 0$ reduces the CSN distribution to a Gaussian distribution and hence, CSN distribution generalizes the Gaussian distribution.

6.2.3 Online Control Examples: Model Predictive Control (MPC) and Reinforcement Learning (RL)

After estimating the state of interest in the process, a controller can be used to better achieve the desired performance. For example, an MPC [315–317] is a model-based advanced controller that finds the optimal controller action given a state. This state is fed into a model that predicts the future behaviour of the system. An optimizer (often quadratic), then finds the best controller actions that would yield the optimal performance. Although the sense of optimality depends on the market and can vary significantly, a common optimality measure can be formulated as shown in Eqn. (6.11).

$$\text{QCF}_t = ||x_t - x_{t,sp}||^2_{Q_t} + ||u_t - u_{t,ref}||^2_{R_t} \tag{6.11}$$

where QCF is a quadratic cost function that is to be minimized at every discrete time step. $x_{t,sp}$ is a setpoint that is defined by a user or a production planning program. $u_{t,ref}$ is a nominal/reference control action, which can also be zero or take the previous action, depending on the controller design. $Q_t \in \mathbb{R}^{p\times p}$ and $R_t \in \mathbb{R}^{q\times q}$ adjust the importance of the state and action in QCF. QCF aims to find a balance between the state (often a product) quality and the controller effort (related to the energy spend). Although this formulation works when states are directly measured, x_t may not be available in practice. Instead, $\hat{y}_t = C_t\hat{x}_t$ is inferred from noisy measurements and utilized in the control task. The resulting optimization problem can be defined as:

$$\begin{aligned} J(u_t) = &\sum_{k=1}^{N_P} ||\hat{y}_{t+k} - y_{t+k,sp}||^2_{Q_k} + \sum_{k=0}^{N_C-1} ||\hat{u}_{t+k} - u_{t+k,ref}||^2_{R_k} \\ &\text{s.t. Eqns. } (8.1), (8.2) \\ &\quad u_{min} \le u_t \le u_{max}, \forall t \end{aligned} \tag{6.12}$$

where J is the cost function, k is a discrete time step, y_{sp} is the setpoint for the observations, N_P and N_C are the prediction and control horizons, and u_{min} and u_{max} represent the input constraints respectively. The *receding horizon* strategy calculates a sequence of $\mathbf{u}_t = \{u(0), u(1), ..., u(N_C - 1)\}$ (which is the solution to the open-loop optimization problem) and implements the first control action, $u_t = u(0) \in \mathbf{u}_t$, in the plant as represented in Eqn. (6.13).

$$\mathbf{u}_t = \arg\min_{\mathbf{u}_t} J(\mathbf{u}_t) \tag{6.13}$$

An advantage of using an MPC is that it can handle various constraints. However, the resulting optimization problem cannot be solved analytically in the presence of constraints, and thus, numerical methods are needed.

RL-based controllers provide alternative solutions to MPC either relying or without relying on models to control the process [40, 318–322]. In the model-free setting, an *agent* interacts with the process by taking the best actions given the current states. Unlike MPC, the agent constitutes a *policy*, $\pi(u_t|x_t)$, which is learned through interaction with the process. During this interaction, a *reward* signal, $r_t \in \mathbb{R}$, is obtained by using sensors, and the agent's goal is to maximize the reward over time by minimizing the *temporal difference (TD) error*. Table 6.1 presents the TD-error in the most commonly used RL algorithms.

Table 6.1: TD-error in the most commonly used RL algorithms.

Algorithm	**TD-error**
DDPG [74]	$r_t + \gamma \mathcal{Q}(x_{t+1}, \pi(x_{t+1})\|\Xi') - \mathcal{Q}(x_t, u_t\|\Xi)$
A2C/A3C [73]	$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(x_{t+k}\|\Xi') - V(x_t\|\Xi)$
TD3 [76]	$r_t + \gamma \min_{i=1,2} \gamma \mathcal{Q}_{\Xi_i'}(x_{t+1}, u_t\|\Xi_i') - \mathcal{Q}(x_t, u_t\|\Xi_{i=1,2})$

where Ξ represents the approximated function parameters to be trained. V and $\mathcal{Q}$ are the value estimations for the state and the state-action pair, respectively. $\gamma \in [0, 1)$ is a *discount factor* that adjusts the contribution of the value and reward functions. Because the state, action, and reward information is utilized in TD-error without any system models, the performance of the agent heavily depends on the accuracy of the sensor that inevitably degrades due to noise, as has been discussed in [323]. This study demonstrates the benefits of the proposed method by using a reward (state) trajectory obtained from an asynchronous advantage actor-critic (A3C) agent. However, the proposed method can be used in any online RL algorithm that involves noisy state or reward measurements in real-time.

Regardless the type of controller, accurate sensory information is required to effectively control the system. There is extensive literature about KF, MPC and KF. Since this study focuses on improving the filtering scheme in the presence of skew noise, the details are skipped. More information about KF, MPC and RL can be found in [289, 324–326].

6.3 Optimal Dimensionality Reduction for Online Implementation

Similar to the Kalman filter, CSN parameters can be calculated recursively [298]. Assume $\epsilon_t \sim \mathcal{N}(\epsilon_t; 0, \Sigma_\epsilon)$ and $\psi_t \sim \mathrm{CSN}(\psi_t; \mu_\psi, \Sigma_\psi, \Gamma_\psi, \nu_\psi, \Delta_\psi)$. For the sake of simplicity in notation, assume that the statistical properties of ϵ and ψ do not vary over time. The prediction step for the corresponding CSN filter can be derived as shown in Eqn. (6.14) [298].

$$\begin{aligned}
\hat{x}_t^- &= A_{t-1}\hat{x}_{t-1}^+ + B_{t-1}u_{t-1} \\
P_t^- &= A_{t-1}P_{t-1}^+A_{t-1}^T + \Sigma_\epsilon \\
\Gamma_t^- &= \Gamma_{t-1}^+\omega_t \\
\nu_t^- &= \nu_{t-1}^+ \\
\Delta_t^- &= \Delta_{t-1}^+ + \Gamma_{t-1}^+(I - \omega_t A_{t-1})P_{t-1}^+\Gamma_{t-1}^{+T} \\
\text{with } \omega_t &= P_{t-1}^+A_{t-1}^T(A_{t-1}P_{t-1}^+A_{t-1}^T + \Sigma_\epsilon)^{-1}
\end{aligned} \tag{6.14}$$

Furthermore, the update step can be derived as follows [298]:

$$\begin{aligned}
\hat{x}_t^+ &= \hat{x}_t^- + K_t(y_t - C_t\hat{x}_t^- - \mu_\psi) \\
P_t^+ &= (I - K_tC_t)P_t^- \\
\Gamma_t^+ &= \begin{bmatrix} \Gamma_t^- \\ -\Gamma_\psi C_t \end{bmatrix} \\
\nu_t^+ &= \begin{bmatrix} \nu_t^- - \Gamma_t^-(\hat{x}_t^+ - \hat{x}_t^-) \\ \nu_\psi - \Gamma_\psi(y_t - C_t\hat{x}_t^- - \mu_\psi) \end{bmatrix} \\
\Delta_t^+ &= \begin{bmatrix} \Delta_t^- & 0 \\ 0 & \Delta_\psi \end{bmatrix}
\end{aligned} \tag{6.15}$$

In addition, the expected mean of the state of interest can be calculated as [327]:

$$\begin{aligned}&\mathbb{E}[x_t|x_t \sim \mathrm{CSN}(x_t; \hat{x}_t^+, P_t^+, \Gamma_t^+, \nu_t^+, \Delta_t^+)] \\ &= \hat{x}_t^+ + P_t^+ \Gamma_t^{+T} \frac{\phi(0; \nu_t^+, \Delta_t^+ + \Gamma_t^+ P_t^+ \Gamma_t^{+T})}{\Phi(0; \nu_t^+, \Delta_t^+ + \Gamma_t^+ P_t^+ \Gamma_t^{+T})}\end{aligned} \tag{6.16}$$

Note that the dimension of the parameters that introduce skewness is increased by one at the end of each update step. This complicates the solution because one cannot derive Eqn. (6.16) analytically when the dimension continuously increases (when $q_2 > 1$). In addition, this inevitable increase causes memory issues that make the skew filter impractical for online implementation.

This study proposes a numerical optimization solution to the dimensionality increase problem by approximating the two-dimensional (2-D) CSN distribution, $\mathrm{CSN}_{n_2,2}(x; \mu, \Sigma, \Gamma, \nu, \Delta)$, at the end of each update step. Consider a 1-D CSN, $\mathrm{CSN}_{n_2,1}(x; \Theta)$ where $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$ represents its unknown parameters. The goal is to find the best parameters, $\Theta^* = \{\theta_1^*, \theta_2^*, \theta_3^*, \theta_4^*, \theta_5^*\}$, such that the difference between the two CSN distributions is minimized. Since n_2 is constant during optimization, it is ignored for the sake of simplicity in notation. A simple solution to this problem is to minimize the difference between $\Phi_1(\theta_3(x - \theta_1); \theta_4, \theta_5)$ and $\Phi_2(\Gamma(x - \mu); \nu, \Delta)$ (the CDF components of the 1-D and 2-D CSNs) since the dimensionality increase is observed in the CDF. However, this solution may miss the useful information that $\phi(x; \mu, \Sigma)$ contains.

This study presents an alternative and effective solution where Θ is calculated by considering the five parameters in the distributions. Different distance/divergence indicators such as the Kullback-Leibler (KL), Jensen-Shannon (JS) divergence, total variation distance (TVD), Hellinger distance, density norms etc. can be used to approximate the high-dimensional CSN. However, these indicators do not consider the underlying space geometry and may yield nonsymmetric, discontinuous, or inaccurate results due to their properties, as shown in [328]. They can reduce the reliability of the filter or can make it diverge during the estimation.

Based on transport theory, the Wasserstein distance (WD) [329] calculates the deviation between two distribution functions while addressing the above-mentioned issues, and it can be derived as shown in Eqn. (6.17).

$$\begin{aligned} d_{\mathcal{W}}(\mathrm{CSN}_1, \mathrm{CSN}_2) &= \left[\int_0^1 |F_1^{-1}(z_1|\cdot) - F_2^{-1}(z_1|\cdot)|^{\zeta} \, dz_1 \right]^{1/\zeta} \\ &= \left[\int_{-\infty}^{\infty} |F_1(z_2|\cdot) - F_2(z_2|\cdot)|^{\zeta} \, dz_2 \right]^{1/\zeta} \end{aligned} \tag{6.17}$$

where ζ is the order of the WD, $F_1^{-1}(\cdot)$ and $F_2^{-1}(\cdot)$ are the quantiles, and $F_1(\cdot)$ and $F_2(\cdot)$ represent the CDFs of $\mathrm{CSN}_1(\cdot)$ and $\mathrm{CSN}_2(\cdot)$ respectively as shown in Eqn. (6.18). For notational simplicity, the function parameters are ommitted in Eqn. (6.17).

$$\begin{aligned} F_1(x; \Theta) &= \int_{-\infty}^{\infty} \mathrm{CSN}_1(x; \Theta, z_3) dz_3 \\ F_2(x; \mu, \Sigma, \Gamma, \nu, \Delta) &= \int_{-\infty}^{\infty} \mathrm{CSN}_2(x; \mu, \Sigma, \Gamma, \nu, \Delta, z_4) dz_4. \end{aligned} \tag{6.18}$$

Note that CSN_1 and CSN_2 are assumed to be in the Wasserstein space, $\mathcal{W}$, which makes the WD more effective than the Euclidean distances like the norm functions. Although the WD can be used for any distribution function, the term "CSN" is and will be used for simplicity and demonstration purposes. By definition, the WD satisfies the following useful metric properties:

$$d(\mathrm{CSN}_1, \mathrm{CSN}_2) = 0 \iff \mathrm{CSN}_1 = \mathrm{CSN}_2 \tag{6.19}$$

$$d(\mathrm{CSN}_1, \mathrm{CSN}_2) = d(\mathrm{CSN}_2, \mathrm{CSN}_1) \tag{6.20}$$

$$d(\mathrm{CSN}_1, \mathrm{CSN}_2) \leq d(\mathrm{CSN}_1, \mathrm{CSN}_3) + d(\mathrm{CSN}_3, \mathrm{CSN}_2) \tag{6.21}$$

By using $\zeta = 1$ for simplicity and the considering equal integration step, dz, the optimization function can be formed as shown in Eqn. (6.22).

$$\min_{\Theta} d(\Theta) = \min_{\Theta} \left[\int_{-\infty}^{\infty} |F_1(z|\Theta) - F_2(z|\cdot)| \, dz \right] \tag{6.22}$$

Moreover, the optimization function and the properties would be valid in the presence of discrete random variables. Depending on the design criteria, constraints can be implemented on Θ, which will not be considered in this study.

The nonlinear optimization problem shown in Eqn. (6.22) can be solved optimally by using numerous optimization techniques including sample or gradient-based methods. This study utilizes the sequential quadratic programming (SQP) [330] approach due to its simplicity and generality. For example, in the presence of prior knowledge about Θ, constrained optimization can be applied to find Θ^* by using SQP. We will also compare it with several optimization techniques such as particle swarm (PS) [331], conjugate gradient (CG) [332], Nelder-Mead [333], BFGS [334], L-BFGS-B [335], COBYLA [336], and trust region [337] algorithms empirically.

The state estimation algorithm with skew dimensionality reduction is given in Algorithm 6.1.

Algorithm 6.1 Online State Estimation with Skew Dimensionality Reduction by Using the SQP Algorithm

```
1. Input $\Sigma_{\epsilon,t}, \mu_{\psi_t}, \Sigma_{\psi_t}, \Gamma_{\psi,t}, \nu_{\psi,t}, \Delta_{\psi,t}, A_t, B_t, C_t, u_{t-1}, y_t$.
2. Initialize $x_0^+ = \mathbb{E}[x_0]$, $P_0^+ = \mathbb{E}[(x_0 - x_0^+)(x_0 - x_0^+)^T]$, $\Theta_0 = \Theta_{init}$
3. For each $t$
    • Observe a skew measurement, $y_t = \text{system}(u_{t-1})$
    • Predict the variables of interest using Eqn. (6.14)
    • Update the variables of interest using Eqn. (6.15)
    • Obtain $\Theta_t = \text{SQP}(\hat{x}_t^+, P_t^+, \Gamma_t^+, \nu_t^+, \Delta_t^+, \Theta_{t-1})$ according to Eqn. (6.22)
    • Calculate the mean using Eqn. (6.16)
4. Output $\mathbb{E}[\text{CSN}(x_t^+; \Theta_t)]$
```

Note that the SQP algorithm requires an initial set of parameters, Θ_{init}, that can be tuned based on process knowledge or via a Monte Carlo simulator on a system model. After the initialization step, the proposed algorithm uses the previous parameter set, Θ_{t-1}, to calculate the current parameter set, Θ_t, in order to reduce computational complexity. In the case of sample-based optimizers, Θ_{init} is not required, and Θ_{t-1} does not need to be used in Step 6 of Algorithm 1; however, the sample-based optimizers may require additional exploration/exploitation hyperparameters.

Due to its computational efficiency, this simple yet effective algorithm can also be used in real-time controllers. In the following sections, an MPC (whose details are given in Algorithm 6.2) will be utilized in a pilot-scale experiment. Due to its simplicity and computational efficiency, the MPC optimizer will be SQP, too. Note that the type of the optimizers does not affect the fundamental concepts explained previously. Therefore, the optimizers can be substituted by other optimization algorithms depending on the accuracy, computational efficiency, etc. that may differ based on the process of interest.

Algorithm 6.2 Online MPC with Skew Dimensionality Reduction by Using the SQP Algorithm

```
1. Input $x_{t,sp}, Q, R, u_{min}, u_{max}, N_P, N_C,$ Algorithm 6.1
2. For each $t$
    • Observe a skew measurement, $y_t$
    • Predict the state, $\mathbb{E}[CSN(\hat{x}_t^+;\Theta)] =$ Algorithm 6.1$(\cdot)$ (proposed solution)
3. Obtain $N_C$-inputs, $\mathbf{u} = \text{SQP}(Q, R, \mathbb{E}[CSN(\cdot)], u_{min}, u_{max}, x_{t,sp})$
4. Pick the first input, $u_t = \mathbf{u}(0)$
5. Implement the input in the system to the noisy measurement $y_{t+1} = \text{System}(u_t)$
```

Note that the optimizer in Algorithm 6.2 requires a system model, which is equivalent to the model used in Algorithm 6.1 and is not mentioned explicitly for the sake of simplicity. Details of the SQP algorithm can be found in [330].

Algorithm 6.3 Online RL Reward Filtering with Skew Dimensionality Reduction by Using the SQP Algorithm

```
1. Input RL agent, Algorithm 6.1
2. For each $t$
    • Observe a situation, $s_t$
    • Take an action, $u_t = \text{Agent}(s_t)$
    • Receive a skew reward and the next situation $y_t, s_{t+1} = \text{System}(u_t)$
    • Predict the reward (proposed solution), $\hat{y}_t^+ = \mathbb{E}[CSN] =$ Algorithm 6.1$(\cdot)$
    • If Update: Then, Update agent's parameters (e.g., a neural network, Q-table, etc.) by using $\hat{y}_t^+$.
```

Algorithm 6.3 represents a general data flow in RL agents and solution to reward (considered to be the state) filtering by using the proposed method. In this example, an RL agent interacts with a system in real-time by observing a “situation”, s_t, taking an action u_t, and receiving a skew reward y_t. Note that the situation corresponds to the “state” in the RL literature, which is renamed for notational simplicity. Note that we will showcase a reward filtering example shortly, which is essentially equivalent to state filtering without loss of generality. Although the proposed method has been tested by using an A3C agent, it can be used to remove the skew noise independent of the type of agent.

6.4 Results and Discussion

The proposed method was tested on two pilot-scale experiments, which represent industrial chemical processes. Implementation details, the experimental setup, and the results obtained by using the proposed online skew filtering and control approaches will be presented in this section.

Note that the CSN noise in this study will be assumed to be zero-mean as shown in Eqn. (6.23). In the presence of biased noise, the filters may need to be improved by incorporating a bias correction mechanism.

$$\mathbb{E}[\psi] = \mu_\psi + \Sigma_\psi \Gamma_\psi^T \frac{\phi(0; \nu_\psi, \Delta_\psi + \Gamma_\psi \Sigma_\psi \Gamma_\psi^T)}{\Phi(0; \nu_\psi, \Delta_\psi + \Gamma_\psi \Sigma_\psi \Gamma_\psi^T)} = 0, \ \forall t \tag{6.23}$$

For comparison purposes, an online median and average filter ([35]) will be used as shown in Eqn. (6.24) and (6.25) respectively.

$$\hat{y}^+_{\text{med},t} = \begin{cases} y_t, & \text{if } t < N \\ \text{MED}(y_{t-N}, ..., y_{t-1}, y_t), & \text{otherwise} \end{cases} \tag{6.24}$$

$$\hat{y}^+_{\text{avg},t} = \hat{y}^+_{\text{avg},t-1} + \beta(y_t - \hat{y}^+_{\text{avg},t-1}) \tag{6.25}$$

where MED is the median operator, $N \geq 3$ is the window size, β is a fixed weight, and $\hat{y}^+_{avg,0} = y_0$. As shown in Eqn. (6.26), the mean squared error (MSE) will be utilized to measure the estimation and control quality.

$$\text{MSE} = \frac{1}{N} \sum_{t=1}^{N} (x_t - Y_t)^2 \tag{6.26}$$

where Y_t represents the predicted state, $\hat{y}_t$, for state estimation and setpoint, $x_{t,sp}$, for control. This study aims to minimize the prediction and control errors, MSE_P and MSE_C respectively.

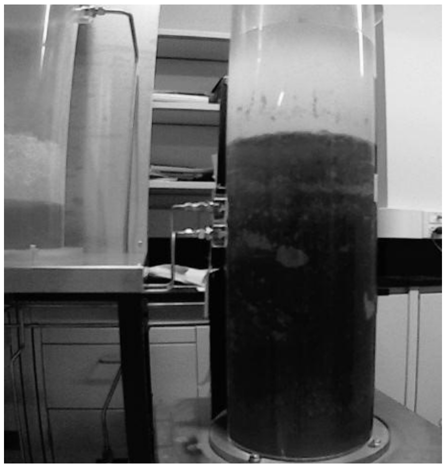


Figure 6.1: Experimental, pilot-scale PSV setup. The black liquid (a water mixture) represents the middlings, the white liquid (oil) mimics the froth layer in the industrial setup.

6.4.1 Implementation Details

The estimator/controller computers have Intel Core i7-4790K CPU (4 CPU cores) at 4.00 GHz, 8 GB DDR4 RAM at 1600 MHz under 64-bit Windows 10 operating system. The filtering and control algorithms were designed in Python v3.7 by using the steps presented in Algorithms 6.1 and 6.2. An open source library, SciPy v1.5.4, was used to design the optimization algorithms. During calculations, real-time communication between the computers (estimator/controller) and experimental setup was established using a socket connection to an Opto22 open platform communication (OPC) server. Nonetheless, the generality of the proposed method does not depend on the type of connectivity. More information about the experimental setup has been given in [338].

6.4.2 Application 1: Online Reward Estimation for the RL Agent in a Primary Separation Vessel (PSV)

The primary separation vessel is a key component in the oil sands industry, where an interface between two liquids is the target variable to be detected and tracked accurately [326]. The tank is schematically shown in Fig. 6.1.

[323] developed a robust interface tracking algorithm, where an RL agent receives a noisy nonlinear reward at each step. Since this reward is utilized for online learning of the agent, it is crucial to reduce the noise effectively in real time.

Without the loss of generality, consider the reward is the state of interest to be estimated ($i.e$, $r_t = x_t$), which can be modelled by using Eqn. (6.27).

$$\begin{aligned} x_{t+1} &= x_t + \epsilon_t \\ y_t &= x_t + \psi_t \end{aligned} \tag{6.27}$$

where $\epsilon_t \sim \mathcal{N}(\epsilon_t; \mu_\epsilon, \Sigma_\epsilon)$ and $\psi_t \sim \text{CSN}(\psi_t; \mu_\psi, \Sigma_\psi, \Gamma_\psi, \nu_\psi, \Delta_\psi)$. Because the RL agent tries to maximize the reward, $r_t = x_t \leq 0$ and $y_t \leq 0$ are often preferred to provide feasibility in control applications [323,326,338]. Due to this property, when the noisy reward magnitude is low, the resulting reward becomes a truncated variable [323]. Inspired from [32,298], we mimic an engineering problem where the sensory measurements are contaminated with skew noise. Since the CSN distribution can generalize the distributions ranging from truncated Gaussian to regular Gaussian, the synthetically added observation noise, ψ, was chosen to follow a CSN distribution. The proposed algorithm will be used to recover the hidden state (true reward) efficiently, as will be discussed in the following paragraphs.

As one of the key components of this study, different distance measures are compared under skew noise with ($\Gamma = 10, \Sigma = 5$), ($\Gamma = 50, \Sigma = 4$) and ($\Gamma = 100, \Sigma = 20$). For optimization, the SQP algorithm was used at this step. As shown in Table 6.2, the Wasserstein-1 distance consistently yielded the lowest MSE for these problems, whereas some of the other functions resulted in either higher MSE or divergence. In addition to this quantitative difference, note that WD's theoretical properties provide quantitative improvements as mentioned previously.

Then, different optimizers were compared in terms of their accuracy and computational speed while using WD as the objective function. As shown in Table 6.3, the SQP algorithm yielded the lowest MSE and execution time compared to the other optimization algorithms due to its simplicity. On the other hand, the other optimizers either caused higher MSE or diverged. Note that some of these optimization schemes were designed for constrained optimization, and thus, the results may differ when there are constraints on Θ. Nevertheless, all tested methods yielded a solution of less than a second for each execution, which showed that all of them could be used for online estimation for systems that have a sampling time of five seconds. In addition,

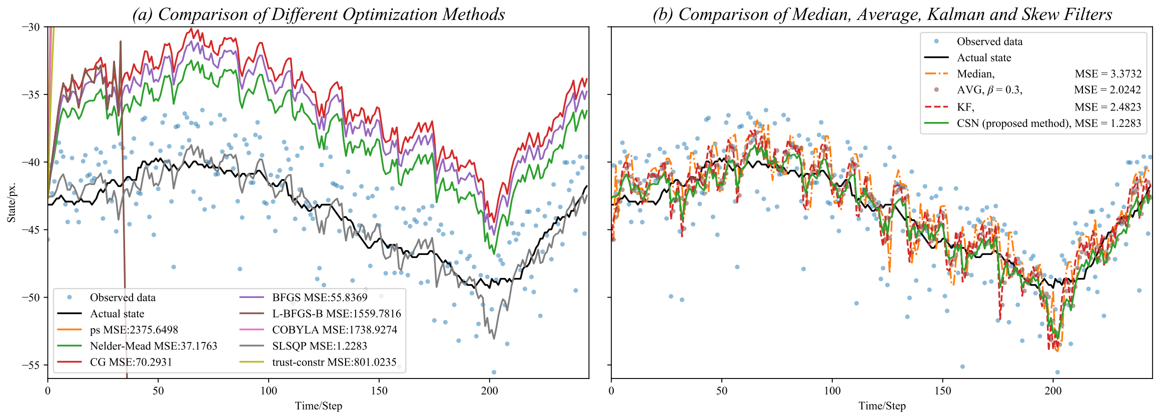


Figure 6.2: Comparison of different a) optimizers b) filters for online state estimation under skew noise with ($\Gamma = 10, \Sigma = 5$). a) Note that all the algorithms except the SQP algorithm diverged during estimation. The SQP algorithm yielded the best performance. b) Median filter showed the worst performance, followed by the Kalman filter. Note that the proposed method significantly outperformed the other methods in terms of estimation accuracy.

Table 6.2: Comparison of different objective functions to minimize the difference between CSN$_1$ and CSN$_2$(*i.e.*, to be used as d in Eqn. (6.22)). Fwd, Rev, Symm-KL represent the forward, reverse and symmetric KL divergence respectively. Note that they are compared under different noise magnitudes with eleven different random seeds, and the MSE is in px. † indicates that at least a trial diverged with a NaN value.

Objective fcn.	MSE for $\Gamma = 10, \Sigma = 5$	MSE for $\Gamma = 50, \Sigma = 4$	MSE for $\Gamma = 100, \Sigma = 20$
χ^2 [339]	NaN	NaN	NaN
Bhattacharyya [340]	1.36E+04 ∓ 1.54E+04	1.51E+04† ∓ 3.10E+03	NaN
TVD [341]	2.46E+01 ∓ 4.06E+01	NaN	1.62E+02 ∓ 1.58E+01
Symm-KL [342]	6.79E+14† ∓ 1.92E+15	1.41E+01 ∓ 1.36E+00	4.94E+04† ∓ 1.27E+05
Fwd-KL [343]	4.74E+30 ∓ 1.50E+31	1.41E+01 ∓ 1.33E+00	6.40E+01 ∓ 6.82E+01
Rev-KL [343]	2.58E+04 ∓ 8.13E+04	7.30E+00 ∓ 1.20E+00	5.78E+16† ∓ 9.30E+16
JS [344]	3.06E+01 ∓ 1.63E+00	5.72E+00 ∓ 5.82E-01	4.07E+01 ∓ 7.66E+00
Hellinger [345]	2.72E+17† ∓ 8.15E+17	6.76E+00 ∓ 6.34E-01	8.58E+04 ∓ 2.52E+05
L_2 [346]	2.95E+01 ∓ 5.42E+00	9.08E+02 ∓ 1.63E+03	5.73E+01 ∓ 9.21E+00
L_1 [346]	1.92E+00 ∓ 3.73E-01	6.76E+00 ∓ 6.34E-01	7.14E+01 ∓ 1.04E+01
Energy [347]	3.16E+01 ∓ 4.34E+00	5.71E+00 ∓ 5.81E-01	1.94E+01 ∓ 4.24E+00
Wasserstein-1 (Best)	1.27E+00 ∓ 2.40E-01	8.57E-01 ∓ 1.69E-01	1.23E+01 ∓ 3.23E+00

Table 6.3: Comparison of different optimization algorithms under different skew noises for 100 data points. The results are averaged over eleven different random seeds, and the MSE values are in px. † indicates that at least a trial diverged with a NaN value.

	MSE for $\Gamma = 10, \Sigma = 5$	MSE for $\Gamma = 50, \Sigma = 4$	MSE for $\Gamma = 100, \Sigma = 20$	Avg. Execution Time/s
PS	1.99E+03† ∓ 8.84E+02	1.23E+03† ∓ 8.41E+02	2.33E+04† ∓ 8.59E+03	25.04 ∓ 2.45
COBYLA	1.31E+03 ∓ 6.75E+02	1.88E+02 ∓ 5.31E+01	7.37E+02 ∓ 6.31E+01	8.66 ∓ 2.95
BFGS	7.48E+01 ∓ 6.82E+01	5.75E+01† ∓ 9.62E+01	6.14E+01 ∓ 9.64E+00	8.69 ∓ 21.11
L-BFGS-B	5.10E+02 ∓ 6.76E+02	1.10E+01 ∓ 8.12E-01	9.14E+01 ∓ 1.14E+02	50.39 ∓ 42.85
Trust-region	6.90E+02 ∓ 2.71E+02	1.11E+01 ∓ 8.12E-01	3.66E+02 ∓ 2.40E+01	1.84 ∓ 0.16
Nelder-Mead	3.39E+01 ∓ 7.42E+00	5.91E+00 ∓ 5.88E-01	2.49E+01 ∓ 7.35E+00	32.25 ∓ 5.05
CG	1.12E+02 ∓ 1.41E+02	1.06E+01 ∓ 7.94E-01	3.50E+01 ∓ 7.07E+00	2.36 ∓ 1.92
SQP (Best)	1.27E+00 ∓ 2.40E-01	8.57E-01 ∓ 1.69E-01	1.23E+01 ∓ 3.23E+00	1.66 ∓ 0.13

Table 6.4: Comparison of the MSE $\mp\sigma$ values (in px.) of the proposed CSN filter and the existing filters under different noises obtained from eleven different random seeds. Median-N represents an online median filter with a window size of $N \geq 3$. AVG is a moving average filter with $\beta = 0.3$, which yielded the lowest MSE value in $\beta = [0, 1]$.

	MSE for $\Gamma = 10, \Sigma = 5$	MSE for $\Gamma = 50, \Sigma = 4$	MSE for $\Gamma = 100, \Sigma = 20$
Median-4	3.12E+00 ∓ 5.10E-01	2.05E+00 ∓ 3.43E-01	4.80E+01 ∓ 8.08E+00
Median-3	2.94E+00 ∓ 4.70E-01	1.90E+00 ∓ 3.04E-01	4.51E+01 ∓ 6.92E+00
AVG	1.80E+00 ∓ 3.42E-01	1.19E+00 ∓ 2.15E-01	2.57E+01 ∓ 4.42E+00
KF	2.05E+00 ∓ 3.44E-01	1.31E+00 ∓ 2.07E-01	2.42E+01 ∓ 4.14E+00
CSN (Proposed method)	1.27E+00 ∓ 2.40E-01	8.57E-01 ∓ 1.69E-01	1.23E+01 ∓ 3.23E+00

the noise properties and skewness parameters could affect these results, which should be considered during the design phase.

After observing that the SQP method yields the lowest error and the lowest computational time, different filters were compared against the SQP-CSN filter under different noise conditions. As shown in Table 6.4, the median (Eqn. (6.24)) filter has higher error values than the other filters because it starts estimation after $N \geq 3$ steps. When the skewness increases, the Kalman filter performs slightly better than the average filter. Nevertheless, the median and average filters are not traditional state estimation methods and do not consider the covariance information. On contrary, the skew filter yields the lowest error consistently because it is the most flexible method within the compared methods.

Fig. 6.2 summarizes the experimental state estimation study, where the median filter results in the largest MSE and the proposed method outperforms the other

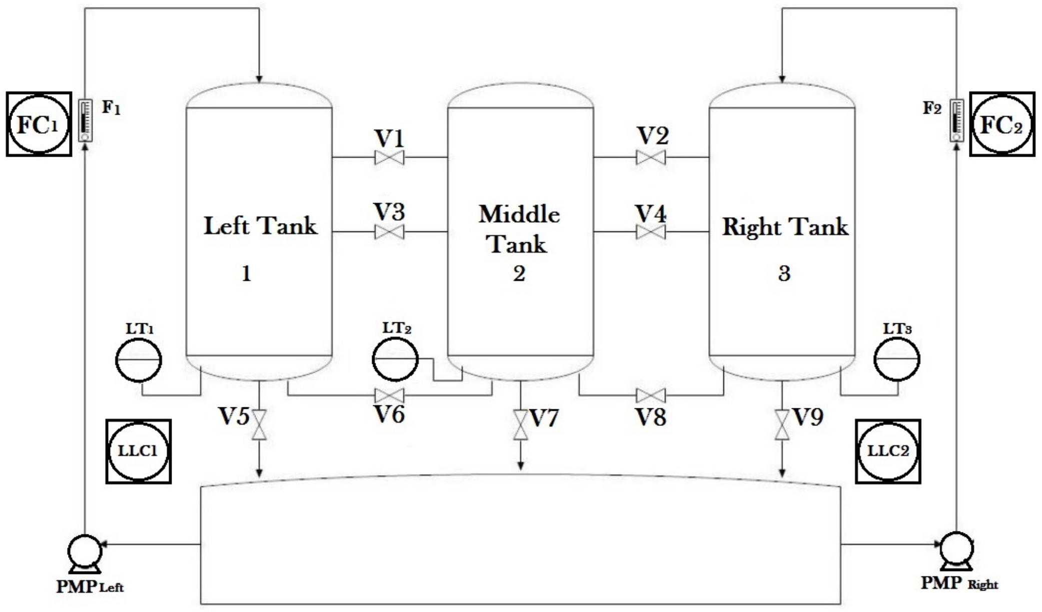


Figure 6.3: Piping and instrumentation diagram (P&ID) of the HTT system. In this study, only left tank and the corresponding instruments (left pump, V1, V3 and V5, LT_1) are used for the sake of simplicity.

methods. Overall, the SQP algorithm combined with WD consistently yielded the best solution in terms of computational time and accuracy under different noise conditions for the real time RL application. Therefore, this combination will be used in the MPC implementation in the following section.

6.4.3 Application 2: Online Level Control in the Hybrid Three Tank (HTT) System via an MPC

The hybrid tank system consists of three identical cylindrical tanks, a storage tank, two pumps, three pressure (level) sensors and nine valves, as shown in Fig. 6.3. The goal of this system is to control the level of the liquid at the desired setpoints in Tanks 1 and 3. In this study, only the left tank will be used. Since the proposed filter and the MPC are multivariate, the method can be extended to multiple tanks trivially.

Four different skew noises were synthetically introduced to the system with varying skewness and scale parameters to mimic realistic filtering problems with asymmetric noise [298, 299], $\Gamma_\psi \in \{1, 2\}$ and $\Sigma_\psi \in \{5, 10\}$. The MPC employed the predicted state values that resulted from the Kalman and the proposed skew filter.

Table 6.5: Prediction and control errors (in cm) by using the Kalman filter and the proposed filter. The standard deviation for ten experiments was found to be ∓0.03 cm.

		$\Gamma = 1$ $\Sigma = 5$	$\Gamma = 1$ $\Sigma = 10$	$\Gamma = 2$ $\Sigma = 5$	$\Gamma = 2$ $\Sigma = 10$
KF+MPC	MSE_P	12.52∓0.03	29.78∓0.03	10.28∓0.03	22.37∓0.03
	MSE_C	13.95∓0.03	27.24∓0.03	12.25∓0.03	21.75∓0.03
CSN+MPC	MSE_P	12.35∓0.03	25.68∓0.03	10.02∓0.03	22.17∓0.03
(Proposed Method)	MSE_C	13.02 ∓0.03	26.27∓0.03	11.86∓0.03	20.51∓0.03

The results are presented numerically in Table 6.5. As the table shows, the proposed method outperforms the Kalman filter in all the tested scenarios in both MSE_P and MSE_C. This happens because the additional parameters provide flexibility to the filter while the optimizers find the best CSN parameters and control actions. Because the proposed method improves the prediction first, the control performance is also improved since the MPC utilizes these improved predictions. Since the optimization method is SQP, the process can be controlled online without any computational overheads.

The experimental results showed that the proposed dimensionality reduction method can yield accurate estimation when WD is combined with SQP while avoiding the dimension increase problem. Due to the simplicity of the utilized approaches, the proposed methodology provides practical solutions to real-time estimation and control problems. Both experimental studies showed that the proposed method outperforms existing filtering techniques, which highlights the effectiveness of the method.

6.5 Conclusion

Online controllers are required to enhance process safety and provide optimal production. However, these controllers utilize noisy measurements that have been traditionally assumed to be Gaussian. Unlike this general assumption, various industrial applications may involve skewed noises that can be represented by closed skew-normal distributions, whose recursive estimations become challenging due to the dimensionality increase. Although several attempts have been made to address this issue, optimal online estimation is still an open challenge.

Inspired from the optimal transport theory, this study developed an online dimensionality reduction technique while considering the entire empirical distribution. In addition to proposing a novel methodology for online skew state estimation, this work provided both theoretical and empirical insights about the proposed method. After comparing different objective functions, it is found that a combination of WD and SQP provides the best performance in both estimation and control. Finally, the proposed scheme was implemented in two experimental setups for real-time RL and MPC applications.

Overall, the proposed method can be beneficial for reducing the dimensionality of multivariate CSN distribution for online applications. Furthermore, it can be utilized in multivariate estimation and advanced control.

Chapter 7

Reinforcement Learning Approach to Autonomous PID Tuning [*]

This chapter proposes an autonomous methodology for PID tuning based on RL theory. Many industrial processes utilize proportional-integral-derivative (PID) controllers due to their practicality and often satisfactory performance. The proper controller parameters depend highly on the operational conditions and process uncertainties. This study combines the recent developments in computer sciences and control theory to address the tuning problem. It formulates the PID tuning problem as a reinforcement learning task with constraints. The proposed scheme identifies an initial approximate step-response model and lets the agent learn dynamics off-line from the model with minimal effort. After achieving a satisfactory training performance on the model, the agent is fine-tuned on-line on the actual process to adapt to the real dynamics, thereby minimizing the training time on the real process and avoiding unnecessary wear, which can be beneficial for industrial applications. This sample-efficient method is tested and demonstrated through a pilot-scale multi-modal tank system. The performance of the method is verified through setpoint tracking and disturbance regulatory experiments. At the end of this chapter, the contextual bandit-based PID tuning methodology is extended to RL-based MPC tuning where

[*]This chapter has been published as **Dogru, O.**, Velswamy, K., Ibrahim, F., Wu, Y., Sundaramoorthy, A.S., Huang, B., Xu, S., Nixon, M. and Bell, N., 2022. Reinforcement learning approach to autonomous PID tuning. Computers & Chemical Engineering, 161, p.107760., and a portion of this chapter has been presented at the 2022 American Control Conference with a publication **Dogru, O.**, Velswamy, K., Ibrahim, F., Wu, Y., Sundaramoorthy, A.S., Huang, B., Xu, S., Nixon, M. and Bell, N., ”Reinforcement Learning Approach to Autonomous PID Tuning,” 2022 American Control Conference (ACC), 2022, pp. 2691-2696, doi: 10.23919/ACC53348.2022.9867687.

an agent adjusts the weights of the objective function of an MPC. Later, the methodology is integrated into an autonomous automation system. The experimental results will be shown in the following chapter for completeness.

7.1 Introduction

Modern industries rely on complex processes that need to be operated efficiently and environmentally friendly. Advanced control schemes, such as model predictive controllers, provide effective solutions to achieve such goals [348, 349]. However, model identification for such schemes may be challenging due to ever-changing process conditions, uncertainty and complexities. Although these controllers deliver satisfactory performance, their operation also heavily depends on the PID control performance in the lower layer of the control hierarchy, which can vary over time.

PID controllers [350] provide stable, robust and simple solutions to control problems. In their simple form, PID controllers have three parameters, which reduce the design effort and hence, make them preferable over more complicated controllers. Their simplistic nature makes them applicable to numerous real-world problems [351]. However, their parameters are process-specific and require careful and continuous tuning.

Optimality is a general term that depends on the current market demand. For example, the goal of a plant may be a function of the resource availability, and the specifications. Other factors, such as physical conditions and the age of the equipment, also determine whether these goals are achievable. In addition, some equipment (*e.g.*, fire extinguisher systems) may need to have aggressive behaviour due to safety concerns. On the other hand, highly nonlinear, reactive or noisy systems may need smoother control policies. Thus, a universal PID parameter setting does not exist. Instead, several methods have been proposed to obtain the optimal PID parameters for the process of interest [352].

The most primitive tuning method is trial-and-error. In this method, PID parameters can be arbitrarily adjusted according to the system response. However, this method requires skilled experts and may damage the equipment if not done carefully.

Similarly, *rule-based* methods [353, 354] focus on various signal properties such as the decay ratio, settling/rise times *etc.* For example, Ziegler-Nichols method [353] pushes the system to an unstable state and adjusts the parameters based on some practical rules. On contrary, *model-based* methods [355–359] use an approximate process model (*e.g.*, a first order plus time delay (FOPTD) model [360]) to adjust the aggressiveness/robustness. Though these methods may provide sufficiently good parameters, frequent tuning may still be necessary in the case of operational variations.

On-line tuning schemes have been proposed to incorporate these process variations during operation. Earlier examples include but are not limited to *continuous cycling method* [353], *relay auto tuning* [361], and *step test method* [353]. These methods excite the system from a steady state to an oscillatory one, and re-tune the parameters. However, these prolonged methodologies may not be acceptable for slow (*e.g.*, chemical) processes.

Owing to the above-mentioned attributes of the RL methodology, it has been employed in various process control and optimization problems [119, 126, 187, 198, 248–250, 338, 362–364]. Several works have reported the use of RL for PID tuning in simulated environments [365, 365–371]. However, they either assume having a complete system model, propose complicated solutions, do not consider safe training, or are only applied to simulated processes. Similarly, [372] developed an algorithm for a simulated process by using random search, which may result in frequent excitation of the system. [373] recently developed a shallow network-based scheme to learn the PID parameters more quickly but with low exploration capability due to its incremental form. Nonetheless, in many industries, the information about the system may be limited to step test data. In addition, general RL schemes may not be used in real-time industrial applications due to safety concerns. Some examples to address the safety issue include off-line learning [374], model-based [112] and expert-based [240] methods. However, they may not be sufficient when the system model is inaccurate or the data is not sufficiently diverse. As an alternative, *constrained* RL has been proposed to limit the agent's behaviour during the learning [242, 243]. This scheme introduces a Lagrangian penalty into the general RL goal and promotes safety on-line. Thus, it makes the RL solution more applicable to real-world problems.

As an extension to the existing non-constrained RL-based tuning schemes, this study utilizes an on-line constrained RL technique for safe and autonomous PI tuning. The proposed method also simplifies the problem formulation by relying on a contextual bandit approach without need to assume Markovian transitions of the state. Utilizing a simple step-response model makes the proposed method an easy-to-utilize tool for the practitioners. Unlike the existing methodologies, the proposed method utilizes an entropy-based systematic exploration scheme and is employed in an experimental setting to demonstrate its efficiency and real-time applicability. The multi-modal control tasks are used to showcase its effectiveness in the presence of setpoint-dependent nonstationarities.

For simplicity and considering many industrial PID controllers take PI form, this chapter will focus on PI tuning. The proposed approach can be extended to a full PID controller by including one additional parameter following the same procedure. As shown in Fig. 7.1, the agent observes a setpoint, changes the PI parameters and receives a reward signal depending on the performance given the new PI parameters. The agent combines multiple objective goals to provide PI tunings for any given setpoint that the users wish the PI controller to control, and explores the state space safely. The main contributions of this study are summarized as follows: Formulating the PI tuning as an on-line RL problem. Constraining the PI parameters and process variables to achieve safe operation. Modeling the system as a step-response model, and gradually learning the model plant mismatch by on-line tuning thus significantly reducing the online training time. Demonstrating the feasibility of the proposed method by using simulated and more importantly experimental case studies. Presenting the general practically through an industrial distributed control system (DCS), namely, DeltaV.

The remainder of the chapter is organized as follows: Detailed background information is provided in Section 7.2, the proposed method is described in Section 7.3, the results are discussed in Section 7.4, the proposed methodology is extended to MPC tuning in Section 7.5, and the concluding remarks are presented in Section 7.6.

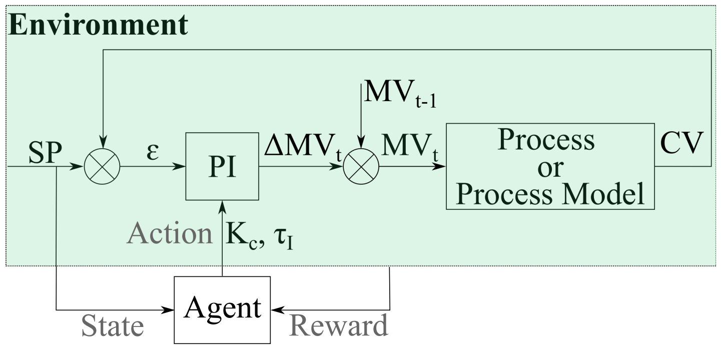


Figure 7.1: Schematic of the proposed scheme. The agent observes the setpoint and adjusts the PI parameters. It calculates the cumulative error (return) at the end of an episode and improves its policy.

7.2 Background

The goal of an RL agent is to obtain an optimal policy by means of smart trial-and-error. This involves an interaction with the environment that emits a reward during the process. In this study, an RL agent is used to tune a PI controller optimally while considering safety rules. This section explains the background of the relevant techniques in detail.

7.2.1 PID Controllers

In its simplest form, a digital PID controller in the velocity form can be written as: [352]

$$\Delta MV_t = K_c \left[(\varepsilon_t - \varepsilon_{t-1}) + \frac{\Delta t}{\tau_I}\varepsilon_t + \frac{\tau_d}{\Delta t}(\varepsilon_t - 2\varepsilon_{t-1} + \varepsilon_{t-2}) \right] \tag{7.1}$$

$$\text{with } \varepsilon = SP - CV \tag{7.2}$$

where ΔMV represents the difference of the manipulated variable (MV. That is, the controller output or process input), K_c is the proportional gain, τ_I is the time constant, τ_d is the derivative constant, Δt indicates the sampling period, and ε is the difference between a setpoint (SP) and the controlled variable (CV) respectively.

Many engineering problems have noisy sensory measurements. To avoid chattering controller outputs, which may cause unstable process behaviour, the derivative term is often ignored. This simplification also reduces the number of parameters to be tuned, which can be beneficial practically. In fact, the majority of basic industrial feedback controllers are PI controllers due to their simplicity and practicality [351]. The resulting PI controller can be written as:

$$\Delta MV_t = K_c \left[(\varepsilon_t - \varepsilon_{t-1}) + \frac{\Delta t}{\tau_I} \varepsilon_t \right] \tag{7.3}$$

7.2.2 PID Tuning

In addition to the design criteria that were discussed above, optimality of PID controllers depends on their parameters. Hence, they should be found optimally and tuned continuously with an effective tuning method to maximize the profit and safety. Earlier studies focused on various model-based, heuristic, and data-driven techniques [7, 355–359, 375–382]. Detailed reviews of additional techniques can be found in [383–389]. Nevertheless, identifying system models, ensuring optimal performance while using heuristic techniques, and achieving general solutions with data-driven techniques may be challenging and time consuming. RL can provide an alternative model-agnostic solution by learning how to tune the controller through smart trial-and-error. Furthermore, it allows the users to include state constraints to improve safety, which is crucial for industrial applications.

7.2.3 Contextual Bandits

A contextual bandit agent observes a state, $x_t \in \mathcal{X}$, of an environment at time t, and takes an action $u_t \in \mathcal{U}$, where $\mathcal{X}$ and $\mathcal{U}$ are the state space and the action space respectively [35]. During this interaction, the agent receives a *reward* signal, $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$, by using the sensory information. Note that the subscript of the reward signal indicates that the reward is obtained after taking an action. The agent's goal is to find the best policy, $\pi^*(u|x)$, that is an optimal mapping from states to actions while maximizing the reward. Note also that contextual bandits [35] simplifies the RL problem without the need to consider the Markovian state transitions.

In the presence of finite state and action spaces, the policy can be represented as a table, in which the rewards obtained for each state-action pair during the interaction are saved. However, forming a table becomes challenging for large or continuous state-action pairs due to the curse of dimensionality. An alternative approach is to parameterize the policy by θ, and optimize it directly by using the rewards. Following the policy gradient theorem [35], the policy update rule for $\pi_\theta = \pi(u|x, \theta)$ of a contextual bandit agent can be written as shown in equation (7.4).

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla \ln \pi(U_t|X_t, \theta_t) \tag{7.4}$$

where α is the learning rate, and the capital letters denote random variables. Although equation (7.4) updates θ in the direction of high reward values, convergence of θ depends on α and R. Moreover, $\theta_t \neq \theta_{t+1}$ if $R \neq 0$ or $\nabla \ln \pi \neq 0$. Inspired from the actor-critic methodology [69], a parameterized baseline, $V(x, \omega)$, can be introduced to reduce the variability in θ and speed up convergence [73]. The modified policy update rule can be defined as shown in equation (7.5).

$$\theta_{t+1} = \theta_t + \alpha(R_{t+1} - V(X_t, \omega)) \nabla \ln \pi(U_t|X_t, \theta_t) \tag{7.5}$$

At the beginning, $V(\cdot, \omega) \neq R$ since ω is often a randomly initialized parameter set [73]. Hence, large $(R_{t+1} - V(X_t, \omega))$ results in aggressive updates in the policy parameters, θ. However, when $V(\cdot)$ converges to R, variability in θ will decrease over time [73].

In addition to the abovementioned modifications to the policy gradient theorem, multiple worker-based asynchronous learning schemes can be used to reduce the learning duration significantly [73] because multiple workers interact with their own environments to allow simultaneous learning while introducing data diversity. These local workers share their knowledge with a global network asynchronously during the off-line training phase, as shown in Fig. 7.2 [323, 326, 338]. During on-line implementation, only the global network can be used. There are other techniques to improve learning such as utilizing multiple critics, target networks, delayed updates, etc. [76], which are not covered in this study.

Thus, updating the policy by using contextual bandit avoids the necessity of the Markovian state transition assumption and simplifies the learning task. In addition, using the *critic* concept of RL as a learnable baseline reduces the variance and improves convergence speed during policy learning without introducing theoretical challenges, resulting in an overall practical scheme.

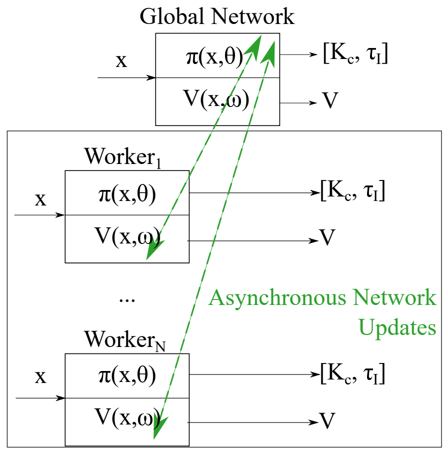


Figure 7.2: The asynchronous learning scheme that promotes data diversity and increases off-line training speed. N-workers simultaneously interact with their environment, predict the value function, and update a global network's and their parameters according to equations (7.13) and (7.14). During on-line implementation, only the global network can be used.

7.3 Constrained PI Tuning with On-line Learning

The PI tuning task can be represented as a *contextual bandit* problem [390] due to its simplicity. The goal of the PI tuning task can be defined as finding the best PI parameters by maximizing the *reward function* while respecting the constraints. The key elements for the proposed RL-based PI tuning problem can be defined as follows:
States: $x = SP$ or final CV. The agent observes the operating point of the process output as the state. Owing to the PI control, the operating point can also be considered as the setpoint of the PI control, which is given by a user. Note that the agent has the knowledge of the SP (which is equal to CV at steady state), which makes the agent adaptive to different setpoints and/or operational conditions.
Actions: $u = [K_c, \tau_I]^T$. The agent adjusts the PI parameters, K_c, τ_I, given observations, x.

The goal of the agent is to find the best PI parameters given a setpoint. In addition, several process constraints should be satisfied in order to address the safety concerns. To achieve this, a constrained value function optimization for the policy can be defined as follows:

$$\begin{aligned} v_\pi^*(x) &= \max_\pi v_\pi(x), \forall x \in \mathcal{X} \\ &s.t. \quad V_{\pi,C} \leq \xi \end{aligned} \tag{7.6}$$

where v_π is the value function for π_θ, $V_{\pi,C} = \mathbb{E}_\pi[C(x)]$ is the expected constraint C, and ξ is a penalty threshold. A solution to this problem is obtained by Lagrangian relaxation as shown in equation (7.7).

$$v_\pi^*(x) = \min_{\Lambda \geq 0} \max_\theta \left[v_{\pi_\theta}(x) - \Lambda \left(V_{\pi_\theta,C} - \xi \right) \right], \forall x \tag{7.7}$$

where Λ is a dynamic Lagrangian coefficient with an initial value of zero and θ represents the policy parameters. Note that the goal of this method is to learn a policy quickly while learning Λ slowly to guarantee convergence. The learning goal is to find a feasible saddle point with respect to Λ and θ [241]. The value function estimation with constraints can be re-defined as:

$$\begin{aligned} V_\pi(x, \Lambda) &= \mathbb{E}_\pi[G_t | X_0 = x] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} R(X_t, U_t) - \Lambda C(X_t, U_t) | X_0 = x \right] \\ &= V - \Lambda V_{\pi,C} \end{aligned} \tag{7.8}$$

Note that the constrained value function ($V_\pi(x, \Lambda)$) can be estimated by using the reward function and the constraint violations only, which makes this method convenient for data-driven constraint imposition [338]. If V ($\forall \pi$) is bounded, every local minima of $V_{\pi,C}$ is feasible, and equation (7.9) is satisfied; then, $V_\pi(x, \Lambda)$ converges to a feasible solution almost surely. Convergence analysis of the method and related proofs have been provided in [241, 391].

$$\sum_{t=0}^{\infty} \alpha_{co,t} = \sum_{t=0}^{\infty} \alpha_{a,t} = \infty, \ \sum_{t=0}^{\infty} (\alpha_{co,t}^2 + \alpha_{a,t}^2) < \infty, \frac{\alpha_{co,t}^2}{\alpha_{a,t}^2} \to 0 \tag{7.9}$$

where α_{co} and α_a are the learning rates for the Lagrangian coefficient and for the policy, respectively. Equation (7.9) shows that the constraints should be learned at a lower rate compared to the policy for the value function to converge to a feasible solution. To ensure safe *exploration* of the PI controller parameters, the constraints,

C, are imposed on the values of the PI parameters (K_c and τ_I), and on the process variables CV and MV as shown in equation (7.10).

$$C_i = \begin{cases} 0, & \text{if } o_{min} \leq CC \leq o_{max} \\ |CC - o_{min}|, & \text{if } o_{min} > CC \\ |CC - o_{max}|, & \text{if } o_{max} < CC \end{cases} \tag{7.10}$$

for $i \in \{1, 2, 3, 4\}$, with

$$o_{min} = \begin{bmatrix} K_{c,ref} - 1 \\ \tau_{I,ref} - 1 \\ 0 \\ 0 \end{bmatrix}, CC = \begin{bmatrix} K_c \\ \tau_I \\ CV \\ MV \end{bmatrix}, o_{max} = \begin{bmatrix} K_{c,ref} + 1 \\ \tau_{I,ref} + 1 \\ 1.1 \times SP \\ 1.1 \times \overline{MV} \end{bmatrix} \tag{7.11}$$

where o represents the upper and lower boundaries of exploration regions as will be discussed later in this section. $K_{c,ref}$ and $\tau_{I,ref}$ are the initial reference PI parameter values such as those calculated from certain PI tuning rules based on step-response models or values retrieved from an existing PI controller. $\overline{MV}$ is the steady state MV value at the corresponding setpoint of the output. The value 1.1 is set in Equation (7.11) for illustration purpose and it can be adjusted according to the actual constraint requirements. Similarly, the value 1 is set for the PI parameter constraints and can be adjusted.

$o(1)$ and $o(2)$ constrain the PI parameters, whereas $o(3)$ and $o(4)$ regulate CV (regarding SP) and MV (according to $\overline{MV}$) respectively. Note that these are not hard constraints but they can be tightened by tuning the corresponding weights. They regularize the agent as an addition to the reward function and can provide a safer operation [338].

Return: G_t. Considering minimization of the deviation from the setpoint and satisfying the constraints, the sum of rewards and constraints represents the return, which can be defined as:

$$G_t = -\sum_{UI} [(CV - SP)^2 + W \left[C_1, C_2, C_3, C_4\right]^T \Lambda] \tag{7.12}$$

where UI is the update interval, and W represents the weight coefficients. These values are selected empirically and can be tuned depending on the control task to

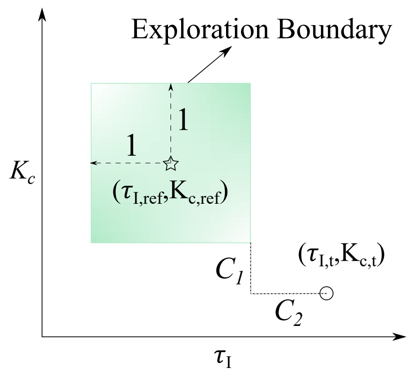


Figure 7.3: The proposed safe exploration of the PI parameters. $K_{c,ref}$ and $\tau_{I,ref}$ are the pre-determined reference PI parameters. These parameters can be obtained by using a step-response model. At any time t, the agent is penalized if the PI parameters are outside the unit exploration boundary.

obtain the desired behaviour. Although there is no state transitions in the contextual bandit setting, time step, t, is used to clearly define the parameter update rules. In an episode, the RL-level x is static, but the lower-level process variables (MV, CV) are dynamic. In this study, W will be increased during learning to provide safer operation. In equation (7.12), the first term is the integral squared error (ISE, with a fixed time step) that indicates the reward, R. The second term represents the constraints, where C_1 and C_2 keep the PI parameters within a pre-defined exploration region. This region is shown geometrically in Fig. 7.3. C_3 and C_4 constrain CV and MV respectively. Violating C results in lower returns. When converging of learning and respecting of these constraints, the final return value will consist only of ISE.

After defining the RL elements and formulating the PI tuning problem as an RL task, a major challenge is to train the agent. Data-driven models may produce general solutions, however training them may require humongous number of samples. Moreover, initial policies may not be safe or robust enough to be deployed in the real process. A solution to this challenge can be using a system model for preliminary training. However, sufficiently accurate models may not be available in practice. As an alternative to such models, less accurate step-response models can be obtained and are commonly available through a simple test, which can be used to train an initial policy. After the policy is obtained through off-line trainings, it can be further tuned on-line and improved by interacting with the real process. This methodology will

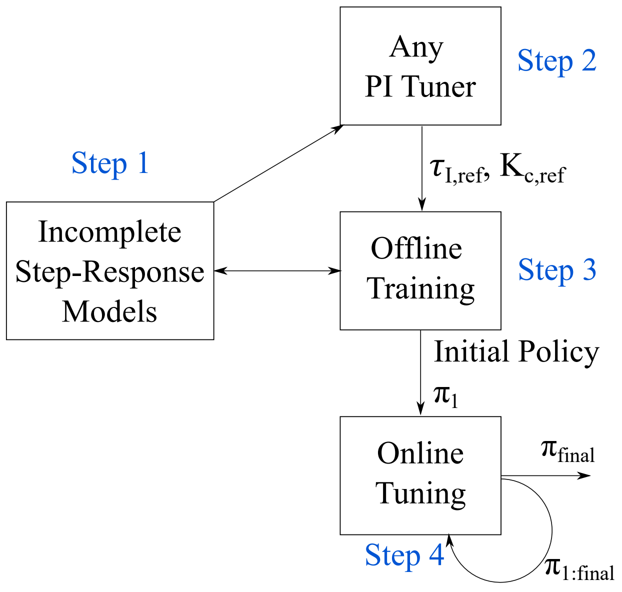


Figure 7.4: The proposed training and deployment methodology. **Step 1:** Two initial, less accurate system models are obtained by using step-response tests. **Step 2:** A PI tuning method is used to obtain the initial PI parameters. **Step 3:** Off-line training through simulation is carried out to obtain the initial policy, π_1, until it satisfies the desired properties at Step 3. At **Step 4**, π_1 can be tuned on-line to learn the real system dynamics.

reduce the training time significantly and provide effective policies without risking of damaging the equipment. The flowchart of the proposed method is illustrated in Fig. 7.4.

The proposed method consists of four steps. In the first step, depending on the nonlinearity of the process, two or more step-response models can be obtained, which may only be valid within a limited operational range. Industrial processes are typically nonlinear or multi-modal. This is why we consider two or more of the locally approximate linear step-response models or the models corresponding to two or more of the modes for the multi-modal process. These models can also be used to obtain rough knowledge (such as a nominal control action value) about the system.

In Step 2, a PI tuner or certain tuning rules can be used to obtain a reference range for the PI parameters. Alternatively, this step may be substituted by an expert's knowledge on the process for the initial PI parameters. This study determines $[K_c, \tau_I]_{ref}$ according to one of the estimated step-response models following the IMC tuning rule [352].

In the third step, the agent uses the reference PI parameters and the step-response models to train itself through the off-line simulations to achieve a satisfactory simulation performance. This step can be repeated until the agent passes the test criteria, which greatly saves subsequent on-line tuning time and prevents the potential risk of equipment damage due to extensive training duration.

After the policy meets the desired specifications in Step 3, it can be deployed on-line on the real physical system of interest. Since the proposed method is on-line, the agent can tune the PI controller further in real-time while updating itself to adapt to the real process dynamics. On-line tuning can occur at once if the real process is time invariant or continuously if the process has varying operating conditions that cause change of its dynamics. In this study, the agent will be on-line fine-tuned for multiple operating conditions to demonstrate the performance improvement effectively. To improve safety, the constraints term can be weighted more by increasing W at the beginning of this step. After the on-line tuning is completed, the agent will be capable to provide a suitable set of PI parameters for any setpoint given by users, as will be discussed in Sections 7.4.7 and 7.4.8.

Based on equations (7.5) and (7.8), and the entropy-based exploration scheme proposed in [73], the parameter update for the proposed method is given as:

$$\omega_{t+1} \leftarrow \omega_t + \alpha_c \nabla_\omega \delta^C(\cdot|\omega)^2 \tag{7.13}$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha_a \delta^C\left(\cdot|\omega\right) \nabla_\theta \ln \pi\left(\cdot, \theta\right) + \beta \pi\left(\cdot\right) \ln \pi\left(\cdot\right) \tag{7.14}$$

$$\text{with } \delta^C = G_t - V(X_t|\omega) = (R_{t+1} - WC^T\Lambda - V(X_t, \omega)) \tag{7.15}$$

$$\Lambda_{t+1} \leftarrow \max(0, \Lambda_t + \alpha_{co}(\mathbb{E}_{\pi,\Lambda_t}[C] - \xi)) \tag{7.16}$$

where δ^C is the constrained temporal difference (TD) error for a static environment, ω represents the critic parameters. $\alpha_c = 1 \times 10^{-3}$, $\alpha_a = 1 \times 10^{-4}$ and $\alpha_{co} = 1 \times 10^{-6}$ are the learning rates of the critic, policy, and the Lagrangian coefficient respectively. $\alpha_c > \alpha_a > \alpha_{co}$ is necessary for the feasibility of the proposed solution [241]. δ is the modified TD-error, which determines the direction of the parameter update. β is the entropy coefficient that adjusts the exploration extent (the randomness degree of the policy distribution). In this study, it will be decreased at the end of off-line training to improve the stability of learning. In theory, the policy ($\pi(\cdot, \theta)$) and critic ($V(\cdot, \omega)$)

can be parameterized by using various types of approximate functions [20, 35]. In this study, neural networks are utilized to capture relationships between the setpoint, value function and PI parameters that are considered to be nonlinear. As mentioned earlier, multiple workers can speed up learning; and in this study, two workers will be used during off-line training. During on-line implementation, only the global agent can be used in real control applications.

Algorithm 7.1 PI Tuning by using the proposed algorithm.

```
 1. Input $\alpha_a, \alpha_c, \alpha_{co}, T_{max}, SP$ (or $SPs$), $UI$, $[K_c, \tau_I]_{ref}$, $\beta$, $W$, $\xi$
 2. Global network parameter vectors $\omega^G$, $\theta^G$, $\Lambda^G$, and global shared time-step
    $T \leftarrow 0$
 3. Worker-specific parameter vectors $\omega^L$, $\theta^L$, $\Lambda^L$, and worker time-step $t \leftarrow 1$
 4. Initialize $\Lambda^G = 0$, Randomly initialize $\omega^G$ and $\theta^G$
 5. Initialize the PI parameters and the Environment ($Env.$) arbitrarily
      • Reset gradients $d\omega \leftarrow 0$, $d\theta \leftarrow 0$, $d\Lambda \leftarrow 0$
      • Sync. workers' parameters $\theta^L = \theta^G$, $\omega^L = \omega^G$, $\Lambda^L = \Lambda^G$  $t_{start} = t$
      • Uniformly sample a random setpoint, $SP \sim SPs$
      • Observe the state $x_t = SP$
      • Sample an action from the policy $u_t = [K_c, \tau_I]_t \sim \pi(x_t, \theta_L)$
      • Update the controller parameters $UpdateController(u_t)$
      • $k = 0$ to $UI$
          (a) $\Delta MV_k = Controller()$, Controller outputs ΔMV
          (b) $MV_k = MV_{k-1} + \Delta MV_k$, MV is updated
          (c) $Env(MV_k)$, $MV$ controls the system
      • $G_t =$ Eqn. (7.12), Receive the return.
      • Calculate, $\delta^C =$ Eqn. (7.15)
      • Accumulate policy gradients wrt. $\theta^L : d\theta \leftarrow d\theta + \nabla_{\theta^L}(\log \pi(u_t|x_t, \theta^L)\delta^C +
        \beta\pi(\cdot) \log \pi(\cdot))$
      • Accumulate critic gradients wrt. $\omega^L : d\omega \leftarrow d\omega + \nabla_{\omega^L}(\delta^C)^2$
 6. Accumulate constraint difference $d\Lambda \leftarrow -(WC^T - \xi)$
 7. Asynchronously update: $\theta^G \leftarrow \theta^G + \alpha_a d\theta$, $\Lambda^G \leftarrow \Lambda^G + \alpha_{co} d\Lambda$, and $\omega^G \leftarrow
    \omega^G + \alpha_c d\omega$.
 8. Update $t \leftarrow t + 1$, and $T \leftarrow T + 1$
 9. **until** $T > T_{max}$
10. output $\pi$
```

Table 7.1: User-defined parameters for the proposed algorithm.

Parameter	Definition	Data Type	Guide
$\alpha_c, \alpha_a, \alpha_{co}$	Learning rates $\alpha_c > \alpha_a > \alpha_{co}$.	Floating Point (Scalars)	Smaller values mean faster convergence. Recommended values: 10^{-3}, 10^{-4}, 10^{-6}, respectively.
β	Entropy coefficient.	Floating Point (Scalar)	A larger β implies more exploration.
T_{max}	Maximum number of episodes.	Integer (Scalar)	A larger T_{max} implies more training time.
UI	Number of steps in an episode.	Integer (Scalar)	Preferably close to process settling time.
o_{min}, o_{max}	Lower and upper limits for the constraints.	Floating Point (Vectors)	Given variable constraint.
W	Constraint weights.	Floating Point (Vector)	A larger W implies more penalty on the corresponding variable if violating its constraint.
ξ	Constraint threshold.	Floating Point (Scalar)	Tolerance to constraint violation.
SPs	Setpoint sets.	Floating Point (Vector)	Desired setpoint.
$[K_c, \tau_I]_{ref}$	Reference PI values.	Floating Point (Vector)	Existing PI controller parameters or initial tuning according to any PI tuning rule.

7.4 Results and Discussion

As described in the earlier sections, the proposed method identifies step response models, obtains reference PI parameters and operational knowledge, offline trains a policy using these approximate models, and online tunes the policy in the real process. Therefore, the hyperparameters listed in Algorithm 7.1 (denoted as inputs) should be defined by the user before using the proposed algorithm. A user friendly guideline about their selection is provided in Table 7.1. The proposed algorithm tunes the PI parameters while minimizing the deviation of the response from the setpoint and respecting the operational constraints. After describing the processes in the case studies, implementation details, the network, learning results, sensitivity analysis, and simulated and experimental test results under process uncertainty and disturbances are presented in this section. To demonstrate the generality and effectiveness of the proposed method, a simulated and two experimental case studies are used in this section. These systems show multi-modal behaviour according to the setpoint. The experimental studies showcase that given reference simple models, the proposed method can be used to tune PI controllers in different settings in real-time.

7.4.1 Simulated Case Study: a Parameter Varying System (PVS)

A first-order parameter-varying system, $P(s)$, is considered and shown in equation (7.17).

$$P(s) = \frac{A_1 \sin(\Omega SP) + B_1}{(A_2 \sin(\Omega SP) + B_2)s + 1} \tag{7.17}$$

where $A_1 = 8$, $A_2 = 11$, $\Omega = 0.3$, $B_1 = 2$, $B_2 = 6$. The gain and the time constant of this system are a function of the setpoint, thus varying during the experiments. These variations make the system suitable to test the proposed adaptive method. The above-mentioned values can be tuned to mimic the nonlinearity or change of the gain and the time constant.

Before off-line training *(Step 1)*: Two step-response models were developed by conducting step-response tests at two setpoints, $SP \in SPs_1 = \{1, 2\}$. Some gain mismatch was deliberately introduced to represent the inaccuracy of the identified step-response models. *(Step 2):* Then, the initial $[K_c, \tau_I]_{ref}$ were determined according to one of the step-response models following the IMC tuning rule [352].

During off-line training *(Step 3)*: A random setpoint was sampled from a uniform distribution, $SP \sim U(SPs_1)$, and the agent was trained on two distinct step-response models for the agent to be trained in the environment of varying operating conditions. These models were interpolated from the two step-response tests.

This step ensures that the agent gains sufficient experience before it is deployed in real-time for on-line tuning without risking the process.

On-line tuning *(Step 4)*: During on-line tuning, a random setpoint was sampled from another uniform distribution, $SP \sim U(SPs_2 = \{4, 5\})$, to demonstrate the adaption to the real process by the proposed scheme. The gains used in this study are geometrically shown in Fig. 7.5. In addition, the penalty coefficient, W, was increased from 1,000 to 2,000 to improve safety.

7.4.2 Experimental Study: a Multi-Modal, Nonlinear Tank System (TS)

A tank system (TS) with three modes [338] is shown in Fig. 7.6. The storage tank contains water that can be moved into Tank 1 by means of a pump. Valves V1-V2 are used to move the water into Tank 2, which changes the system dynamics according to the water level. As the level increases, the process gain decreases. This multi-modal behaviour makes the controller design more challenging. Valves V3 and V5 were kept open to avoid overflowing. Valve V4 was opened for 20 seconds to introduce disturbance during the test.

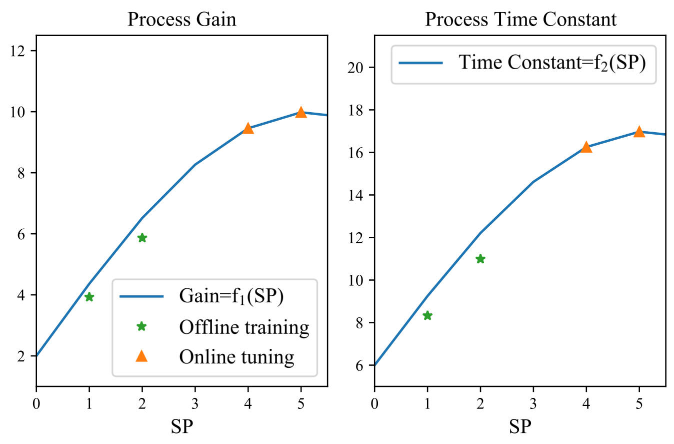


Figure 7.5: Gain and time constant of PVS that is shown in equation (7.17) with respect to setpoint. The green stars and the orange triangles indicate the gains used during off-line training and on-line tuning phases respectively. Note the deviation of the stars from the true gain and time constants reflects the model mismatch between the off-line training and on-line tuning phases.

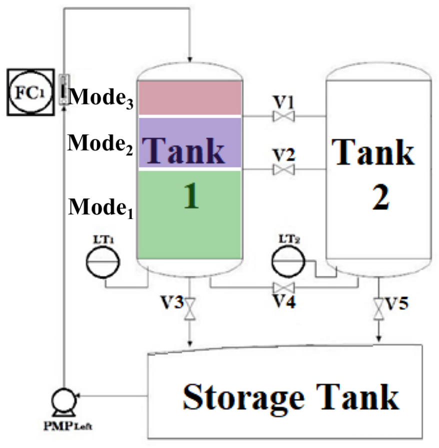


Figure 7.6: P&ID of the tank system. Storage tank contains a water that is pumped into Tank 1 by using a pump. Tank 1 has three modes due to operating valves V1 and V2 that remove the water from Tank 1 into Tank 2. Mode$_1$ $\in [0, 15.3]$cm, Mode$_2$ $\in [15.3, 30.6]$cm, and Mode$_3$ $\in [30.6, 41.3]$cm, where each mode increase reduces the process gain. Valves V3 and V5 were kept open to create a continuous process. Valve V4 was opened to introduce disturbance during the test phase.

Before off-line training *(Step 1)*: Two step-response models were developed by conducting step-response tests with the first set of setpoints ($SP \in SPs_3 = \{2, 10\}$). These tests were deliberately chosen to mimic the industrial cases where only inaccurate models that cannot represent the complete process dynamics are available through step tests. *(Step 2):* Then, $[K_c, \tau_I]_{ref}$ were determined according to one of these models following the IMC tuning rule [352]. Note that these calculated PI parameters were used to provide an initial tuning and can be obtained by using any other PI tuning rule.

During off-line training *(Step 3)*: The inaccurate step-response models (that were interpolated from the two step-response tests) were used to train the agent within the first set of setpoints ($SP \in SPs_4 = [2, 10]$) in the off-line simulations. During off-line training, the agent learns the relationship between the switching setpoints and the PI parameters.

Generally, the agent parameters of the approximated function (*e.g.*, the neural network) are randomly initialized and may not be sufficient to control the system of interest. This step, as proposed in this work, ensures the neural network parameters are improved before the real-time implementation. Additionally, this step can utilize multiple workers, saves training time, and prevents the equipment from a risk of getting damaged due to the initial random explorations of the agent.

On-line tuning *(Step 4)*: If the agent (π_1) is ready for the real-time deployment, the multiple worker scheme should be converted into a single one in the actual implementation as there is only one real environment available. Then, π_1 can be on-line tuned to learn the multi-modal behaviour as well as the uncertainties of the real physical system.

In this study, the agent was online-tuned by using setpoints, $SP \in SPs_5 = \{11, 12, 20\}$. Note that there is a valve below Mode$_2$ that reduces the process gain as shown in Fig. 7.6. Therefore, the system behaviour is setpoint-dependent (multi-modal) and the agent learns this multi-modality during on-line training. After on-line tuning the agent for about one day of real-time experimentation, its PI parameter suggestions were used in the test phase. The on-line tuning duration can differ (*e.g.*, vary from 1 to 24 hrs) depending on the complexity of the process and uncertainties

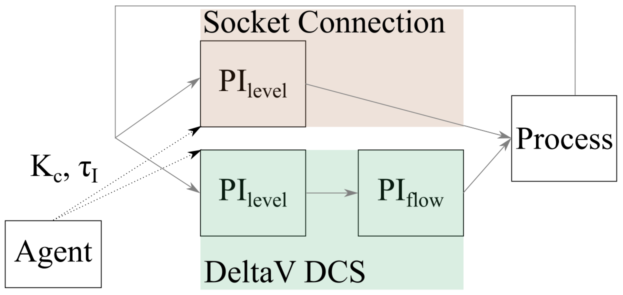


Figure 7.7: PI tuning in different configurations for level control of a process. Top: Socket connection uses a velocity form PI controller. Bottom: level controller is cascaded with a flow controller to enhance safety. The cascade controlling scheme and DeltaV implementation mimics an industrial setup. The agent in both cases tune only the level controller, and these configurations demonstrate the effectiveness of the proposed method.

as well as how much mismatch between the step-response model and the real process and how tightly the PID is to be tuned. For the sake of diversity and to demonstrate how general the proposed method is, two agents will be trained in the TS.

7.4.2.1 Agent with Socket Connection

This agent tuned a velocity-form PI controller that was connected to the experiment directly. This implementation showcases how the agent adapts to the changes in the simplest controlling scheme as shown in Fig. 7.8, where the agent directly tunes the parameters of a level PI controller parameters (PI_{level}). For simplicity, the weight coefficient (W) was kept constant during online tuning while reducing β to limit the exploration. V4 was opened for 20 seconds to introduce disturbance during the test. This implementation will be called TS-a.

7.4.2.2 Agent with DeltaV Connection

In contrast to the previous case, cascaded controllers are preferred in various industries to improve safety, avoid wearing of the equipment, reduction of disturbances *etc.* A related example is shown in Fig. 7.8, where the process is indirectly controlled by using a flow PI controller (PI_{flow}), which receives a reference signal from PI_{level}. In this case the 'optimum' PI_{level} parameters depend on those of PI_{flow}, therefore they will be different from those of TS-a for the same system. Although there is a broad literature about cascade controller tuning [392–399], this work fixes the PI_{flow}

parameters and tunes only $\mathrm{PI}_{\mathrm{level}}$. Therefore, the proposed method is still considered noncascade controller tuning. In addition, the controller can be in the position form. Since the agent is model-free, it can tune a PI controller in the position form as shown in Eqn. (7.18).

$$MV_t = \overline{MV} + K_c \left[\varepsilon_t + \frac{\Delta t}{\tau_I} \sum_{j=1}^{t} \varepsilon_j \right] \tag{7.18}$$

As will be discussed in Section 7.4.3, the RL-based algorithm is used to tune a PI controller that is implemented in an industry standard DCS known as the DeltaV system [400]. Similar to the simulated case, W will be increased from 5,000 to 10,000 while reducing β in this case study. This implementation will be called TS-b. Overall, these implementations demonstrate that the agent can tune PI controllers in various configurations, starting from a simple step-response model.

7.4.3 Implementation Details for both Simulated and Experimental Systems

Pseudocode of the proposed algorithm is provided in Algorithm 7.1[†]. As shown in Fig. 7.2, the algorithm consists of $N = 2$-workers that interact with their own environments during off-line training. The number of workers is limited by the available computational resources. At the on-line implementation phase, only a single worker can be used. These workers can be any differentiable function that can approximate complex functions. In this case study, the workers were feed-forward neural networks with their own setpoints to improve data diversity and exploration. The reference PI parameters, $[K_c, \tau_I]_{ref}$, can be obtained by using any method as described in Section 7.2.2. During off-line training, an episode consisted of $UI = 1000$ steps. The sampling time for the PI controllers of PVS and TS was one and five seconds, respectively.

After off-line training the agents in the simulated environments, the agents were on-line tuned for around a day. The on-line tuning time can vary depending on the quality of the step-response models used for the off-line training. Note that the on-line tuning was conducted on the actual process and resulted in improvements in the

[†]The source code is given in `https://github.com/oguzhan-dogru/RL_PID_Tuning`. The user-defined hyperparameters are summarized in the algorithm's caption and the README.md file of the source code.

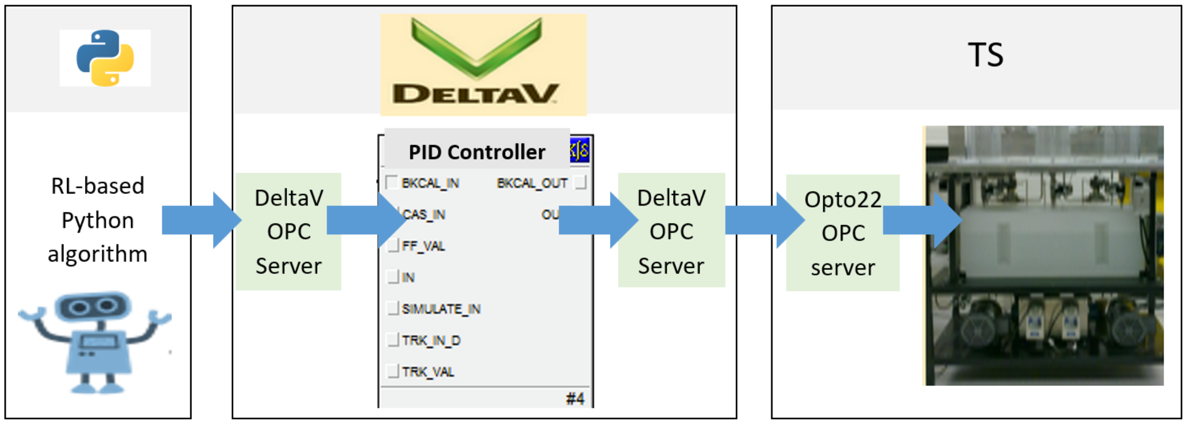


Figure 7.8: The Python RL-based algorithm tunes the DeltaV-PI parameters via the DeltaV OPC server that also enables the PI to manipulate TS-b via the Opto22 OPC server.

policy as shown in the result section.

The proposed RL-based tuning scheme is applicable to different types of industrial DCS. This study uses DeltaV as an example to illustrate the connection due to its broad applicability in the industry. The main advantages of using such DCS are that it can continuously run an entire plant at peak performance, synchronize control strategies, inputs, and outputs safely and securely while optimizing production. The proposed method can be directly plugged into the existing control schemes without major changes. The DeltaV Application station is equipped with a DeltaV Open Platform Communications (OPC) server that, on the one hand, connects the RL-based tuning algorithm to the DeltaV-PID controller and tunes its parameter via a Client-Server OPC communication protocol, as shown in Fig. 7.8. A Python package known as OpenOPC is used to connect the code to the DeltaV OPC server, and a Matrikon OPC UA Tunneller is used to bridge between the 32-bit DeltaV OPC server and the 64-bit Python code. On the other hand, the DeltaV OPC server enables the smartly tuned DeltaV-PID to manipulate the pumps of TS-b via another OPC server (Opto22). This connection is known as Server-to-Server OPC communication.

Real-time communication between the RL agent and the TS was directly established using a socket connection to an Opto22 server and to the DeltaV DCS through Open Platform Communications (OPC) servers. A computer with Intel Core i5-4590 CPU (4 CPU cores) at 3.30 GHz, 8 GB RAM at 1600 MHz under 64-bit Windows 7 operating system was used for the on-line tuning phase. The offline learning phase was conducted on a Lambda deep learning workstation with an Intel Core i9-9820X CPU (10 CPU cores) at 3.30 GHz, 64 GB DDR4 RAM at 2666 MHz under 64-bit Windows 10 operating system. More information about the experimental setup can be found in [338].

7.4.4 Details of the Agent Network

In this study, the agents consisted of deep neural networks, whose results may depend on the hyperparameters. This subsection presents the network details as well as the hyperparameters that have been used throughout the study.

Fig. 7.9 demonstrates the details of the network. The agent consisted of 300 hidden and 5 output neurons. The PI parameters were sampled from a Gaussian distribution that is parameterized by the policy outputs. The variance of this distribution decreased over time, which made the PI parameters more consistent over the time it is trained. The critic outputs a scalar that estimates the value of the setpoint (*i.e.*, $V(SP|\omega)$). To train the agent, a modified policy gradient method [73] was used. Alternative policy gradient based methods (*e.g.*, deep deterministic policy gradient [74] or proximal policy optimization [79]) can be used to obtain the PI parameters. Note that these methods may suffer from longer training time or learning instability. Because it promotes efficient learning, RMSprop [260] algorithm was used to optimize the agent's parameters (as shown in equations (7.13) and (7.14)). The above-mentioned structure and hyperparameters resulted in consistent results over a variety of trials in both PVS and TS, as will be shown in Section 7.4.6. Nonetheless, they can be adjusted further if the system of interest is significantly different.

7.4.5 Results: Learning

During the learning, the entropy coefficient, β (shown in equation (7.14)), was decreased to reduce the randomness of the policy and to improve the stability of learning. As shown in Fig. 7.10, β was reduced at the end of off-line training, which could alternatively be reduced continuously through learning.

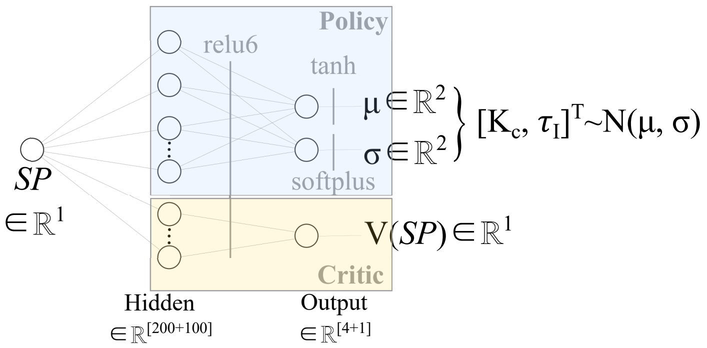
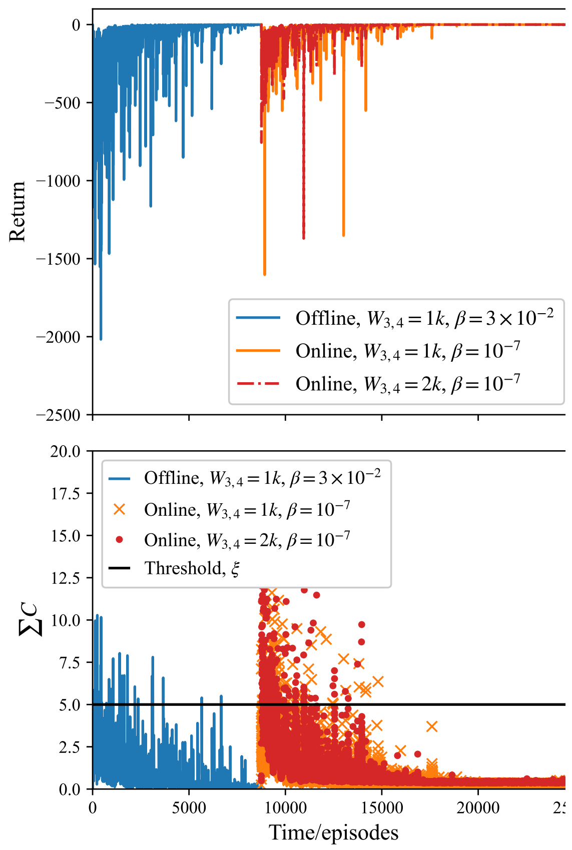
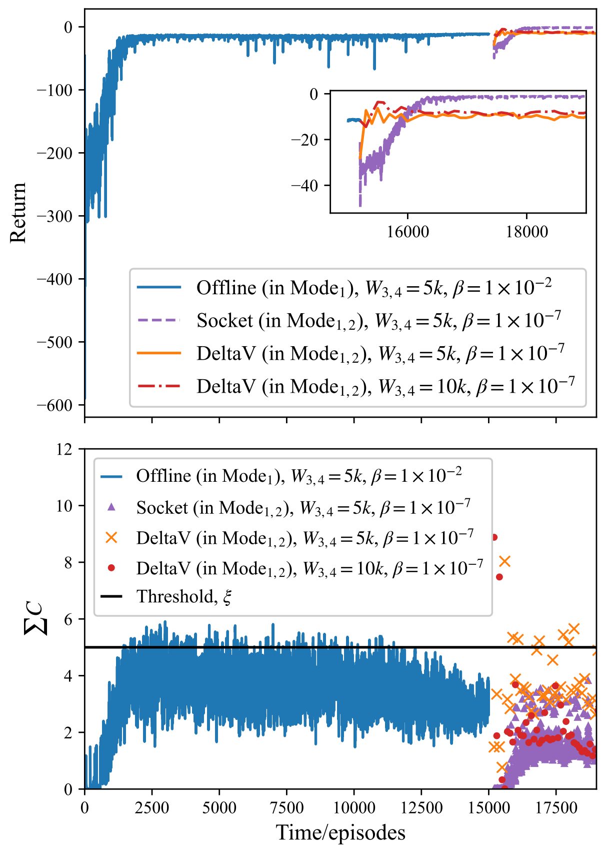

Policy
relu6
tanh
$\mu \in \mathbb{R}^2$
$\sigma \in \mathbb{R}^2$
$[K_c, \tau_I]^T \sim N(\mu, \sigma)$
softplus
$SP \in \mathbb{R}^1$
$V(SP) \in \mathbb{R}^1$
Critic
Hidden $\in \mathbb{R}^{[200+100]}$
Output $\in \mathbb{R}^{[4+1]}$

Figure 7.9: A neural network structure that represents the agent. The input (SP) is fed into the policy and critic networks that consist of 200 and 100 neurons respectively. The outputs are fed into nonlinear ReLU6 [401] activation function. The policy outputs a mean (that comes from tanh function) and a variance (that comes from softplus function). PI parameters are sampled from a Gaussian distribution, which is parameterized by using these mean and variance values. Critic, on the other hand, outputs a scalar value function with linear activation.

Table 7.2: Simulated sensitivity analysis for PVS with varying parameters. 'Result' indicates whether the related hyperparameters affected the results positively (+), negatively (-), or not significantly (NS).

Agent name	**Network update interval**	**Fixed entropy coef., β**	**Activation function (for σ)**	**Setpoint scale (SS), SP=SP/SS**	**Number of workers**	**Result**
PVS$_b$ (Baseline)	Every 10 Episodes	3×10^{-2}	softplus	10	2	
M1	10	3×10^{-2}	softplus	10	**4**	NS
M2	10	3×10^{-2}	softplus	10	**8**	NS
M3	**5**	3×10^{-2}	softplus	10	2	NS
M4	**20**	3×10^{-2}	softplus	10	2	NS
M5	10	$\mathbf{4}\times 10^{-2}$	softplus	10	2	NS
M6	10	$\mathbf{5}\times 10^{-2}$	softplus	10	2	-
M7	10	$\mathbf{1}\times 10^{-1}$	softplus	10	2	-
M8	10	3×10^{-2}	**sigmoid**	10	2	NS
M9	10	3×10^{-2}	**ReLU**	10	2	Did not learn
M10	10	$\mathbf{2}\times 10^{-2}$	softplus	10	2	+
M11	10	$\mathbf{1}\times 10^{-2}$	softplus	10	2	+
M12	10	3×10^{-2}	softplus	**5**	2	NS
M13	10	3×10^{-2}	softplus	**20**	2	NS
M14	10	$\mathbf{5}\times 10^{-3}$	softplus	10	2	+





(a) Return (top) and constraint (bottom) curves for PVS.

(b) Return (top) and constraint (bottom) curves for TS.

Figure 7.10: Learning and constraint curves for (a) PVS and (b) TS. The entropy coefficient was decreased during learning to improve the consistency of the policy. After the off-line training phase, the entropy coefficient was set to $\beta = 10^{-7}$. Then, the agent was tuned on-line to adapt to the true system dynamics. The figures show convergence for both the cases. Note that both the case studies had different set-points during the off-line training and on-line tuning phases that affected the system dynamics. This change in the dynamics caused reduction in the returns for both the cases.

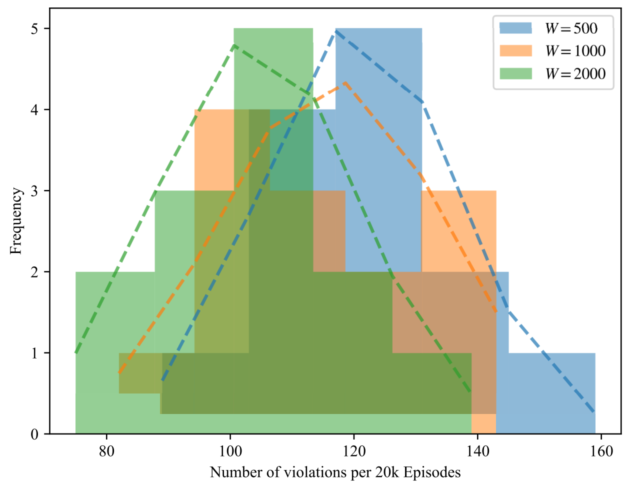


Figure 7.11: Distribution of constraint violations as a function of W over twelve trials. The mean values for the constraint violation were 120, 116 and 104 for $W = 500, 1,000$ and $2,000$ respectively. The dashed lines demonstrate the up-scaled distributions. The shift in the distribution indicates that increasing W can improve safety in terms of constraint satisfaction.

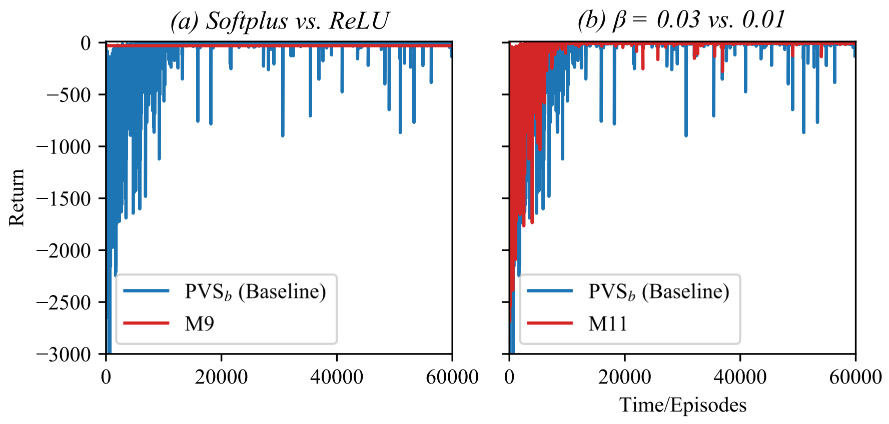


Figure 7.12: Comparisons of PVS$_b$ with (a) M9 and (b) M11. Note that the ReLU function prevented the agent from learning. On the other hand, return became more consistent as the entropy was decreased from 0.03 to 0.01. Therefore, an entropy annealing scheme was chosen during training.

7.4.5.1 Simulated Case

For the simulated system, namely, PVS, two sets of setpoints (SPs_1 and SPs_2) were used during off-line training and on-line tuning to improve the setpoint diversity. Note that the system gain and time constant were increased during on-line tuning to simulate varying system behaviour. As shown in Fig. 7.10a, off-line training ended

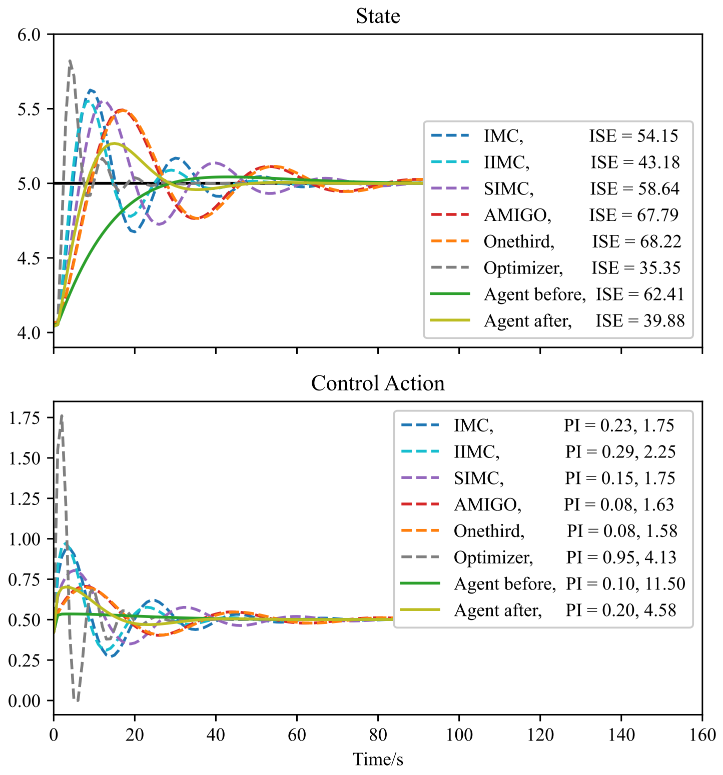


Figure 7.13: Simulated step response tests for PVS at $SP = 5$ (which was used during on-line tuning). The dashed lines indicate alternative PI tuning methods, the solid lines indicate the agent. Note that the tuning parameters other than the ones that were suggested by the on-line tuned agent were obtained at $SP = 1$. The change in the setpoint caused significant oscillations for other tuning methods whereas the agent proposed PI parameters that yielded safer and smoother transitions with lower error. Note that the optimizer results in lower ISE, however, it overshoots the SP by more than 10%, which is not desired. The agent suggested slower and more aggressive controller parameters, before and after on-line tuning respectively.

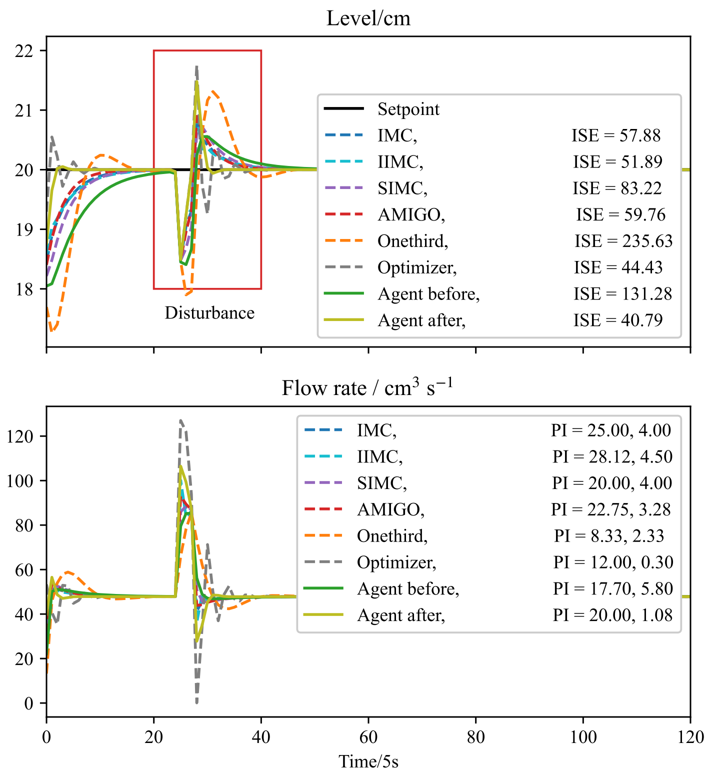


Figure 7.14: Experimental step response tests for TS-a at $SP = 20$ (which corresponds to Mode$_2$ and was used during on-line tuning). The red boxes highlight the disturbance that has been introduced by using V4. The dashed lines indicate alternative PI tuning methods, the solid lines indicate the agent. Note that when the setpoint increases, the process gain and the time constant increase. Note that the compared methods and the agent before on-line tuning resulted in slower responses. The changes in the system dynamics make the agent choose higher gains and lower time constants during on-line tuning to minimize the error. C_3 helps minimizing the ISE while C_4 preventing extremely aggressive actions at $SP = 20$. Note that the agent-tuned controllers can handle the disturbance more quickly without oscillations.

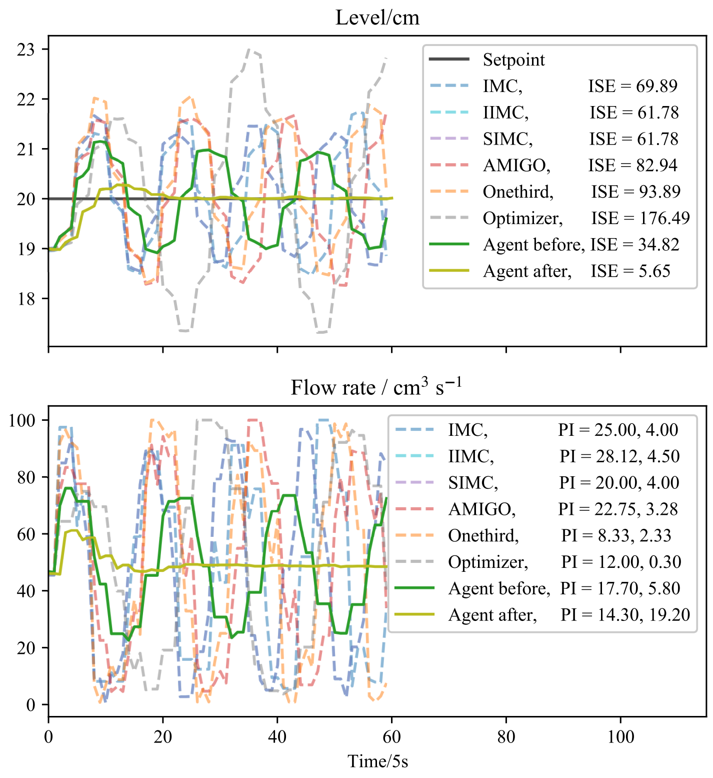


Figure 7.15: Experimental step response tests for TS-b at $SP = 20$ (which corresponds to Mode$_2$ and was used during on-line tuning). The dashed lines indicate alternative PI tuning methods, the solid lines indicate the agent. Note that when the setpoint increases, the process gain and the time constant increase. Additionally these parameters were tuned for the direct control through PI$_{\text{level}}$, which caused oscillations when PI$_{\text{flow}}$ was introduced. Owing to the constraints introduced during learning, the agent avoids oscillations after six hours of training.

at episode 8,750 with a high return value. Then, the agent started interacting with the new environment (the actual process) that initially yielded lower returns owing to the mismatch between the step-response model and the actual process. After some initial exploration, the agent converged to an optimum point. At the initial phases of on-line tuning, the agent violated the constraint threshold, ξ, since the PI parameters suggested at SPs_1 were not best fit for SPs_2. After increasing W from 1,000 to 2,000 at the beginning of the on-line tuning phase, the violations decreased from 113 to 109. Nevertheless, the violations decreased over time for both cases as the agent adapted to the new dynamics.

As shown in Fig. 7.11, the mean values for the constraint violation were 120, 116 and 104 for $W = 500$, $1,000$ and $2,000$ respectively, over twelve simulations. The results show that increasing W can provide safer operation by reducing constraint violation.

7.4.5.2 Experimental Case

For the experimental system, namely, TS, sets of setpoints SPs_4 and SPs_5 were used during off-line training and on-line tuning respectively since they correspond to different modes in the tank. As shown in Fig. 7.10b, off-line training ended at around episode 15,000 with a high return value. Then, the agent started interacting with the real environment that yielded lower returns (similar to the previous case). After realizing the changes in the system, the agent gradually improved its performance by finding better PI parameters while respecting its constraint threshold. Since the socket implementation (TS-a) and the offline training utilized the same control structure, the agent did not violate the constraints, however it started from lower return values compared to TS-b. On the other hand, the initial PI parameters resulted in the DeltaV implementation (TS-b), causing oscillations and constraint violations. After adapting to the process changes, the agent avoided oscillations, resulting in safer and more efficient control. Similar to the PVS, increasing W resulted in a decrease in the constraint violations from seven to two for $W = 5,000$ and $W = 10,000$ respectively. This is because the cascade control structure changes the optimum PI parameters. Increasing W made the agent *regret* its aggressive actions. Note that in both the cases the agent improved its performance by respecting the constraint threshold, ξ.

Overall, the results showed that annealing entropy coefficient, β, improved the convergence. Moreover, in industrial applications, a short on-line tuning time is desired. As shown in Fig. 7.10, the policies converge to their optima in around a day of on-line tuning because they had been trained on the step-response model. As such, even utilizing inaccurate models still helps the agent learn the system dynamics faster, which is an appealing solution for industrial applications in the sense of reducing the on-line tuning time.

7.4.6 Sensitivity Analysis

Properties of a stochastic policy may depend on the hyperparameters of the neural networks. An experimental sensitivity analysis was performed on PVS (at the off-line training stage) to find the optimal strategy and hyperparameters to obtain a consistent policy. After creating a baseline (PVS$_b$), some of the hyperparameters were deliberately changed. This analysis was performed prior to using the agent in the on-line phase and in TS. As shown in Table 7.2, most of the hyperparameters did not affect the consistency significantly. Using the rectified linear unit (ReLU) function for the standard deviation of the policy prevented the agent from learning at all. On the other hand, the policy was found to be sensitive to the entropy coefficient (β). In particular, lower β values resulted in more consistent policies. This is because the entropy term regularizes the exploration of the policy. When the entropy is higher, the policy tends to take more diverse actions. For the sake of brevity, only comparisons among PVS$_b$, M9 and M11 are shown graphically. As demonstrated in Fig. 7.12, changing the activation function from softplus to ReLU prevented the agent from learning due to the non-smoothness of ReLU. Moreover, the agent became more consistent over time after reducing the entropy coefficient. Note that the results may vary depending on the system of concern.

7.4.7 Simulation Case: PVS Step Test

Before and after the on-line tuning phase, a setpoint was fed into the agent to obtain its PI parameter suggestions. To demonstrate the effectiveness of the proposed method, a step test in the setpoint for the closed-loop process was conducted at $SP = 5$. The true system model was used during the test to mimic a practical scenario (note that inaccurate step-response models at SPs_1 were used during off-line training). During the tests, the step magnitude was standardized as one for the sake of comparability.

As shown in Fig. 7.13, a major result is that the agent's performance is better than the methods available in the literature [376, 377] in terms of error and step-response behaviour. This is because the compared methods provided PI parameters that were determined based on SPs_1. These PI parameters caused significant overshoots and oscillations at $SP = 5$. Since the agent was offline trained in the same model, its PI parameter suggestions before on-line tuning yielded a similar ISE value as the compared methods. However, the controller performance was improved after on-line tuning since the agent adapted to the changes. After this adaptation, the agent suggested PI parameters that resulted in lower error values while respecting its constraints. Additionally, the optimizer-based PI parameters resulted in lower ISE, but higher overshoots that are undesired. This is because the optimizer greedily minimizes the ISE without taking the soft constraints into consideration.

7.4.8 Experiment Case: TS Step Test

In the testing phase, we mimic an actual industrial application scenario in the sense that the agent has been trained as a mature tuning expert. At this phase, one only needs to specify a desired setpoint value since the agent is able to provide appropriate PI parameters without further trial and error. Thus, during this experiment, a setpoint was given to the agent to obtain the corresponding PI parameters. The procedure was repeated before and after the on-line tuning phase to showcase the adaptiveness of the proposed method. $SP = 20$ was selected for the test phase. Besides, V4 was opened to introduce disturbance to TS-a.

As shown in Fig. 7.14, the agent before on-line tuning suggested PI parameters similar to the ones suggested by the compared methods. The optimization-based tuning method resulted in two oscillations due to its aggressiveness. AMIGO and IMC methods yielded similar results, whereas the one-third method resulted in a delayed overshoot. Note that during on-line tuning (in Mode1 and Mode$_2$), the decrease in the process gain results in a slower response, which increases the error if the PI parameters are not adjusted. After on-line tuning, the agent recommended more aggressive PI parameters to reduce the error at higher setpoints while respecting the constraints. This recommendation reduced the rise time significantly when the setpoint increased. In addition, C_3 helped minimizing ISE while C_4 helped avoiding aggressive actions, providing a better overall performance. Additionally, agent-tuned controllers handled the disturbance more quickly.

In contrast, as shown in Fig. 7.15, the initial parameters suggested by the agent and other tuning methods resulted in oscillatory behaviour because they were obtained for direct control through $\mathrm{PI}_{\mathrm{level}}$. Despite this difference, the agent was able to adapt to the differences introduced by $\mathrm{PI}_{\mathrm{flow}}$ in TS-b after six hours of on-line tuning. The results emphasize the versatility of the proposed method that can adapt to changes in the control strategy starting from simple step-response tests.

As a result of the proposed constrained PI tuning scheme, the agent improved its overall performance. These results also show that the agent adapts to process changes, and thus the proposed method can be beneficial for tuning PID for multi-modal or varying dynamic processes.

7.5 MPC Tuning as a Reinforcement Learning Problem

This section extends the autonomous PID tuning to an autonomous MPC tuning methodology and integrates it into a more complex control structure in the following chapter of the thesis. In the presence of considerably accurate models, the controller can be tuned to reject disturbances, track setpoints, or have a specific transient performance [402]. However, controller tuning can include more complex aspects (such as economic or safety conditions to be satisfied) that can have nonlinear relationships between the controller parameters and the objective functions, which are used to control the system. It is challenging to mathematically model them. Such scenarios make closed-form solutions to finding optimal controller parameters impractical, which can result in poor control performance.

RL methodology provides a data-driven alternative to controller tuning and can train initial policies from simplified models. Furthermore, the agents can use non-linear, non-quadratic, economic, and/or safety-based arbitrary reward functions. Through an intelligent search of the state/action spaces and Bellman's optimality principles discussed in the previous section, the agent can look for solutions to controller tuning. The following points are assumed for the RL environment.

1. The offline environment is stochastic with Gaussian noise and linear time-invariant. This assumption mimics realistic noise and ensures the optimum set of parameters for the same system does not change over time.

2. The rewards are semi-infinite. That is, $r_t \in (-\infty, 0], \forall t$. This assumption ensures a bounded maximum value for the maximization operation shown in Eqn. (7.19).

3. The real system model is similar to but different from that of the offline model. This assumption rationalizes the RL-based MPC tuning since an offline-learned policy can be online fine-tuned in the real system. Also, it supports Assumption 1 because when Assumptions 1 and 4 hold, the set of parameters obtained from offline training will be reasonable ones to start with in the real system.

4. The state transitions follow the Markovian property. That is, the next state does not depend on the previous states given the current state and the action. This assumption makes temporal difference learning applicable without storing the previous state-action pairs, rationalizes the RL-based MPC tuning, and reduces the computational complexity.

$$J_2(\theta) = \max_{\theta} \mathbb{E}_{\pi_\theta}[G_t|\theta] \tag{7.19}$$

Given these assumptions, the MDP is defined with the following elements.

State, $x_\tau^{A2} = \Delta\overline{J} = \overline{J}_\tau - \overline{J}_{\tau-1} \in \mathbb{R}^1$: It represents the change in the MPC observed N-step cost function. That is, the low-level states and actions in the MPC cost function are substituted with their observed values as shown in Eqns. (7.20) and (7.21). $A2$ represents the agent's name to distinguish its elements (*e.g.*, state, action, reward, time) from the low-level ones.

$$\begin{aligned}\overline{J}_{\tau-1} &= \textstyle\sum_{k=0}^{N-1}\left[||\overline{x}_{t+k}-x_{t+k,sp}||^2_{Q_N^{[1]}}+||\overline{u}_{t+k}-u_{t+k,ref}||^2_{\lambda_N}+||\Delta\overline{u}_k||^2_{Q_N^{[2]}}\right] \\ &\quad +||\overline{x}_{t+N}-x_{t+N,sp}||^2_{Q_N^{[3]}} \qquad (7.20)\\ \overline{J}_{\tau} &= \textstyle\sum_{k=N}^{2N-1}\left[||\overline{x}_{t+k}-x_{t+k,sp}||^2_{Q_{2N}^{[1]}}+||\overline{u}_{t+k}-u_{t+k,ref}||^2_{\lambda_{2N}}+||\Delta\overline{u}_k||^2_{Q_{2N}^{[2]}}\right] \\ &\quad +||\overline{x}_{t+N}-x_{t+N,sp}||^2_{Q_{2N}^{[3]}} \qquad (7.21)\end{aligned}$$

where $\bar{\cdot}$ represents the observed value, and $\tau \in \{N, 2N, 3N, ...\}$ is the discrete time step of the agent. For simplicity, N is kept unchanged in this study. However, the methodology can be applied to varying N with appropriate adjustments to reflect the contribution of varying N in $\overline{J}$. Note also that the MPC weights are the final values in the horizon. This property indicates that the MPC weights are kept constant for the N steps, and the agent changes λ at the end of N steps (at $t = \{N, 2N, 3N, ...\}$) during learning.

Actions, $u_\tau^{A2} = [\lambda(1), \lambda(2)] \in \mathbb{R}^2$: Although an MPC has multiple parameters (*i.e.*, $Q^{[1]}, Q^{[2]}, Q^{[3]}, \lambda, N$, and sometimes another explicit horizon, N_C, for the control input sequence), this study focuses on tuning the λ parameter for proof of concept. Another reason for this simplification is that the Q-weights, in the MPC optimization step, adjust the relative importance in the presence of λ. Therefore, given the improvement in the MPC observed performance, the agent estimates the MPC weights, λ_τ. Since $x_{\tau+1}^{A2} = f(x_\tau^{A2}, u_\tau^{A2})$ and $x_{2N} = f(x_\tau^{A2}, x_N, u_\tau^{A2}, u_N)$, this setup satisfies the Markovian property in multiple levels and is suitable for an RL solution.

Reward, $r_{\tau+1}^{A2} \in \mathbb{R}^1_-$: Reward is the only supervisory signal that monitors the agent's performance given the state-action pair. As shown in Eqn. (7.22), the reward function determines the magnitude of the parameter update during the policy optimization step.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha(G_t - V(X_t,\omega))\nabla \ln \pi(U_t|X_t,\theta_t)\\ &= \theta_t + \alpha(R_{t+1} + \gamma V(X_{t+1},\omega) - V(X_t,\omega))\nabla \ln \pi(U_t|X_t,\theta_t) \qquad (7.22)\end{aligned}$$

In this study, the agent's goal is to produce MPC weights such that x_t and u_t (*i.e.*, the controlled and manipulated variables) are within safe limits, and the product yield is maximized, as shown in Eqn. (7.23).

$$
\begin{aligned}
-r^{A2}_{\tau+1} =& \sum_{k=N}^{2N-1} [\mathbf{1}_{\text{cond}_1} |\overline{x}_{t+k} - x_{t+k,sp}| + \mathbf{1}_{\text{cond}_2} |\overline{u}_{t+k} - u_{t+k,ref}| \\
&+ |\overline{x}_{t+k} - x_{t+k,sp} \times (1 + c_1)|] \qquad (7.23)
\end{aligned}
$$

$$
\text{with } \text{cond}_1 = (\overline{x}_{t+k} > x_{t+k,sp} \times (1 + c_1)) \vee (\overline{x}_{t+k} < x_{t+k,sp} \times (1 - c_1)) \qquad (7.24)
$$

$$
\text{and } \text{cond}_2 = (\overline{u}_{t+k} > u_{t+k,sp} \times (1 + c_2)) \vee (\overline{u}_{t+k} < u_{t+k,sp} \times (1 - c_2)) \qquad (7.25)
$$

where $c_1 = 0.1 \in \mathcal{C}$ and $c_2 = 0.999 \in \mathcal{C}$ adjust the magnitude of the limits of the safe zone/tube, and $\mathcal{C} = (0, 100]\%$.

7.6 Conclusion

Despite ever-advancing control technologies, PID controllers are an indispensable element in industrial applications due to their practicality and simplicity. Although they provide robust and stable solutions, their parameters may require frequent tuning due to uncertainties and operation condition changes. Classical methods rely on system models or various rules to tune PID controllers, where autonomous tuning is a major challenge. This study has proposed an end-to-end methodology, starting from inaccurate step-response models, which may be the only available model for many complex processes. Following the simple tuning procedure, the initial or reference PI parameters were obtained by using an off-line tuning rule to provide a baseline for the RL agent. The proposed off-line tuning step can be convenient for many industries where the on-line tuning may be costly. At the on-line tuning phase, the agent can safely improve its performance within a shorter period by interacting with the real environments that involve sensory uncertainties. Additionally, the proposed scheme has been tested successfully on a pilot-scale experiment by using industry standard connectivity schemes with a DCS in the loop, to highlight its applicability potential. Consequently, this scheme can be beneficial for industries where continuous identification of complex processes is challenging, human tunings are expensive, and on-line learning from scratch is costly or risky. This work imposes soft constraints on the variables in the reward function. One of the future research directions is a consideration of hard constraints through a direct optimization of lower-level process variables.

Chapter 8

Experimental Implementation of Autonomous Automation System.

This chapter integrates the previous chapters to propose an autonomous advanced control architecture for a pilot-scale experimental setup as a proof of concept. The architecture consists of a state estimator agent (described in Chapters 1 and 3), a model predictive controller (described in Chapter 4), two PID controllers (described in Chapter 5), an MPC tuner (described in Chapter 5), a planner/setpoint optimizer, and a plant to be controlled efficiently and safely. Such an autonomous system requires optimal sequential decision-making under uncertainty, whose components will be discussed in this section. Similar examples can be seen in the robotics sector, where various data is obtained using sensors and processed using high-performance hardware (such as graphical processing units) to calculate optimal trajectories and low-level actions in robots and autonomous cars.

8.1 Introduction

Model predictive control (MPC) is a modern technique that optimizes a constrained control trajectory over time through a dynamic system model with multiple inputs/outputs [403]. These models can be input/output, step, and impulse response models and the constraints can be enforced on both state and action spaces [404]. The controller can improve a process by using an objective function that can be a weighted mixture of low-level process variables, auxiliary terms that target economic benefits, etc. [405–407]. Moreover, special types of MPC have been proposed to address control-specific subjects, such as robustness and stability [408, 409].

Although an MPC can be used to maximize product yield [410], improve safety, and/or reduce environmental footprint, several factors limit its usability. For example, accurate system models might not be available for complex systems, the hard constraints make the optimization problem intractable (requiring numerical optimization techniques that can be time-inefficient or inaccurate), or the weight matrices in the objective function need to be tuned for desired control performance. Though several studies and practical workarounds address some of these challenges, weight tuning is an open research subject that has been investigated through numerous methodologies, including optimization [411–417], heuristics [418, 419], gradient descent with line search [420], H_∞ loop shaping [421], controller matching [422], etc.

On the other hand, reinforcement learning (RL) approaches provide an alternative solution to weight tuning by combining optimal control theory and data-driven learning [35]. In the RL methodology, an agent interacts with its environment to find the best policy, which maximizes the agent's goal. This goal can be encoded in a reward signal by experts, and the agent can optimize this policy by associating the reward signal with the state-action pairs. Unlike heuristic adaptive methods, an RL agent integrates statistical learning and Bellman's optimality principle to consider the future outcomes of the current action. A few approaches have been proposed to tune MPC through RL methodology, including a work reported in [423], which tunes the weight matrices for state and action contribution to the MPC cost. The authors have proposed to use Q-learning and showed promising closed-loop performance in a simulated environment. Moreover, the authors of [424] have used the soft actor-critic algorithm to tune the MPC prediction horizon for a control problem in a collision avoidance environment. Another study proposed to use the TD3 algorithm and showed their agent could achieve an expert-like performance in a simulated vehicle-control environment [425].

As proof of concept, this work, on the other hand, simplifies the MPC tuning problem by tuning only the action weight matrix (namely λ). The contributions of this chapter are as follows:

- Proposing a simple yet effective operating point optimizer (Agent 3) that considers economic and environmental aspects of the operation.

- Integrating the MPC tuning agent (Agent 2, developed in the previous chapter) into an autonomous control scheme that consists of an interface tracking agent (Agent 1), a Kalman filter, an operating point optimizer agent (Agent 3), an MPC, and a PID controller.
- and realizing the resulting structure in a pilot-scale experimental plant.

The remainder of this chapter is structured as follows: Section 8.2 provides detailed background information, Section 8.3 describes the proposed method, Section 8.4 discusses the empirical outcomes, Section 8.5 outlines the challenges of implementing autonomous process automation systems, and Section 8.6 concludes with remarks.

8.2 Background

This chapter is an integration of techniques developed in the previous chapters. Therefore, the background theory and methodology of state estimation and control elements are revisited in this section.

8.2.1 State Estimation and Model Predictive Control

Physical processes may be represented by a discrete-time time-invariant state-space model, as shown in Eqns. (8.1) and (8.2).

$$x_{t+1} = A_t x_t + B_t u_t + E_t \epsilon_t \tag{8.1}$$

$$y_t = C_t x_t + D_t u_t + F_t \psi_t \tag{8.2}$$

where $x_t \in \mathbb{R}^n$, $u_t \in \mathbb{R}^q$, $y_t \in \mathbb{R}^p$ are the state, control, and measurement respectively. t, ϵ_t and ψ_t, are the discrete time, state noise, and measurement noise, respectively. A_t, B_t, C_t, D_t, E_t, and F_t are gain matrices with proper dimensions. In this chapter, the following simplifications will be considered: $C_t = E_t = F_t = 1$ and $D_t = 0$.

A state estimation method can recover the unknown state of interest, x_t in the presence of noisy measurements. For example, a Kalman filter estimates the mean and covariance for the state of interest as follows [31]:

$$\hat{x}_t^- = A_{t-1}\hat{x}_{t-1}^+ + B_{t-1}u_{t-1} \tag{8.3}$$

$$P_t^- = A_{t-1}P_{t-1}^+A_{t-1}^T + \Sigma_{\epsilon,t-1} \tag{8.4}$$

$$K_t = P_t^- C_t^T (C_t P_t^- C_t^T + \Sigma_{\psi,t})^{-1} \tag{8.5}$$

$$\hat{x}_t^+ = \hat{x}_t^- + K_t(y_t - C_t\hat{x}_t^-) \tag{8.6}$$

$$P_t^+ = (I - K_tC_t)P_t^- \tag{8.7}$$

where $(\cdot)^-$ and $(\cdot)^+$ represent a priori and a posteriori estimates respectively. K_t is the Kalman gain, and $\Sigma_{\epsilon,t}$ and $\Sigma_{\psi,t}$ are the known state and measurement noise covariances.

In practice, an MPC can be used to achieve the desired process performance [315–317]. In this study, an MPC will solve the optimization problem shown in Eqns. (8.8) and (8.9).

$$\begin{aligned}\min_{\mathbf{u}_t} J(\mathbf{u}_t) = \min_{\mathbf{u}_t} \sum_{k=0}^{N-1} ||\hat{x}_{t+k} - x_{t+k,sp}||^2_{Q_k^{[1]}} + ||u_{t+k} - u_{t+k,ref}||^2_{\lambda_k} + ||\Delta u_k||^2_{Q_k^{[2]}} \\ + ||\hat{x}_{t+N} - x_{t+N,sp}||^2_{Q_N^{[3]}} \qquad (8.8)\end{aligned}$$

$$\begin{aligned}\textbf{s.t. } & \text{Eqns. } (8.1), (8.2), \\ & u_{min} \le u \le u_{max}, \forall t \\ & \hat{x}_{min} \le \hat{x} \le \hat{x}_{max}, \forall t \\ & \hat{x_0} = \mathbb{E}[\hat{x}], \hat{x} \in \mathcal{X}, u \in \mathcal{U} \qquad (8.9)\end{aligned}$$

where J is a quadratic cost function that is to be minimized at every discrete time step with $\mathbf{u}_t = \{u(0), u(1), ..., u(N-1)\}$. $\hat{y}_t = C_t\hat{x}_t$ is the estimated state. $x_{t,sp}$ and $u_{t,ref}$ are state and control action setpoints, $\Delta u = (u_{t+1} - u_t)$ is a velocity term, N is the prediction horizon, and $(\cdot)_{min}$ and $(\cdot)_{max}$ are the lower and upper limits. $Q_k^{[1]} \in \mathbb{R}^{p\times p}$, $Q_k^{[2]} \in \mathbb{R}^{q\times q}$, $Q_N^{[3]} \in \mathbb{R}^{p\times p}$ and $\lambda_k \in \mathbb{R}^{q\times q}$ are the weight matrices in diagonal form, and $\mathcal{X}$ and $\mathcal{U}$ are the state and action spaces, respectively. After the sequence, $\mathbf{u}_t$ (i.e., the solution to the open-loop optimization problem) is calculated, the first control action, $u_t = u(0) \in \mathbf{u}_t$, is sent to the plant.

8.2.2 PID Control

Although an MPC provides optimal control action to the plant, this action variable can have large variations. Therefore, it is appropriate to filter this action by using various filtering approaches. This study uses a PID controller to adjust the behaviour of u_t calculated by the MPC. That is, instead of sending u_t to the actual plant, it is sent to a PID controller, more specifically a PI controller. The resulting digital PI controller can be written as: [352]

$$MV_t = \bar{MV} + K_c \left[\varepsilon_t + \frac{\Delta t}{\tau_I} \sum_{j=1}^{t} \varepsilon_j \right] \tag{8.10}$$

$$\text{with } \varepsilon = u_t - MV_t \tag{8.11}$$

where MV_t is the control action sent to the plant, $\bar{MV}$ is a steady state bias term, K_c is the proportional gain, τ_I is the time constant, Δt indicates the sampling period, ε_t is the difference between a setpoint determined by the MPC (u_t), and the manipulated variable (MV_t) measured by using the flowmeter. The resulting MPC-PID combination is shown in Fig. 8.1.

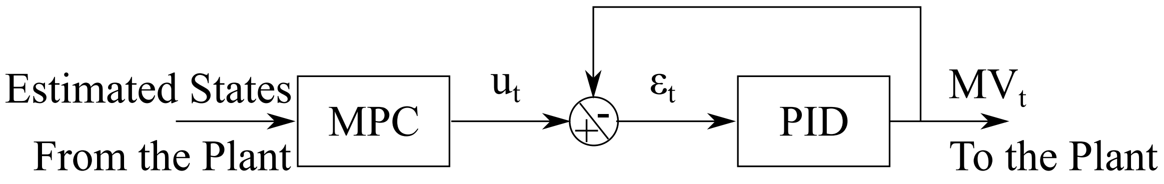


Figure 8.1: MPC-PID Combination.

Reinforcement learning modifies the control problem by substituting the controller with an agent and learning a policy instead of calculating the optimal trajectory at each time step. A Markov decision process, $\mathcal{M} = \langle \mathcal{X}, \mathcal{U}, \mathcal{R}, p, \gamma \rangle$, is a stochastic process that mathematically represents the environment to be controlled, where $\langle \cdot \rangle$ is a tuple. In this setting, the agent observes a state, $x_t \in \mathcal{X}$, and takes an action $u_t \in \mathcal{U}$. As a consequence, the system evolves according to $p(x', r|x, u)$, while emitting a *reward* signal, $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$. Model-free algorithms consider $p(\cdot)$ to be unknown to the agent, making the learning process data-driven. In the MDP, the agent's purpose is to learn how to maximize the discounted return, G, which is shown in Eqn. (8.12).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{8.12}$$

where $\gamma \in [0,1]$ is a weight that governs the contribution of future benefits to the return function, and capital letters indicate that the reward is a stochastic variable. The agent samples its actions from a stochastic policy, $\pi(u|x)$, whose performance is monitored using a value function, during its interactions with the environment. The type of value function can be chosen based on the policy evaluation approach. Learning can be accomplished by recursively solving the Bellman equations as shown in Eqns. (8.13)-(8.14) [37].

$$\begin{aligned} v_\pi(x) &= \mathbb{E}_\pi\left[G_t | X_t = x\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | X_t = x\right], \forall x \in \mathcal{X} \\ &= \sum_u \pi(u|x) \sum_{x'} \sum_r p(x', r|x, u) \times \left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} | X_{t+1} = x'\right]\right] \\ &= \sum_u \pi(u|x) \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma v_\pi(x')\right] \end{aligned} \tag{8.13}$$

$$\begin{aligned} q_\pi(x,u) &= \mathbb{E}_\pi\left[G_t | X_t = x, U_t = u\right], \forall x, u \in \mathcal{X} \times \mathcal{U} \\ &= \sum_{x'} \sum_r p(x', r|x, u) \left[r + \gamma \sum_{u'} \pi\left(u'|x'\right) q_\pi(x', u')\right] \end{aligned} \tag{8.14}$$

where $\mathbb{E}[\cdot]$ denotes the expectation of a random variable. These equations assume the Markov property, which asserts that future states rely solely on present states and are independent of past ones. Following the recursions, the optimal value functions can be found by using Eqns. (8.15) and (8.16).

$$v^*(x) = \max_\pi v_\pi(x), \forall x \in \mathcal{X} \tag{8.15}$$

$$\begin{aligned} q^*(x,u) &= \max_\pi q_\pi(x,u), \forall x, u \in \mathcal{X} \times \mathcal{U} \\ &= \mathbb{E}\left[R_{t+1} + \gamma v^*(X_{t+1}) | X_t = x, U_t = u\right] \end{aligned} \tag{8.16}$$

Finally, Eqn. (8.17) may be used to calculate the optimum (also known as greedy) policy.

$$\pi^*(x) = \arg\max_u q_\pi^*(x,u) \tag{8.17}$$

The system model, $p(\cdot)$, however, might not be available, making Eqns. (8.13) and (8.14) invalid. Moreover, in large/continuous space control problems it might not be possible to determine exact values of $v(x)$ or $q(x, u)$. To address these issues, approximate value functions, $V(x|\omega)$ or $Q(x, u|\omega)$, might take the place of exact solutions. Here, ω specifies the parameters of the value functions.

Although value functions can result in effective policies, the agent can directly learn a parameterized policy, $\pi_\theta(u|x, \theta)$, by maximizing the expected return, as shown in Eqn. (8.18).

$$J_2(\theta) = \max_\theta \mathbb{E}_{\pi_\theta}[G_t|\theta] \tag{8.18}$$

where $J_2(\theta)$ is the objective function. The policy update rule can be derived by applying the policy gradient theorem [35], as shown in Eqn. (8.19). For simplicity of notation, $\pi_\theta(\cdot)$ will be represented as $\pi(\cdot)$.

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \ln \pi(U_t|X_t, \theta_t) \tag{8.19}$$

where α is the learning rate. Although θ is updated by Eqn. (8.19) in the direction of high return values, α and G are still important for θ to converge. Moreover, $|\Delta\theta| = |\theta_{t+1} - \theta_t| \geq 0$ if $|G| \geq 0$ or $|\nabla \ln \pi| \geq 0$. Through a parameterized baseline, $V(x, \omega)$, the actor-critic technique integrates value-based methodologies and the policy gradient theorem. This combination decreases θ variability and promotes convergence. Eqn. (8.20) shows how to define the modified policy update rule.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha(G_t - V(X_t, \omega))\nabla \ln \pi(U_t|X_t, \theta_t) \\ &= \theta_t + \alpha(R_{t+1} + \gamma V(X_{t+1}, \omega) - V(X_t, \omega))\nabla \ln \pi(U_t|X_t, \theta_t)\end{aligned} \tag{8.20}$$

Initially, $V(X_t, \omega) \neq (R_{t+1} + \gamma V(X_{t+1}, \omega))$ since ω is often a set of randomly initialized parameters [73]. As a result, high values of $(R_{t+1} + \gamma V(X_{t+1}, \omega)$ will result in aggressive modifications of policy parameters, θ. However, when $V(X_t, \omega)$ approaches $(R_{t+1} + \gamma V(X_{t+1}, \omega)$, the variability in θ decreases over time [73].

8.3 Autonomous Process Automation

Autonomous process control involves sequential and complex processes that require intelligent and optimum decision-making strategies due to safety, optimality, and environmental concerns. This subsection integrates an MPC with a Kalman filter and multiple RL agents operating at different levels in the control hierarchy, as shown in Fig. 8.2.

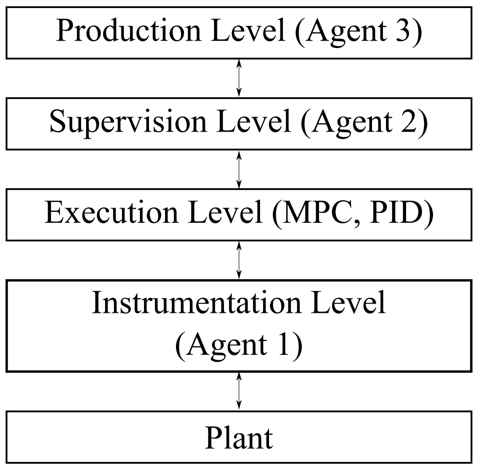

Production Level (Agent 3)
Supervision Level (Agent 2)
Execution Level (MPC, PID)
Instrumentation Level (Agent 1)
Plant

Figure 8.2: An autonomous control hierarchy with the elements used in this work. Agent 1 [326] is an interface tracking agent that estimates the state of interest, cascaded MPC and PID [426] control the state of interest, Agent 2 tunes the MPC parameters, and Agent 3 optimizes the operating point at the uppermost control level.

To provide an autonomous infrastructure, this chapter integrates the techniques developed in the previous chapters. The elements of the infrastructure include Agents 1-3, an MPC, a PID controller, and a Kalman filter. The process flow diagram (PFD) of the resulting infrastructure is shown in Fig. 8.3.

8.3.1 Agent 1

Agent 1 is an interface tracking agent that observes individual images and moves a $N_B \times M_B$-sized virtual box, B, the center of which represents the interface, $x(2)$ [326]. Its MDP is defined as follows:

States, $x_t^{A1} \in B = \mathbb{R}_+^{N_B \times M_B}$: Intensity of pixels within the virtual box, with $N_B = 42$ and $M_B = 157$.

Actions, $u_t^{A1} \in \{-1, 0, 1\}$ px.: Move the virtual box's center by one pixel up or down, or stop.

Reward, $r_{t+1}^{A1} \in \mathbb{R}_-$: The disparity in the measurement of the DP cell and the position of the center of the box (with reference to the bottom of the PSV), given in

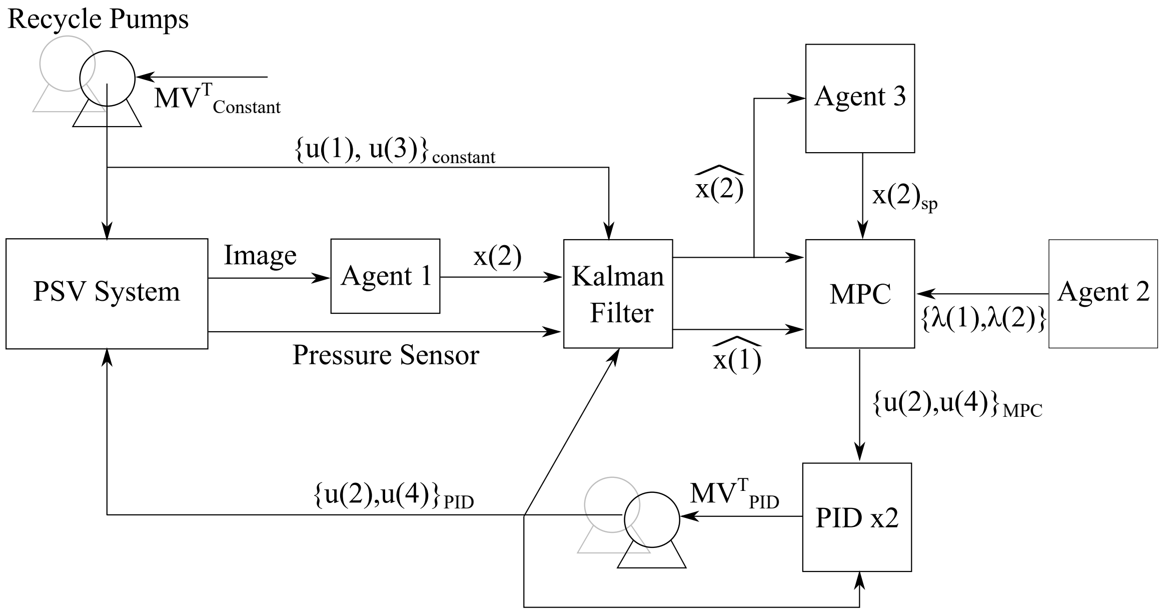


Figure 8.3: A simplified PFD of the autonomous control infrastructure. The autonomy refers to safe and optimum control of the two states, $x(1)$(top level) and $x(2)$(interface level). Agent 1 receives a PSV image and predicts the interface level, $x(2)$. A pressure sensor monitors the top level, $x(1)$. A Kalman filter uses a model to filter these estimations, and sends its predictions, $x(\hat{1})$ and $x(\hat{2})$ to the MPC. Agent 2 observes the MPC performance and tunes the MPC parameters, $\lambda(1)$ and $\lambda(2)$. Agent 3 observes the interface and estimates its operating point, $x(2)_{sp}$ to maximize the recovery rate. The PID controllers receive the MPC output and adjust the control input to provide smoother transitions. The resulting outputs, $\{u(1)_{constant}, u(2)_{PID}, u(3)_{constant}, u(4)_{PID}$ are sent to the pilot-scale experimental setup and the Kalman filter.

Eqn. (8.21).

$$r^{A1}_{t+1} = -|x(2)_t - \widehat{x(2)}_t| \tag{8.21}$$

$x(2)_t$ is the actual interface level and $\widehat{x(2)}_t$ is the center of the box at t.

8.3.2 Agent 2

Agent 2 tunes the MPC controller by observing a performance metric and changing the MPC control action weight. Its MDP has the following elements:

State, $x^{A2}_\tau = \Delta\overline{J} = \overline{J}_\tau - \overline{J}_{\tau-1} \in \mathbb{R}^1$: It represents the change in the MPC observed N-step cost function. That is, the low-level states and actions in the MPC cost function are substituted with their observed values as shown in Eqns. (8.22) and (8.23). $A2$ represents the agent's name to distinguish its elements (*e.g.*, state, action, reward, time) from the low-level ones.

$$\overline{J}_{\tau-1} = \sum_{k=0}^{N-1}\left[||\overline{x}_{t+k} - x_{t+k,sp}||^2_{Q^{[1]}_N} + ||\overline{u}_{t+k} - u_{t+k,ref}||^2_{\lambda_N} + ||\Delta\overline{u}_k||^2_{Q^{[2]}_N}\right] + ||\overline{x}_{t+N} - x_{t+N,sp}||^2_{Q^{[3]}_N} \tag{8.22}$$

$$\overline{J}_{\tau} = \sum_{k=N}^{2N-1}\left[||\overline{x}_{t+k} - x_{t+k,sp}||^2_{Q^{[1]}_{2N}} + ||\overline{u}_{t+k} - u_{t+k,ref}||^2_{\lambda_{2N}} + ||\Delta\overline{u}_k||^2_{Q^{[2]}_{2N}}\right] + ||\overline{x}_{t+N} - x_{t+N,sp}||^2_{Q^{[3]}_{2N}} \tag{8.23}$$

where $\bar{\cdot}$ represents the observed value, and $\tau \in \{N, 2N, 3N, ...\}$ is the discrete time step of the agent. For simplicity, N is kept unchanged in this study. However, the methodology can be applied to varying N with appropriate adjustments to reflect the contribution of varying N in $\overline{J}$. Note also that the MPC weights are the final values in the horizon. This property indicates that the MPC weights are kept constant for the N steps, and the agent changes λ at the end of N steps (at $t = \{N, 2N, 3N, ...\}$) during learning.

Actions, $u^{A2}_{\tau} = [\lambda(1), \lambda(2)] \in \mathbb{R}^2$: Although an MPC has multiple parameters (*i.e.*, $Q^{[1]}, Q^{[2]}, Q^{[3]}, \lambda, N$, and sometimes another explicit horizon, N_C, for the control input sequence), this study focuses on tuning the λ parameter for proof of concept. Another reason for this simplification is that the Q-weights, in the MPC optimization step, adjust the relative importance in the presence of λ. Therefore, given the improvement in the MPC observed performance, the agent estimates the MPC weights, λ_{τ}. Since $x^{A2}_{\tau+1} = f(x^{A2}_{\tau}, u^{A2}_{\tau})$ and $x_{2N} = f(x^{A2}_{\tau}, x_N, u^{A2}_{\tau}, u_N)$, this setup satisfies the Markovian property in multiple levels and is suitable for an RL solution.

Reward, $r^{A2}_{\tau+1} \in \mathbb{R}^1_-$: Reward is the only supervisory signal that monitors the agent's performance given the state-action pair. As shown in Eqn. (8.24), the reward function determines the magnitude of the parameter update during the policy optimization step.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha(G_t - V(X_t, \omega))\nabla \ln \pi(U_t|X_t, \theta_t) \\ &= \theta_t + \alpha(R_{t+1} + \gamma V(X_{t+1}, \omega) - V(X_t, \omega))\nabla \ln \pi(U_t|X_t, \theta_t)\end{aligned} \tag{8.24}$$

In this study, the agent's goal is to produce MPC weights such that x_t and u_t (*i.e.*, the controlled and manipulated variables) are within safe limits, and the product yield is maximized, as shown in Eqn. (8.25).

$$\begin{aligned}
-r^{A2}_{\tau+1} &= \sum_{k=N}^{2N-1} [\mathbf{1}_{\text{cond}_1}|\overline{x}_{t+k} - x_{t+k,sp}| + \mathbf{1}_{\text{cond}_2}|\overline{u}_{t+k} - u_{t+k,ref}| \\
&+ \ |\overline{x}_{t+k} - x_{t+k,sp} \times (1+c_1)|] \qquad (8.25) \\
\text{with cond}_1 &= (\overline{x}_{t+k} > x_{t+k,sp} \times (1+c_1)) \vee (\overline{x}_{t+k} < x_{t+k,sp} \times (1-c_1)) \quad (8.26) \\
\text{and cond}_2 &= (\overline{u}_{t+k} > u_{t+k,sp} \times (1+c_2)) \vee (\overline{u}_{t+k} < u_{t+k,sp} \times (1-c_2)) \quad (8.27)
\end{aligned}$$

where $c_1 = 0.1 \in \mathcal{C}$ and $c_2 = 0.999 \in \mathcal{C}$ adjust the magnitude of the limits of the safe zone/tube, and $\mathcal{C} = (0, 100]\%$.

8.3.3 Agent 3

Agent 3 optimizes the operating region for the interface by observing the interface (estimated by Agent 1) and changing the operating point for the MPC. Its methodology is inspired by [126] and MDP is defined as follows:

State, $x^{A3}_{\tau\tau} = x(2)_{\tau\tau} \in \mathbb{R}^1_+$: Interface level location, estimated by Agent 1. In this study, it is assumed that the interface level location will converge to the setpoint and reach a steady state. Since the controllers change this location during the operation, the location is a dynamic term, and this state respects the Markov property.

Actions, $u^{A3}_{\tau\tau} \in \mathbb{R}^1$: Change in the operating point, $I_{\tau_\tau} = x(2)_{\tau\tau,sp} = x(2)_{\tau\tau-1,sp} + u^{A3}_{\tau\tau}$.

Reward, $r^{A3}_{\tau\tau+1} \in [0, 100]\%$: The reward is a function of environmental footprint and recovery rate, which are often the main aspects during primary separation vessel (PSV) operation in the oil-sands industry [326]. More specifically, if the interface level is too high (with respect to an oil/bitumen outlet), the water and solids mix into the high-value bitumen, lowering the product quality and risking the downstream equipment. On the other hand, if the level is too low, the bitumen can mix into the tailings stream and result in environmental damage. Therefore, there is a *safe and optimum* operation range in the PSV operation, as shown in Fig. 8.4.

The resulting reward function for the PSV experiment is given in Eqn. (8.28).

$$r^{A3}_{\tau\tau+1} = \begin{cases} x(2)_{\tau\tau} \times -14.286 + 857.14 & \text{if } 60 > x(2)_{\tau\tau} \geq 55 \\ 100 & \text{if } 53 > x(2)_{\tau\tau} > 42 \\ x(2)_{\tau\tau} \times 8.33 - 250 & \text{if } 42 \geq x(2)_{\tau\tau} \geq 30 \\ 0 & \text{otherwise} \end{cases} \qquad (8.28)$$

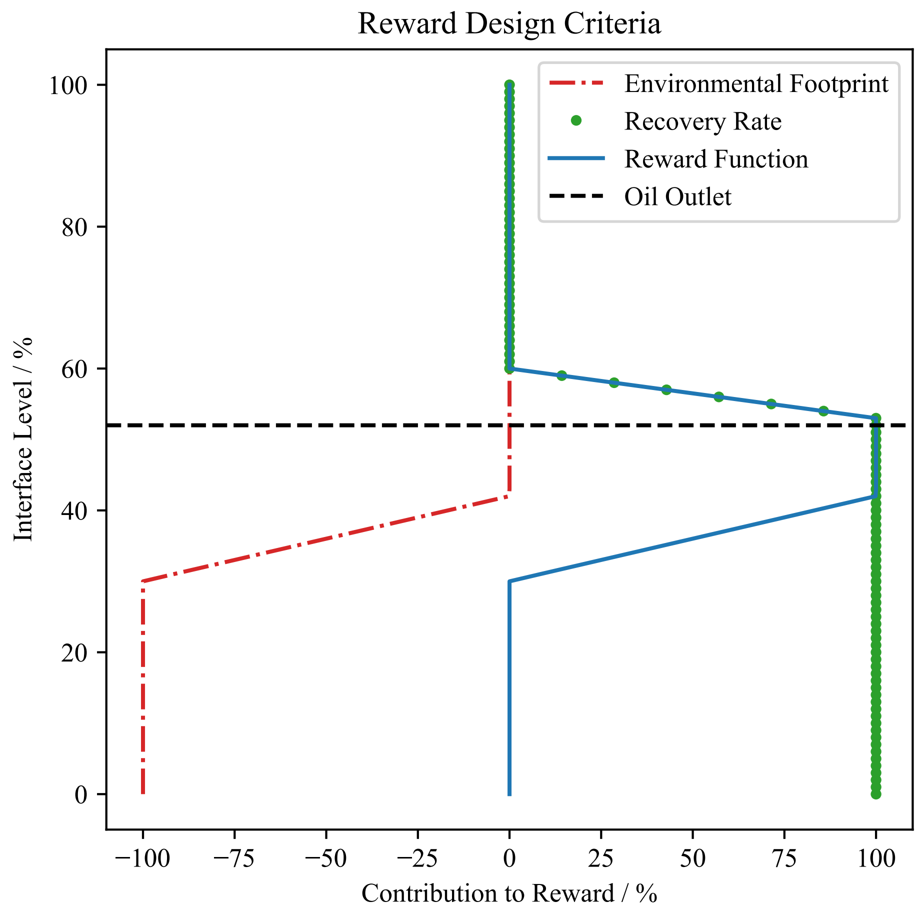


Figure 8.4: Reward Design for Agent 3. If the level is too high (with respect to a fixed oil/bitumen outlet, shown as the black dashed line), the product quality decreases and the water-solid mixture can potentially damage the equipment, which is indicated as zero contribution to the reward function. If the level is too low, the bitumen can mix into the tailings stream, damaging the environment, which is indicated as minus a hundred contribution to the reward function.

8.4 Results and Discussion

This section presents the agents' offline and online learning curves that demonstrate the agents' adaptability and experimental control results in the integrated infrastructure. Note that these agents interact with the environment at different frequencies. That is, $\frac{1}{t} = \frac{1}{0.5}s^{-1}$, $\frac{1}{\tau} = \frac{1}{1.5}s^{-1}$ (during offline training), $\frac{1}{\tau} = \frac{1}{400}s^{-1}$ (and once after online tuning), $\frac{1}{\tau\tau} = \frac{1}{400}s^{-1}$.

8.4.1 Learning Results

Agent 1 was trained on an experimental video clip with a random and continuous trajectory to learn the interface dynamics from the visual data, as described in [326]. Then, the agent was fine-tuned online with synthetic occlusion and lighting changes for visual robustness.

As shown in Fig. 8.5, during offline training, the agent converged to an effective policy that resulted in near-zero return values following initial exploration. The offset was due to the initial location of the virtual box. After offline training, the agent was exposed to synthetic occlusion and lighting abnormalities that initially deteriorated its performance. However, the agent quickly adapted to these visual changes and converged to high rewards again. This behaviour shows that the agent can quickly learn effective policies and adapt to further changes as an intelligent decision-maker.

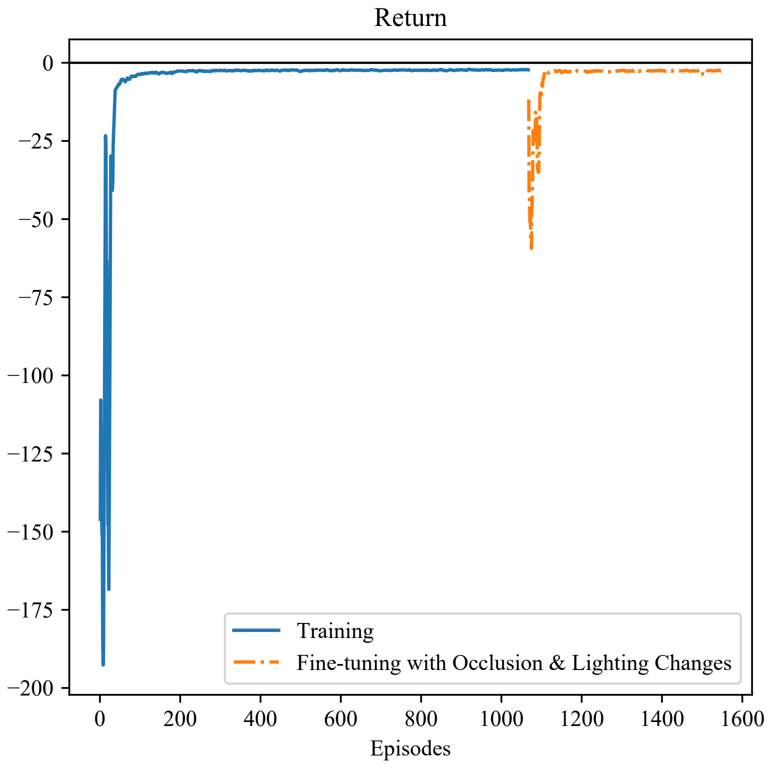


Figure 8.5: The learning curve of Agent 1. The agent converges to an optimum point after around 170 episodes. At around the 1000th episode, the agent yields poorer performance due to synthetic occlusion and lighting abnormalities. After around 50 episodes, the agent converges to higher returns.

Agent 2 was trained on a first principles state-space model with synthetic Gaussian noise that mimics the pilot scale experimental setup. To demonstrate the adaptation from the simulated to the real environment, a model-plant mismatch was deliberately introduced.

In the PSV experiment, there are two low-level states, namely the interface level and the top level. The primary goal is to maintain the interface level at the desired region as shown in Fig. 8.4. Moreover, the top level should also be at safe levels to avoid overflows, which makes this control a secondary objective. The experimental setup has four controlled pumps, four flow rates attached to their pipes, and two pressure sensors (DP cells). In this study, only the bottom DP cell is used to correct the top-level estimation. For simplicity, the oil and water feed rates are fixed at 50%. The resulting experimental setup is graphically shown in Fig. 8.6 and mathematically represented in Eqn. (8.29).

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ \begin{bmatrix} x(1) \\ x(2) \end{bmatrix}_{t+1} &= \begin{bmatrix} x(1) \\ x(2) \end{bmatrix}_t + \begin{bmatrix} \chi & -\chi & \chi & -\chi \\ 0 & 0 & \chi & -\chi \end{bmatrix} \begin{bmatrix} u(1) \\ u(2) \\ u(3) \\ u(4) \end{bmatrix}_t \end{aligned} \tag{8.29}$$

$$\begin{aligned} y_t &= x_t \\ \begin{bmatrix} y(1) \\ y(2) \end{bmatrix}_t &= \begin{bmatrix} x(1) \\ x(2) \end{bmatrix}_t \end{aligned} \tag{8.30}$$

where $x(1)$ is the top level, y_{tot} and $y(2)$ are measured by using the bottom DP cell and the camera respectively, and the details of u and the overall control structure are shown explicitly in Figs. 8.3 and 8.6. Since the system does not evolve when the pumps do not work, $A = I$. Moreover, $\chi = \frac{50T_s}{3A_{tank}}$ is identified through system identification and experimental validations [29] where T_s is the sampling time, and $A_{tank} = 706.86$ is the circular area of the tank .

Although the real process has four pumps, resulting in a four-input-four-output equation set, only the two pumps are controlled via MPC and PID in practice to mimic an industrial setup. The training phase of Agent 2 involves computationally expensive calculations, including the combination of a Kalman Filter and MPC. To speed up the training phase of the agent, the simulated state-space model was sim-

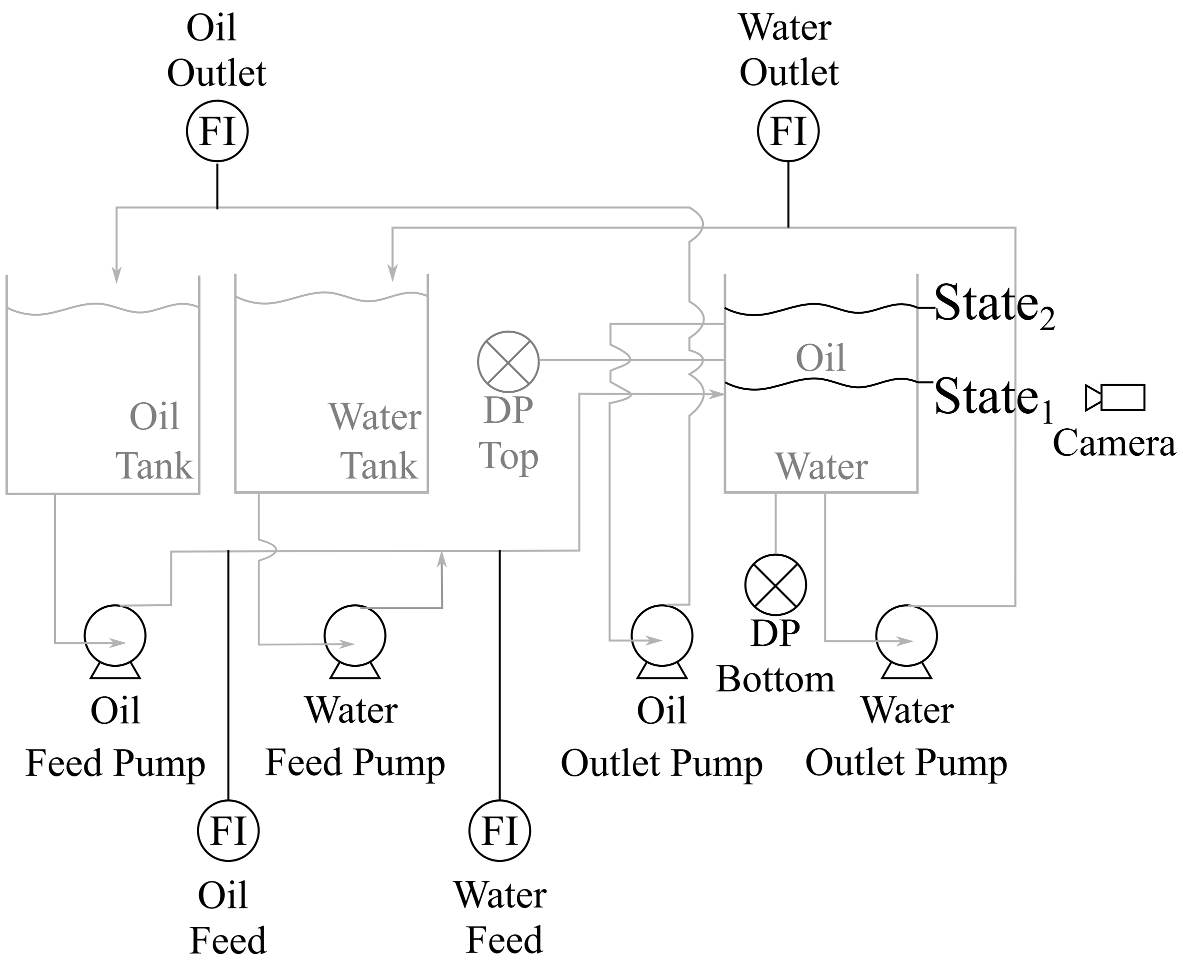


Figure 8.6: PSV Experimental Setup. The used equipment is highlighted using black colour. For simplicity, oil and water feed rates ($u(1)$ and $u(3)$) are fixed to 50%. The bottom DP cell and the camera provide the measurements for the top and interface levels respectively. The MPC and PID control the water and oil outlet pumps.

plified to a two-input-two-output system. Furthermore, to show the effectiveness and adaptive nature of the RL methodology, a model-plant mismatch was introduced by deliberately modifying the A and B matrices (as shown in Eqn. (8.31)) before offline training.

$$\begin{bmatrix} x(1) \\ x(2) \end{bmatrix}_{t+1} = \begin{bmatrix} 0 & 1 \\ -0.25 & 0.5 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix}_{t} + \begin{bmatrix} 0.8 & 0 \\ 0 & 0.4 \end{bmatrix} \begin{bmatrix} u(2) \\ u(4) \end{bmatrix}_{t} \tag{8.31}$$

After the prior training, the agent was online fine-tuned in the experimental setup to learn the model-plant mismatch. As shown in Fig. 8.7, the agent conducted initial exploration until around the 17500th episode and then converged to a maximum return with a mean value of -800. Then, the agent was further trained on the experimental setup for 264 episodes, which yielded near-zero returns, indicating no tube violations. This is because the simulated uncertainty was greater than the experimental uncertainty, which resulted in less frequent tube violations.

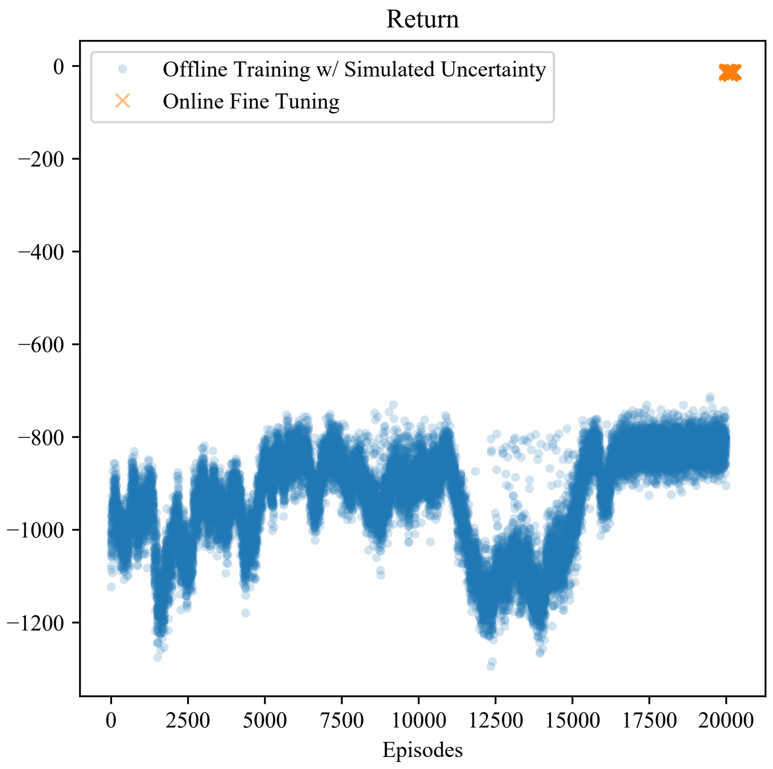


Figure 8.7: The learning curve of Agent 2. Following an initial exploration, the agent converged to an optimum point after around 17500 episodes. After 264 episodes of online tuning in the experimental setup, the agent showed no significant violations, resulting in near-zero returns values.

Agent 3 aims to optimize a setpoint while maintaining it within a safe and optimal region. At a high level, this RL problem can be simplified to manipulating a random variable to keep it within a region while observing the reward, shown in Eqn. (8.28). To speed up the training of this agent while providing a broad range of experience, this agent was trained on a simulated random walk model that represents the interface movement as shown in Eqn. (8.32).

$$x(2)_{\tau\tau,sp} = x(2)_{\tau\tau-1,sp} + u^{A3}_{\tau\tau} \tag{8.32}$$

where $u^{A3}_{\tau\tau} \in [-5, 5]$cm to prevent aggressive actions (large changes in the setpoint). During this agent's training phase, $x(2)_{\tau\tau,sp}$ was initialized at uniformly random points outside the safe region to encourage exploration. That is, the agent was forced to start at setpoints that resulted in low-reward values and improve the reward by moving into the safe and optimal region.

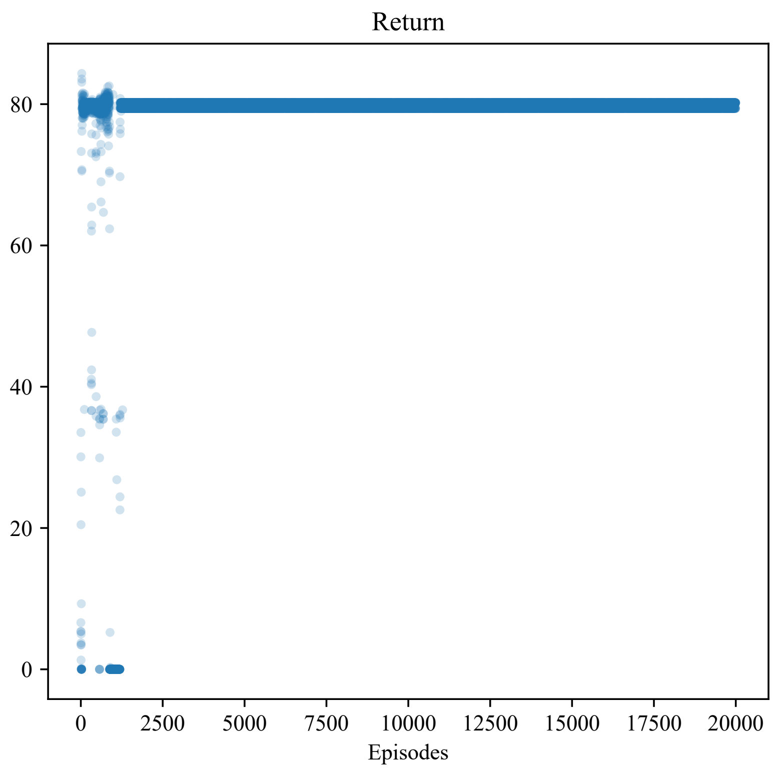


Figure 8.8: The learning curve of Agent 3. The agent started exploring the state-action spaces until around the 1500th episode. Later, it converged to higher returns and yielded high recovery rates.

As shown in Fig. 8.8, this agent initially improved its performance but could not maintain it. Following further exploration until the 1500th episode, the agent found better operating points, resulting in higher recovery rates. These results show that a simplified state-space model can act as a quicker learning environment in terms of both wall clock time and the number of samples/episodes compared to larger state-space models or more complicated models.

8.4.2 Experimental Control Results

After observing convergence in all the agents' learning curves, they were integrated and utilized in the experimental setup to demonstrate closed-loop autonomous performance.

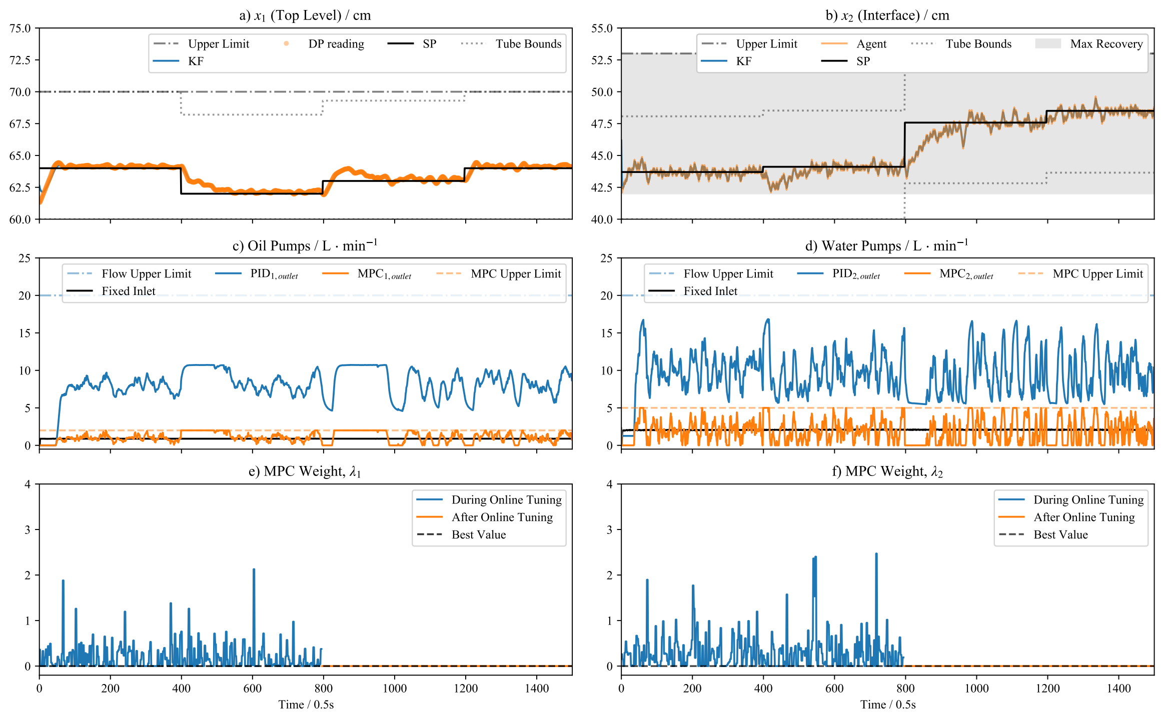


Figure 8.9: Integrated control results with the agents and the controllers. a) The Kalman filter accurately estimated the top level, and the MPC-PID combination resulted in insignificant deviations from the user-defined setpoints. b) The agent accurately estimated the interface by using the images, and the Kalman filter slightly reduced its variance. Agent 3 kept the interface within the maximum recovery range, and the tube constraints were never violated by the MPC-PID combination. c-d) The MPC and PID never violated the related flow rate boundaries. e-f) Agent 2 performed initial exploration in the real system and learned that the low MPC weights yielded better performance. At around the 800th time step, the agent output two average MPC weights for the system.

As shown in Fig. 8.9-**a)** The Kalman filter successfully estimated the top level, and the MPC-PID combination caused minor deviations from the user-defined setpoints. **b)** Using the images obtained from the PSV system, Agent 1 predicted the interface with small deviations, and the Kalman filter marginally reduced its variance. Agent 3 kept the interface within the maximum recovery range, and the MPC-PID combination never exceeded the tube constraints. **c-d)** The MPC and PID never surpassed the flow rate boundaries. **e-f)** Agent 2 conducted preliminary exploration in the real system and discovered that lower MPC weights resulted in better performance. The agent generated two average MPC weights for the system around the

800th time instant, and those weights were used after that instance.

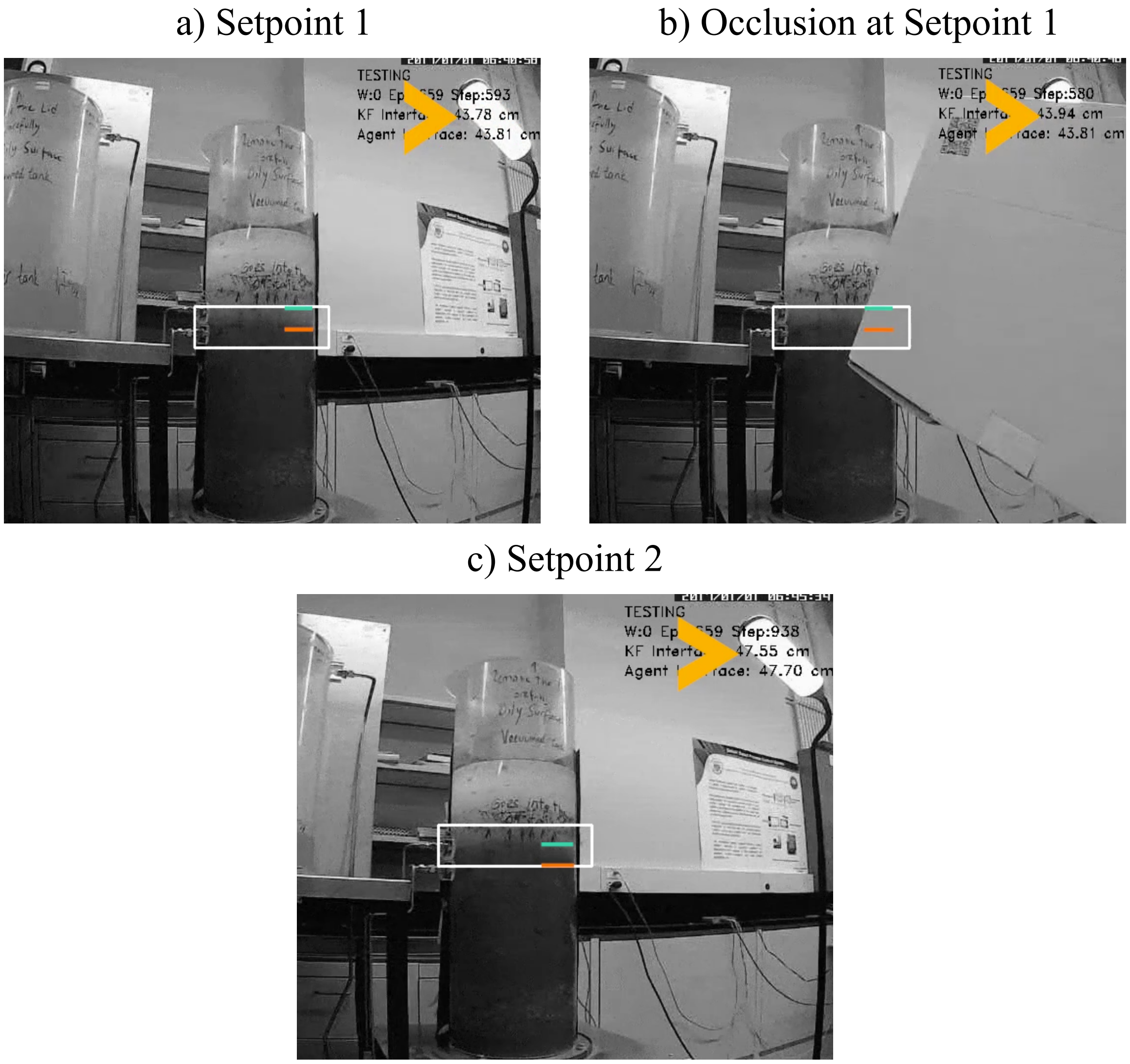


Figure 8.10: Camera snapshots of the closed-loop control in the presence of camera- and pressure-sensor noise, occlusion, and lighting changes (indicated with a yellow arrow at the top right corners of the subfigures). The center of the white box denotes the level estimated by Agent 1. a) The interface level is at the lower setpoint (which is shown as the orange line and determined by Agent 3), Agent 1 tracks the interface accurately and sends its estimation to the controllers, and the controllers keep the interface at the setpoint. b) An occluding object is introduced at the same setpoint. Agent 1 accurately tracks the interface level. c) Agent 3 moves the setpoint to a higher level within the safe region. As a result, the controllers move the interface to this new setpoint (indicated by the cyan line). Agent 1 provides reliable estimations by tracking the interface accurately.

Some instances of these time-series results are also shown in Fig. 8.10. Fig. 8.10-a shows that the interface level was at the lower setpoint (shown as the orange line and estimated by Agent 3), Agent 1 precisely tracked the interface and provided its estimation to the controllers, and the controllers maintained the interface at the setpoint. Even though the PSV was partially blocked by an object (shown in Fig. 8.10-b), Agent 1 provided accurate level estimations. In addition, Fig. 8.10-c shows that Agent 3 changed the setpoint within the safe area. As a consequence, the controllers relocated the interface to this new setpoint (indicated by the cyan line) while Agent 1 provided the correct estimations of the interface. These results proved the concept of an autonomous control scheme in an experimental setup, which can be beneficial for process industries involving multiple controllers, filters, state estimators, and agents that involve nonlinear objective functions and unknown system dynamics. These results also showed that such accurate tracking/control/optimization results can be achieved starting with simplified simulated models and learning the model-plant mismatch without compromising the safety of the process.

8.5 Challenges of Experimental Implementation of the Autonomous Process Automation System

This chapter has provided a proof of concept for process autonomy with some crucial elements in process control. However, intelligent or autonomous process automation constitutes intermediate and more complex processes that work in 'harmony'. One can think of this harmony as sequential teamwork of software and hardware, where the end of a process leads to a new one continuously and optimally, which is challenging to attain. One of the most primitive examples of such sequences can be sets of logic defined by operators via programmable logic controllers (PLCs), where the computer monitors some key process variables and switches between conditions. However, in this example, since these conditions are defined by the operator considering a limited number of situations, *process optimality and safety* can be compromised. A better alternative is to define higher-level objectives and extensively investigate all possible scenarios to improve safety and optimality. Other challenges include implementing *hard constraints* since their selection and enforcement require process knowledge and often a process model. Selecting/developing appropriate system identification methodologies while incorporating prior expert information can result in successful applications toward full autonomy. Another challenge with data-driven methods is the extensive *training time*, demonstrated in earlier chapters, where the model-free agents can require millions of training data points. Some solutions to this problem include pre-training the agents offline (using approximated process models or digital twins) since the agents' parameters are randomly initialized, using model-based RL agents where the agent learns from both sampled data and the process model, using a long sequence of historical data, or using sparse agents. Nevertheless, system identification is a substantial challenge at this step too. Although the agents provided in this thesis demonstrate increased robustness due to the type of neural networks (e.g., convolutional networks), another challenge is the *disturbance rejection* capabilities of the agents, overlooking which can deteriorate *learning and process stability.* Since these instabilities also impact the training time, disturbance rejection should be carefully handled. Moreover, *fault detection, tolerance, and recovery* are crucial aspects of process control. An autonomous process automation system should be capable of detecting a fault (ideally in advance), being tolerant to detected/undetected ones, and recovering from the resulting undesired events (e.g., instability, over/undershoots, offsets etc.). Finally, *commissioning* an autonomous process automation system is complex and requires multi-disciplinary teams to cooperate at each implementation step. After planning the project and designing, installing, testing, and calibrating the hardware and software, it is the practitioners' responsibility to ensure the process variables and decisions made by the agents/controllers are within the desired limits. For example, consider a case where an operator controls a process in 'manual mode', and the operator wants to deploy a controller that has passed multiple tests considering noise, unmeasured/measured disturbances and various dynamics in simulated environments but in the actual process. In this case, the operator should continuously monitor the process, gradually introduce the new element (control action or state estimation) to avoid abrupt changes in process variables and be ready to intervene in case of unexpected process behaviour. This gradual introduction can be achieved by using filters (e.g., low-pass) and switches. Another case is the shutdown and startup of a process, where the process moves from a steady state to a transition state. Since there can be nonlinear, impulsive, or reverse dynamics, it is crucial to maintain smooth transitions during the shutdown and startup phases. The following list summarizes the abovementioned challenges to highlight some of the possible extensions to this thesis. Note that addressing a challenge could resolve another one due to their similarity.

- Process safety and optimality,
- Constraints,
- Training time,
- Disturbance rejection, learning and process stability,
- Optimality,
- Fault detection, tolerance, and recovery,
- Commissioning.

8.6 Conclusion

This study demonstrated the feasibility of autonomous control in process industries. The control structure incorporated nonlinear/piecewise linear, continuous/discontinuous, quadratic/non-quadratic, symmetric/asymmetric reward, and state functions. The research also showed that the RL agents could be used on various types of data with diverse sizes, indicating that the RL is a scalable approach that could be used at multiple levels of the control hierarchy. Using observed values in the cost function emphasizes the RL methodology's adaptability and model-free nature, as opposed to traditional techniques. When the agents are used with well-studied approaches such as MPC, they may produce robust control, which can enhance safety and production performance in process industries.

Chapter 9

Concluding Remarks

This chapter summarizes the findings of the proposed methodologies and their application in numerous case studies. In addition, several prospective research endeavours in this area are highlighted.

9.1 Conclusion

The overarching premise investigated in this thesis is intelligent decision-making under uncertainty toward autonomous industrial control. As a result, all of the proposed techniques concentrate on certain aspects of reinforcement learning. Computer vision, state estimation, and process control are the three concepts investigated here. Each of these tasks is prevalent in many industrial processes and is complicated in nature, requiring the utilization of smart algorithms to execute them. A combination of the presented chapters results in an autonomous control mechanism, which is an essential component toward full autonomy in process industries.

The first contribution, presented in Chapter 3, is the literature review on actor-critic reinforcement learning algorithms and a robust interface tracking algorithm. The proposed algorithm targets the lowest level in the control hierarchy, namely, the instrumentation level. Unlike the existing methods, the proposed algorithm provides an end-to-end interface tracking methodology consisting of convolutional and convolutional long short-term memory networks. This structure combines the neighbouring spatial and spatio-temporal features without any explicit models or strict assumptions. Unlike the supervised/unsupervised learning methods, the proposed

RL-based tracking algorithm requires a significantly small number of images that can quickly be labelled by a user (manually) or a low-uncertainty pressure sensor (automatically). This agent showcases superior performance compared to existing methods in terms of robustness, which is one of the main requirements in state estimation and control. Finally, this work contributes to deep learning-based RL solutions by using a dimensionality reduction technique to illustrate the high-dimensional states' value functions as a performance indicator.

The second contribution, provided in Chapter 4, aims at designing an RL-based safe controller by considering the operational limitations, such as safety criteria. To achieve this goal, the proposed approach combines a deep actor-critic agent with random setpoint initialization and a Lagrangian-based soft-constrained learning scheme. The case studies shows that the soft-constrained scheme can provide smooth state transitions with low-variance actions, while multiple workers accelerate the offline training phase. Furthermore, an exploration metric was proposed inspired by the set theory.

The third contribution, shown in Chapter 5, targets the constrained nature of the uncertain quadratic/non-quadratic cost function that is commonly used in RL and process control. This chapter shows that the reduction in the signal-to-noise ratio in a process irrecoverably deteriorates the control policy and results in poor tracking/control performance. Considering the sensory noise in processes, the proposed method formulates the reward/cost function as a dynamic process with its transition and observation models. Unlike the frequentist inference methods (with a point estimate), the proposed method estimates the first and second moments of the constrained reward by using a constrained particle filter. Furthermore, the constrained estimations are calculated in multiple threads of a computer to reduce the computational time.

The fourth contribution, shown in Chapter 6, considers the dimensionality increase in online state estimation through skew state estimation. Although a closed skew-normal distribution increases the degree of freedom in state estimation, the dimensionality of its location and scale parameters increase at the end of each filtering step. This issue slows down the inferential calculation, making obtaining closed-form/analytical solutions impossible and online inference infeasible over time. After mathematically formulating the dimensionality reduction as an optimization method, empirical analyses were conducted to compare numerous statistical distance functions and optimization techniques. Finally, the proposed skew estimation scheme was utilized in reward estimation and state estimation problems.

The fifth contribution, introduced in Chapter 7, aims at finding a practical solution for the PID tuning problem. Since complex industrial plants can utilize thousands of control loops of various natures, tuning the PID controllers can be a tedious task with unknown system behaviour. Inspired by the actor-critic methodology, this chapter developed a constrained contextual bandit that autonomously tunes the PID controllers by using step-response models that are straightforward to obtain. Following a preliminary offline training phase, the proposed agent was deployed in a distributed control scheme to learn the model plant mismatch through online interaction.

The sixth contribution, described in Chapters 7 and 8, focuses on developing an autonomous MPC tuner and integrating it with an autonomous advanced control infrastructure. Although several classical methods can design MPC parameters offline, there could be significant performance degradation due to model plant mismatch or variations in operational conditions. Furthermore, implementing non-linear/non-quadratic or customizing the performance criteria through complex functions can be challenging. However, the proposed actor-critic approach can provide optimal solution alternatives to such problems through smart trial-and-error. Due to its modular and model-agnostic nature, such an agent can be trained on approximate system models and be integrated into more complex schemes, which highlights the proposed method's versatility. This chapter combines the necessary elements of an autonomous robust control scheme by integrating an interface tracking agent, a Kalman filter, an operating point optimizer, a model predictive controller, an MPC tuner, and a cascaded PID controller.

In summary, complete process autonomy can be achieved by combining state estimators, controllers and parameter optimizers at different layers of the control hierarchy. All these elements have unique structures, frequencies and targets, which makes the overall infrastructure too complex to model. Hence, smart data-driven algorithms with substantial process knowledge and algorithmic improvements, as shown in this thesis, can provide beneficial, tractable and scalable solutions to industrial process control.

9.2 Future scope

This section describes the several research areas that might be pursued in terms of additions and enhancements to the methodology described in the thesis. It should be noted that the following research directions would enhance not only offline but also online learning.

9.2.1 Stable Online Learning

The data-driven algorithms provided in the RL-oriented chapters rely on parameterized policies that are trained simultaneously with parameterized value functions that regularize learning. This combination improves the stability of RL agents, unlike supervised/unsupervised learning methods, which directly optimize a single set of parameters by using an objective function (e.g., regression/classification loss functions). Nonetheless, accurate learning techniques often require a large number of parameters, which increase the number of data points needed and deteriorate the stability during recursive learning. Due to technological advancements, a parameterized complex function (like a neural network) can be trained via automatic differentiation (also called "autograd") methods; however, the optimizers' hyperparameters (such as learning rate) significantly impact the stability. Although some regularization/scheduling techniques (e.g., learning rate annealing) have shown excellent empirical performance in global competitions and collaborative engineering events (e.g., "hackathons"), hyperparameter selection is an outstanding research topic. To address such challenges, Bayesian inference (e.g., a Kalman filter) can be infused into value/policy learning. Note that in recursive parameter estimation, the learning rate is the scalable, non-adaptive, and deterministic equivalent of the Kalman gain.

9.2.2 Model-Based, Sparse Learning

Although the proposed agents sample data points by using simulated models during offline training, the agents cannot access the underlying model structure or modify them. Since the system models are unknown to the agents, learning often requires a considerable amount of data (around millions of data points). When simulation models become more complex or involve intermediate calculations (e.g., the constrained nonlinear trajectory optimization steps in the MPC tuner), the dependency on data points significantly prolongs the learning time. Model-based learning can be utilized in learning to reduce learning time (through "imaginary/simulated" trajectory generation) and provide more robust policy/value functions through control-based approaches (like H_∞ methods). However, the validity of the model in the presence of few data points is a significant challenge due to the accuracy and stability of the model. In addition, learning global models requires retaining and optimizing using past information (as in Bayesian optimization), which can be infeasible in terms of computation. Some of these identification/learning challenges can be addressed through sparse parameterization (of the agents and system models) and Bayesian inference (e.g., a Kalman filter).

9.2.3 Robust Learning/Control

The agents/controllers often use sensory information that is pre-processed by a filter (like a moving average/Kalman filter). Although some types of neural networks (e.g., convolutional networks) can be insensitive to input variations -in fact, a neural network acts as a non-linear low-pass filter on the input data-, data-driven learning might suffer from high variations in the input data. As an alternative to convolutional or statistical filters, spectral filters can decompose the input into its varying-frequency elements. Such a spectral filter can be integrated into learning by considering the uncertainties in the state and reward functions.

References

[1] S. H. Nair, V. Govindarajan, T. Lin, C. Meissen, H. E. Tseng, and F. Borrelli, "Stochastic mpc with multi-modal predictions for traffic intersections," 2021.

[2] J. S. Wilson, *Sensor technology handbook.* Elsevier, 2004.

[3] J. A. Guzmán-Rabasa, F. R. López-Estrada, B. M. González-Contreras, G. Valencia-Palomo, M. Chadli, and M. Perez-Patricio, "Actuator fault detection and isolation on a quadrotor unmanned aerial vehicle modeled as a linear parameter-varying system," *Measurement and Control*, vol. 52, no. 9-10, pp. 1228–1239, 2019.

[4] Y. Chen, X. Huang, and J. Zhan, "Gain-scheduled robust control for multi-agent linear parameter-varying systems with communication delays," *International Journal of Robust and Nonlinear Control*, vol. 32, no. 2, pp. 792–806, 2022.

[5] J. Deng and B. Huang, "Identification of nonlinear parameter varying systems with missing output data," *AIChE Journal*, vol. 58, no. 11, pp. 3454–3467, 2012.

[6] O. Adeyemi, I. Grove, S. Peets, Y. Domun, and T. Norton, "Dynamic neural network modelling of soil moisture content for predictive irrigation scheduling," *Sensors*, vol. 18, no. 10, p. 3408, 2018.

[7] J. Berner, K. Soltesz, T. Hägglund, and K. J. Åström, "An experimental comparison of pid autotuners," *Control Engineering Practice*, vol. 73, pp. 124–133, 2018.

[8] N. Wang, Y. Gao, and X. Zhang, "Data-driven performance-prescribed reinforcement learning control of an unmanned surface vehicle," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 12, pp. 5456–5467, 2021.

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

[10] M. F. Aslan, A. Durdu, A. Yusefi, K. Sabanci, and C. Sungur, "A tutorial: Mobile robotics, slam, bayesian filter, keyframe bundle adjustment and ros applications," *Robot Operating System (ROS)*, pp. 227–269, 2021.

[11] D. Simon, "Kalman filtering," *Embedded systems programming*, vol. 14, no. 6, pp. 72–79, 2001.

[12] A. Azzalini and A. Capitanio, "Distributions generated by perturbation of symmetry with emphasis on a multivariate skew t-distribution," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 65, no. 2, pp. 367–389, 2003.

[13] G. Toulemonde, A. Guillou, and P. Naveau, "Particle filtering for gumbel-distributed daily maxima of methane and nitrous oxide," *Environmetrics*, vol. 24, no. 1, pp. 51–62, 2013.

[14] M. Bai, Y. Huang, B. Chen, and Y. Zhang, "A novel robust kalman filtering framework based on normal-skew mixture distribution," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2021.

[15] A. Azzalini and A. D. Valle, "The multivariate skew-normal distribution," *Biometrika*, vol. 83, no. 4, pp. 715–726, 1996.

[16] R. Henriksen, "The truncated second-order nonlinear filter revisited," *IEEE Transactions on Automatic Control*, vol. 27, no. 1, pp. 247–251, 1982.

[17] X. Jiang, "Iterative truncated arithmetic mean filter and its properties," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 1537–1547, 2012.

[18] W. Dong, P. Wang, W. Yin, G. Shi, F. Wu, and X. Lu, "Denoising prior driven deep neural network for image restoration," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 10, pp. 2305–2318, 2018.

[19] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[20] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[21] C. C. Margossian, "A review of automatic differentiation and its efficient implementation," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 9, no. 4, p. e1305, 2019.

[22] G. Chen, "A gentle tutorial of recurrent neural network with error backpropagation," *arXiv preprint arXiv:1610.02583*, 2016.

[23] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[26] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[27] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," in *International conference on artificial neural networks*, pp. 799–804, Springer, 2005.

[28] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[29] B. Huang, Y. Qi, and A. M. Murshed, *Dynamic modeling and predictive control in solid oxide fuel cells: first principle and data-based approaches.* John Wiley & Sons, 2013.

[30] S. Särkkä, *Bayesian filtering and smoothing.* No. 3, Cambridge university press, 2013.

[31] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches.* John Wiley & Sons, 2006.

[32] H. Nurminen, T. Ardeshiri, R. Piché, and F. Gustafsson, "Skew-*t* filter and smoother with improved covariance matrix approximation," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5618–5633, 2018.

[33] G. González-Farías, A. Domínguez-Molina, and A. K. Gupta, "Additive properties of skew normal random vectors," *Journal of statistical planning and inference*, vol. 126, no. 2, pp. 521–534, 2004.

[34] R. Bellman *et al.*, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

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

[36] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

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

[38] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[40] R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," *Computers Chemical Engineering*, vol. 139, p. 106886, 2020.

[41] T. Lu, K. Zhang, and Y. Shi, "Sarsa-based model predictive control with improved performance and computational complexity," in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*, pp. 01–06, IEEE, 2022.

[42] S. Meyn, *Control Systems and Reinforcement Learning.* Cambridge University Press, 2022.

[43] Q. Fu, L. Hu, H. Wu, F. Hu, W. Hu, and J. Chen, "A sarsa-based adaptive controller for building energy conservation," *Journal of Computational Methods in Sciences and Engineering*, vol. 18, no. 2, pp. 329–338, 2018.

[44] Q. Shi, H.-K. Lam, B. Xiao, and S.-H. Tsai, "Adaptive pid controller based on q-learning algorithm," *CAAI Transactions on Intelligence Technology*, vol. 3, no. 4, pp. 235–244, 2018.

[45] Q. Wei, T. Li, and D. Liu, "Learning control for air conditioning systems via human expressions," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 8, pp. 7662–7671, 2020.

[46] A. Kekuda, R. Anirudh, and M. Krishnan, "Reinforcement learning based intelligent traffic signal control using n-step sarsa," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pp. 379–384, IEEE, 2021.

[47] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstra-

tions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[48] X. Dong, J. Shen, W. Wang, Y. Liu, L. Shao, and F. Porikli, "Hyperparameter optimization for tracking with continuous deep q-learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 518–527, 2018.

[49] D. Zhang, F. R. Yu, and R. Yang, "Blockchain-based distributed software-defined vehicular networks: A dueling deep q-learning approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1086–1100, 2019.

[50] C. Tallec, L. Blier, and Y. Ollivier, "Making deep q-learning methods robust to time discretization," in *International Conference on Machine Learning*, pp. 6096–6104, PMLR, 2019.

[51] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep q-learning," in *Learning for Dynamics and Control*, pp. 486–489, PMLR, 2020.

[52] B. Ning, F. H. T. Lin, and S. Jaimungal, "Double deep q-learning for optimal execution," *Applied Mathematical Finance*, vol. 28, no. 4, pp. 361–380, 2021.

[53] P. Casgrain, B. Ning, and S. Jaimungal, "Deep q-learning for nash equilibria: Nash-dqn," *Applied Mathematical Finance*, pp. 1–17, 2022.

[54] J. N. Tsitsiklis and B. Van Roy, "Analysis of temporal-diffference learning with function approximation," in *Advances in neural information processing systems*, pp. 1075–1081, 1997.

[55] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[56] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural networks*, vol. 3, no. 6, pp. 671–692, 1990.

[57] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[58] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[59] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[60] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[61] R. Munos, "Policy gradient in continuous time," *Journal of Machine Learning Research*, vol. 7, pp. 771–791, 2006.

[62] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, IEEE, 2006.

[63] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *International conference on artificial neural networks*, pp. 387–396, Springer, 2008.

[64] J. Bhandari and D. Russo, "Global optimality guarantees for policy gradient methods," *arXiv preprint arXiv:1906.01786*, 2019.

[65] B. Gravell, P. M. Esfahani, and T. Summers, "Learning robust control for lqr systems with multiplicative noise via policy gradient," *arXiv preprint arXiv:1905.13547*, 2019.

[66] C.-A. Cheng, X. Yan, and B. Boots, "Trajectory-wise control variates for variance reduction in policy gradient methods," in *Conference on Robot Learning*, pp. 1379–1394, PMLR, 2020.

[67] J. Bu, A. Mesbahi, and M. Mesbahi, “Policy gradient-based algorithms for continuous-time linear quadratic control,” *arXiv preprint arXiv:2006.09178*, 2020.

[68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[69] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[70] V. R. Konda and T. J. N., “On actor-critic algorithms,” *SIAM journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[71] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[72] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv preprint arXiv:1801.01290*, 2018.

[73] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, pp. 1928–1937, 2016.

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

[75] P. Langevin, “Sur la théorie du mouvement brownien,” *Compt. Rendus*, vol. 146, pp. 530–533, 1908.

[76] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, pp. 1587–1596, PMLR, 2018.

[77] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[78] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

[79] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[80] T. Shuprajhaa, S. K. Sujit, and K. Srinivasan, "Reinforcement learning based adaptive pid controller design for control of linear/nonlinear unstable processes," *Applied Soft Computing*, vol. 128, p. 109450, 2022.

[81] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[82] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in neural information processing systems*, pp. 5279–5288, 2017.

[83] R. Grosse and J. Martens, "A kronecker-factored approximate fisher matrix for convolution layers," in *International Conference on Machine Learning*, pp. 573–582, 2016.

[84] J. Martens, J. Ba, and M. Johnson, "Kronecker-factored curvature approximations for recurrent neural networks," in *International Conference on Learning Representations*, 2018.

[85] R. Zhang, R. Leteurtre, B. Striner, A. Alanazi, A. Alghafis, and O. K. Tonguz, "Partially detected intelligent traffic signal control: Environmental adaptation," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 1956–1960, 2019.

[86] L. Zhong, J. Hu, H. Shen, C. Xu, Z. Huang, and B. Ren, "Slice allocation of 5g network for smart grid with deep reinforcement learning acktr," in *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pp. 242–249, 2022.

[87] V. Taboga, A. Bellahsen, and H. Dagdougui, "An enhanced adaptivity of reinforcement learning-based temperature control in buildings using generalized training," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 2, pp. 255–266, 2022.

[88] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[89] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, and R. Munos, "The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning," *arXiv preprint arXiv:1704.04651*, 2017.

[90] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[91] X. Yu, Y. Fan, S. Xu, and L. Ou, "A self-adaptive sac-pid control approach based on reinforcement learning for mobile robots," *International Journal of Robust and Nonlinear Control*, 2021.

[92] W. Sun, Y. Zou, X. Zhang, N. Guo, B. Zhang, and G. Du, "High robustness energy management strategy of hybrid electric vehicle based on improved soft actor-critic deep reinforcement learning," *Energy*, p. 124806, 2022.

[93] G. Campos, N. H. El-Farra, and A. Palazoglu, "Soft actor-critic deep reinforcement learning with hybrid mixed-integer actions for demand responsive scheduling of energy systems," *Industrial & Engineering Chemistry Research*, 2022.

[94] S. Ljung and L. Ljung, "Error propagation properties of recursive least-squares adaptation algorithms," *Automatica*, vol. 21, no. 2, pp. 157–167, 1985.

[95] Z. Zhang, X. Li, J. An, W. Man, and G. Zhang, "Model-free attitude control of spacecraft based on pid-guide td3 algorithm," *International Journal of Aerospace Engineering*, vol. 2020, 2020.

[96] Z. Chu, B. Sun, D. Zhu, M. Zhang, and C. Luo, "Motion control of unmanned underwater vehicles via deep imitation reinforcement learning algorithm," *IET Intelligent Transport Systems*, vol. 14, no. 7, pp. 764–774, 2020.

[97] J. Zhou, S. Xue, Y. Xue, Y. Liao, J. Liu, and W. Zhao, "A novel energy management strategy of hybrid electric vehicle via an improved td3 deep reinforcement learning," *Energy*, vol. 224, p. 120118, 2021.

[98] C. Fu and Y. Zhang, "Research and application of predictive control method based on deep reinforcement learning for hvac systems," *IEEE Access*, vol. 9, pp. 130845–130852, 2021.

[99] X. Yuan, Y. Wang, R. Zhang, Q. Gao, Z. Zhou, R. Zhou, and F. Yin, "Reinforcement learning control of hydraulic servo system based on td3 algorithm," *Machines*, vol. 10, no. 12, p. 1244, 2022.

[100] D. Dutta and S. R. Upreti, "A survey and comparative evaluation of actor-critic methods in process control," *The Canadian Journal of Chemical Engineering*, vol. 100, no. 9, pp. 2028–2056, 2022.

[101] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[102] R. Bellman, “Dynamic programming,” *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[103] V. S. Borkar, “An actor-critic algorithm for constrained markov decision processes,” *Systems & control letters*, vol. 54, no. 3, pp. 207–213, 2005.

[104] S. Bhatnagar, M. Ghavamzadeh, M. Lee, and R. S. Sutton, “Incremental natural actor-critic algorithms,” in *Advances in neural information processing systems*, pp. 105–112, 2008.

[105] J. Peters and S. Schaal, “Natural actor-critic,” *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[106] D. Vrabie and F. Lewis, “Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems,” *Neural Networks*, vol. 22, no. 3, pp. 237–246, 2009.

[107] K. G. Vamvoudakis and F. L. Lewis, “Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem,” *Automatica*, vol. 46, no. 5, pp. 878–888, 2010.

[108] T. Degris, M. White, and R. S. Sutton, “Off-policy actor-critic,” *arXiv preprint arXiv:1205.4839*, 2012.

[109] S. Bhasin, R. Kamalapurkar, M. Johnson, K. G. Vamvoudakis, F. L. Lewis, and W. E. Dixon, “A novel actor–critic–identifier architecture for approximate optimal control of uncertain nonlinear systems,” *Automatica*, vol. 49, no. 1, pp. 82–92, 2013.

[110] D. Zhao, B. Wang, and D. Liu, “A supervised actor–critic approach for adaptive cruise control,” *Soft Computing*, vol. 17, no. 11, pp. 2089–2099, 2013.

[111] H. Modares, F. L. Lewis, and M.-B. Naghibi-Sistani, “Integral reinforcement learning and experience replay for adaptive optimal control of partially-unknown constrained-input continuous-time systems,” *Automatica*, vol. 50, no. 1, pp. 193–202, 2014.

[112] S. J. Chang, J. Y. Lee, J. B. Park, and Y. H. Choi, “An online fault tolerant actor-critic neuro-control for a class of nonlinear systems using neural network hjb approach,” *International Journal of Control, Automation and Systems*, vol. 13, no. 2, pp. 311–318, 2015.

[113] B. Kiumarsi and F. L. Lewis, “Actor–critic-based optimal tracking for partially unknown nonlinear discrete-time systems,” *IEEE transactions on neural networks and learning systems*, vol. 26, no. 1, pp. 140–151, 2014.

[114] R. Song, F. Lewis, Q. Wei, H.-G. Zhang, Z.-P. Jiang, and D. Levine, “Multiple actor-critic structures for continuous-time optimal control using input-output data,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 4, pp. 851–865, 2015.

[115] C. Allen, K. Asadi, M. Roderick, A.-r. Mohamed, G. Konidaris, and M. Littman, “Mean actor critic,” *arXiv preprint arXiv:1709.00503*, 2017.

[116] N. K. Dhar, N. K. Verma, and L. Behera, “Adaptive critic-based event-triggered control for hvac system,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 178–188, 2017.

[117] Q.-Y. Fan, G.-H. Yang, and D. Ye, “Quantization-based adaptive actor-critic tracking control with tracking error constraints,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 4, pp. 970–980, 2017.

[118] Y. Wei, F. R. Yu, M. Song, and Z. Han, “User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach,” *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 680–692, 2017.

[119] Y. Wang, K. Velswamy, and B. Huang, “A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems,” *Processes*, vol. 5, no. 3, p. 46, 2017.

[120] B. Chen, D. Wang, P. Li, S. Wang, and H. Lu, "Real-time'actor-critic'tracking," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 318–334, 2018.

[121] M.-B. Radac, R.-E. Precup, and R.-C. Roman, "Data-driven model reference control of mimo vertical tank systems with model-free vrft and q-learning," *ISA transactions*, vol. 73, pp. 227–238, 2018.

[122] Z. Yang, Y. Chen, M. Hong, and Z. Wang, "On the global convergence of actor-critic: A case for linear quadratic regulator with ergodic cost," *arXiv preprint arXiv:1907.06246*, 2019.

[123] Y. Lv, J. Na, and X. Ren, "Online h infinity control for completely unknown nonlinear systems via an identifier–critic-based adp structure," *International Journal of Control*, vol. 92, no. 1, pp. 100–111, 2019.

[124] Z. Hou, K. Zhang, Y. Wan, D. Li, C. Fu, and H. Yu, "Off-policy maximum entropy reinforcement learning: Soft actor-critic with advantage weighted mixture policy (sac-awmp)," *arXiv preprint arXiv:2002.02829*, 2020.

[125] Y. Zhang, B. Zhao, and D. Liu, "Deterministic policy gradient adaptive dynamic programming for model-free optimal control," *Neurocomputing*, 2020.

[126] H. Shafi, K. Velswamy, F. Ibrahim, and B. Huang, "A hierarchical constrained reinforcement learning for optimization of bitumen recovery rate in a primary separation vessel," *Computers & Chemical Engineering*, vol. 140, p. 106939, 2020.

[127] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[128] S. D.-C. Shashua and S. Mannor, "Trust region value optimization using kalman filtering," *arXiv preprint arXiv:1901.07860*, 2019.

[129] S. D.-C. Shashua and S. Mannor, "Kalman meets bellman: Improving policy evaluation through value tracking," *arXiv preprint arXiv:2002.07171*, 2020.

[130] P.-H. Su, P. Budzianowski, S. Ultes, M. Gasic, and S. Young, "Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management," *arXiv preprint arXiv:1707.00130*, 2017.

[131] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[132] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv: 2004.07219*, 2020.

[133] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[134] P. I. Pavlov, "Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex," *Annals of neurosciences*, vol. 17, no. 3, p. 136, 2010.

[135] J. Masliyah, Z. Zhou, Z. Xu, J. Czarnecki, and H. Hamza, "Understanding water-based bitumen extraction from athabasca oil sands," *Can. J. Chem. Eng.*, vol. 82, pp. 628 – 654, 2004.

[136] P. Jampana, S. Shah, and R. Kadali, "Computer vision based interface level control in separation cells," *Control Engineering Practice*, vol. 18, pp. 349 – 357, 2010.

[137] A. Vicente, R. Raveendran, B. Huang, S. Sedghi, A. Narang, H. Jiang, and W. Mitchell, "Computer vision system for froth-middlings interface level detection in the primary separation vessels," *Computers & Chemical Engineering*, vol. 123, pp. 357 – 370, 2019.

[138] Z. Liu, H. Kodamana, A. Afacan, and B. Huang, “Dynamic prediction of interface level using spatial temporal markov random field,” *Computers & Chemical Engineering*, vol. 128, pp. 301–311, 2019.

[139] R. Xie, N. M. Jan, K. Hao, L. Chen, and B. Huang, “Supervised variational autoencoders for soft sensor modeling with missing data,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2820–2828, 2019.

[140] R. Raveendran, H. Kodamana, and B. Huang, “Process monitoring using a generalized probabilistic linear latent variable model,” *Automatica*, vol. 96, pp. 73–83, 2018.

[141] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[142] G. S. Babu, P. Zhao, and X.-L. Li, “Deep convolutional neural network based regression approach for estimation of remaining useful life,” in *International conference on database systems for advanced applications*, pp. 214–228, Springer, 2016.

[143] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[144] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[145] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[146] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[147] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[148] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Q. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[149] A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on mri," *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 102–127, 2019.

[150] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, p. 92, 2019.

[151] X. Wu, J. Chen, L. Xie, L. L. T. Chan, and C.-I. Chen, "Development of convolutional neural network based gaussian process regression to construct a novel probabilistic virtual metrology in multi-stage semiconductor processes," *Control Engineering Practice*, vol. 96, p. 104262, 2020.

[152] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the gpu–past, present and future," *Medical image analysis*, vol. 17, no. 8, pp. 1073–1094, 2013.

[153] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[154] R. Xie, K. Hao, B. Huang, L. Chen, and X. Cai, "Data-driven modeling based on two-stream λ gated recurrent unit network with soft sensor application," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 8, pp. 7034–7043, 2019.

[155] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[156] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[157] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[158] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, pp. 802–810, 2015.

[159] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[160] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[161] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[162] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[163] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[164] X. Yuan, B. Huang, Y. Wang, C. Yang, and W. Gui, "Deep learning-based feature representation and its application for soft sensor modeling with variable-wise weighted sae," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3235–3243, 2018.

[165] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[166] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*, pp. 270–279, Springer, 2018.

[167] E. L. Thorndike, *Animal Intelligence.* Hafner, Darien, CT, 1911.

[168] B. Farley and W. Clark, "Simulation of self-organizing systems by digital computer," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.

[169] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems Magazine*, vol. 12, pp. 19–22, April 1992.

[170] D. E. Kirk, *Optimal control theory: an introduction.* Courier Corporation, 2004.

[171] C. Szepesvári, *Algorithms for reinforcement learning.* Morgan & Claypool Publishers, 2010.

[172] L. Ljung, *System Identification.* American Cancer Society, 1999.

[173] H. Kodamana, B. Huang, R. Ranjan, Y. Zhao, R. Tan, and N. Sammaknejad, "Approaches to robust process identification: A review and tutorial of probabilistic methods," *Journal of Process Control*, vol. 66, pp. 68–83, 2018.

[174] M. D. Pendrith and C. Sammut, *On reinforcement learning of control actions in noisy and non-Markovian domains.* Citeseer, 1994.

[175] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," *CoRR*, vol. abs/1706.01905, 2017.

[176] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in neural information processing systems*, pp. 2753–2762, 2017.

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

[178] K. Ciosek, Q. Vuong, R. Loftin, and K. Hofmann, "Better exploration with optimistic actor-critic," 2019.

[179] K. S. Luck, M. Vecerik, S. Stepputtis, H. B. Amor, and J. Scholz, "Improved exploration through latent trajectory optimization in deep deterministic policy gradient," 2019.

[180] L. Couffignal, *Les machines a calculer, leurs principes, leur evolution, Gauthier-Villars, Paris.* 1933.

[181] A. M. Turing, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, pp. 433–460, 10 1950.

[182] C. Arf, *Makine dusunebilir mi ve nasil dusunebilir, Ataturk Universitesi, Universite Calismalarini Muhite Yayma ve Halk Egitimi Yayinlari Konferanslar Serisi, Erzurum, 91-103.* 1959.

[183] S. Spielberg, R. Gopaluni, and P. Loewen, "Deep reinforcement learning approaches for process control," in *2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, pp. 201–206, IEEE, 2017.

[184] B. J. Pandian and M. M. Noel, “Tracking control of a continuous stirred tank reactor using direct and tuned reinforcement learning based controllers,” *Chemical Product and Process Modeling*, vol. 13, no. 3, 2018.

[185] T. A. Badgwell, J. H. Lee, and K.-H. Liu, “Reinforcement learning   overview of recent progress and implications for process control,” in *13th International Symposium on Process Systems Engineering (PSE 2018)* (M. R. Eden, M. G. Ierapetritou, and G. P. Towler, eds.), vol. 44 of *Computer Aided Chemical Engineering*, pp. 71 – 85, Elsevier, 2018.

[186] Y. Ruan, Y. Zhang, T. Mao, X. Zhou, D. Li, and H. Zhou, “Trajectory optimization and positioning control for batch process using learning control,” *Control Engineering Practice*, vol. 85, pp. 1–10, 2019.

[187] L. Zhu, Y. Cui, G. Takami, H. Kanokogi, and T. Matsubara, “Scalable reinforcement learning for plant-wide control of vinyl acetate monomer process,” *Control Engineering Practice*, vol. 97, p. 104331, 2020.

[188] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[189] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, “Human-level performance in 3d multiplayer games with population-based reinforcement learning,” vol. 364, no. 6443, pp. 859–865, 2019.

[190] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[191] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[192] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv: 1909.07528*, 2019.

[193] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[194] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," in *International Conference on Machine Learning*, pp. 507–517, PMLR, 2020.

[195] I. O. Bucak and M. A. Zohdy, "Reinforcement learning control of nonlinear multi-link system," *Engineering Applications of Artificial Intelligence*, vol. 14, no. 5, pp. 563 – 575, 2001.

[196] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[197] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, 2020.

[198] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, "Low-level autonomous control and tracking of quadrotor using reinforcement learning," *Control Engineering Practice*, vol. 95, p. 104222, 2020.

[199] S. Mathe, A. Pirinen, and C. Sminchisescu, "Reinforcement learning for visual object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2894–2902, 2016.

[200] J. König, S. Malberg, M. Martens, S. Niehaus, A. Krohn-Grimberghe, and A. Ramaswamy, "Multi-stage reinforcement learning for object detection," in *Science and Information Conference*, pp. 178–191, Springer, 2019.

[201] E. Halici and A. A. Alatan, “Object localization without bounding box information using generative adversarial reinforcement learning,” in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 3728–3732, Oct 2018.

[202] D. Zhang, H. Maei, X. Wang, and Y. Wang, “Deep reinforcement learning for visual object tracking in videos,” *CoRR*, vol. abs/1701.08936, 2017.

[203] W. Luo, P. Sun, Y. Mu, and W. Liu, “End-to-end active object tracking via reinforcement learning,” *CoRR*, vol. abs/1705.10561, 2017.

[204] L. Ren, J. Lu, Z. Wang, Q. Tian, and J. Zhou, “Collaborative deep reinforcement learning for multi-object tracking,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 586–602, 2018.

[205] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Young Choi, “Action-decision networks for visual tracking with deep reinforcement learning,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2711–2720, 2017.

[206] J. Choi, J. Kwon, and K. M. Lee, “Visual tracking by reinforced decision making,” *CoRR*, vol. abs/1702.06291, 2017.

[207] P. Li, D. Wang, L. Wang, and H. Lu, “Deep visual tracking: Review and experimental comparison,” *Pattern Recognition*, vol. 76, pp. 323 – 338, 2018.

[208] B. X. Chen and J. K. Tsotsos, “Fast visual object tracking with rotated bounding boxes,” *arXiv preprint arXiv:1907.03892*, 2019.

[209] Z. Wang, J. Xu, L. Liu, F. Zhu, and L. Shao, “Ranet: Ranking attention network for fast video object segmentation,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3978–3987, 2019.

[210] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Pro-*

ceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 1725–1732, 2014.

[211] J. Li, X. Chen, E. Hovy, and D. Jurafsky, "Visualizing and understanding neural models in nlp," *arXiv preprint arXiv:1506.01066*, 2015.

[212] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[213] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-sne effectively," *Distill*, vol. 1, no. 10, p. e2, 2016.

[214] N. B. Zrihem, T. Zahavy, and S. Mannor, "Visualizing dynamics: from t-sne to semi-mdps," *arXiv preprint arXiv:1606.07112*, 2016.

[215] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau, "Combined reinforcement learning via abstract representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3582–3589, 2019.

[216] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[217] Y. Zhao, A. Fatehi, and B. Huang, "A data-driven hybrid arx and markov chain modeling approach to process identification with time-varying time delays," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 5, pp. 4226–4236, 2016.

[218] B. Huang and R. Kadali, *Dynamic modeling, predictive control and performance monitoring: a data-driven subspace approach.* Springer, 2008.

[219] R. C. Gonzalez and R. E. Woods, *Digital Image Processing.* Pearson, 4 ed., 2018.

[220] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever, “Generative pretraining from pixels,” in *Proceedings of the 37th International Conference on Machine Learning*, 2020.

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

[222] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, p. 484, 2016.

[223] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, “Agent57: Outperforming the atari human benchmark,” *arXiv preprint arXiv:2003.13350*, 2020.

[224] B. K. M. Powell, D. Machalek, and T. Quah, “Real-time optimization using reinforcement learning,” *Computers & Chemical Engineering*, vol. 143, p. 107077, 2020.

[225] T. Barrett, W. Clements, J. Foerster, and A. Lvovsky, “Exploratory combinatorial optimization with reinforcement learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3243–3250, Apr. 2020.

[226] W. He, H. Gao, C. Zhou, C. Yang, and Z. Li, “Reinforcement learning control of a flexible two-link manipulator: An experimental investigation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–11, 2020.

[227] Z. Zheng, L. Ruan, M. Zhu, and X. Guo, “Reinforcement learning control for underactuated surface vessel with output error constraints and uncertainties,” *Neurocomputing*, vol. 399, pp. 479 – 490, 2020.

[228] I. Fox, J. Lee, R. Pop-Busui, and J. Wiens, “Deep reinforcement learning for closed-loop blood glucose control,” in *Proceedings of the 5th Machine Learning for Healthcare Conference* (F. Doshi-Velez, J. Fackler, K. Jung, D. Kale,

R. Ranganath, B. Wallace, and J. Wiens, eds.), vol. 126 of *Proceedings of Machine Learning Research*, (Virtual), pp. 508–536, PMLR, 07–08 Aug 2020.

[229] G. A. de Morais, L. B. Marcos, J. N. A. Bueno, N. F. de Resende, M. H. Terra, and V. Grassi Jr, "Vision-based robust control framework based on deep reinforcement learning applied to autonomous ground vehicles," *Control Engineering Practice*, vol. 104, p. 104630, 2020.

[230] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, "Low-level autonomous control and tracking of quadrotor using reinforcement learning," *Control Engineering Practice*, vol. 95, p. 104222, 2020.

[231] Z. Ge, Z. Song, S. X. Ding, and B. Huang, "Data mining and analytics in the process industry: The role of machine learning," *IEEE Access*, vol. 5, pp. 20590–20616, 2017.

[232] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[233] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[234] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952.

[235] L. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes.* Harvard University Press, 1980.

[236] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv preprint arXiv:2006.16712*, 2020.

[237] A. Plaat, W. Kosters, and M. Preuss, "Model-based deep reinforcement learning for high-dimensional problems, a survey," *arXiv preprint arXiv:2008.05598*, 2020.

[238] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[239] K. P. Wabersich and M. N. Zeilinger, "Performance and safety of bayesian model predictive control: Scalable model-based rl with guarantees," *arXiv preprint arXiv:2006.03483*, 2020.

[240] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum, "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations," *arXiv preprint arXiv:1904.06387*, 2019.

[241] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[242] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[243] L. Zheng and L. J. Ratliff, "Constrained upper confidence reinforcement learning," *arXiv preprint arXiv:2001.09377*, 2020.

[244] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *arXiv preprint arXiv:1502.05767*, 2018.

[245] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[246] D. Simon, J. Löfberg, and T. Glad, "Nonlinear model predictive control using feedback linearization and local inner convex constraint approximations," in *2013 European Control Conference (ECC)*, pp. 2056–2061, IEEE, 2013.

[247] S. Nagendra, N. Podila, R. Ugarakhod, and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 26–32, IEEE, 2017.

[248] N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. J. Pister, "Low level control of a quadrotor with deep model-based reinforcement learning," *CoRR*, vol. abs/1901.03737, 2019.

[249] B. J. Pandian and M. M. Noel, “Control of a bioreactor using a new partially supervised reinforcement learning algorithm,” *Journal of Process Control*, vol. 69, pp. 16–29, 2018.

[250] Y. Ma, W. Zhu, M. G. Benton, and J. Romagnoli, “Continuous control of a polymerization system with deep reinforcement learning,” *Journal of Process Control*, vol. 75, pp. 40–47, 2019.

[251] A. Fatehi and B. Huang, “Kalman filtering approach to multi-rate information fusion in the presence of irregular sampling rate and variable measurement delay,” *Journal of Process Control*, vol. 53, pp. 15–25, 2017.

[252] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning.,” in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[253] M. J. Olanrewaju, B. Huang, and A. Afacan, “Online composition estimation and experiment validation of distillation processes with switching dynamics,” *Chemical engineering science*, vol. 65, no. 5, pp. 1597–1608, 2010.

[254] W. C. Li and L. T. Biegler, “Process control strategies for constrained nonlinear systems,” *Industrial & engineering chemistry research*, vol. 27, no. 8, pp. 1421–1433, 1988.

[255] Z. Cai and N. Vasconcelos, “Cascade R-CNN: high quality object detection and instance segmentation,” *CoRR*, vol. abs/1906.09756, 2019.

[256] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, “Using of jaccard coefficient for keywords similarity,” in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, pp. 380–384, 2013.

[257] X.-S. Yang and X. He, “Bat algorithm: Literature review and applications,” *Int. J. Bio-Inspired Comput.*, vol. 5, pp. 141–149, July 2013.

[258] V. Kirubakaran, T. Radhakrishnan, and N. Sivakumaran, “Metaheuristic patient estimation based patient-specific fuzzy aggregated artificial pancreas de-

sign," *Industrial & Engineering Chemistry Research*, vol. 53, no. 39, pp. 15052–15070, 2014.

[259] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[260] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, p. 8, 2012.

[261] M. Ben-Ari and F. Mondada, "Finite state machines," in *Elements of Robotics*, pp. 55–61, Springer, 2018.

[262] L. Zhang and B. Huang, "Robust model predictive control of singular systems," *IEEE Transactions on Automatic Control*, vol. 49, no. 6, pp. 1000–1006, 2004.

[263] G. E. Dullerud and F. Paganini, *A course in robust control theory: a convex approach*, vol. 36. Springer Science & Business Media, 2013.

[264] M. Shah and F. Schneider, "Control hierarchy in a distributed process control system," in *Software for Computer Control 1982*, pp. 133–138, Elsevier, 1983.

[265] R. Chiplunkar and B. Huang, "Siamese neural network-based supervised slow feature extraction for soft sensor application," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 9, pp. 8953–8962, 2021.

[266] H. Chen, Z. Chai, B. Jiang, and B. Huang, "Data-driven fault detection for dynamic systems with performance degradation: A unified transfer learning framework," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2020.

[267] X. Yang, H. He, and X. Zhong, "Adaptive dynamic programming for robust regulation and its application to power systems," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5722–5732, 2017.

[268] X. Lin, B. Zhou, and Y. Xia, “Online recursive power management strategy based on the reinforcement learning algorithm with cosine similarity and a forgetting factor,” *IEEE Transactions on Industrial Electronics*, 2020.

[269] X. Zhao, B. Tao, L. Qian, and H. Ding, “Model-based actor-critic learning for optimal tracking control of robots with input saturation,” *IEEE Transactions on Industrial Electronics*, 2020.

[270] O. Dogru, K. Velswamy, and B. Huang, “Actor-critic reinforcement learning and application in developing computer-vision-based interface tracking,” *Engineering*, 2021.

[271] S. Mukherjee, H. Bai, and A. Chakrabortty, “Reduced-dimensional reinforcement learning control using singular perturbation approximations,” *Automatica*, vol. 126, p. 109451, 2021.

[272] J. Lee and R. S. Sutton, “Policy iterations for reinforcement learning problems in continuous time and space - fundamental theory and methods,” *Automatica*, vol. 126, p. 109421, 2021.

[273] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[274] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem, “Direct shape optimization through deep reinforcement learning,” *Journal of Computational Physics*, vol. 428, p. 110080, 2021.

[275] J. Wang, Y. Liu, and B. Li, “Reinforcement learning with perturbed rewards,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 6202–6209, 2020.

[276] T. Everitt, V. Krakovna, L. Orseau, M. Hutter, and S. Legg, “Reinforcement learning with a corrupted reward channel,” *arXiv preprint arXiv:1705.08417*, 2017.

[277] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[278] W. Mekarapiruk and R. Luus, "Optimal control of inequality state constrained systems," *Industrial & Engineering Chemistry Research*, vol. 36, no. 5, pp. 1686–1694, 1997.

[279] D. Simon and D. L. Simon, "Constrained kalman filtering via density function truncation for turbofan engine health estimation," *International Journal of Systems Science*, vol. 41, no. 2, pp. 159–171, 2010.

[280] R. Kandepu, L. Imsland, and B. A. Foss, "Constrained state estimation using the unscented kalman filter," in *2008 16th Mediterranean Conference on Control and Automation*, pp. 1453–1458, IEEE, 2008.

[281] J. Prakash, B. Huang, and S. L. Shah, "Recursive constrained state estimation using modified extended kalman filter," *Computers & chemical engineering*, vol. 65, pp. 9–17, 2014.

[282] J. Morimoto and K. Doya, "Robust reinforcement learning," *Neural computation*, vol. 17, no. 2, pp. 335–359, 2005.

[283] A. Tulsyan, R. Bhushan Gopaluni, and S. R. Khare, "Particle filtering without tears: A primer for beginners," *Computers & Chemical Engineering*, vol. 95, pp. 130–145, 2016.

[284] Z. Zhao, B. Huang, and F. Liu, "Constrained particle filtering methods for state estimation of nonlinear process," *AIChE Journal*, vol. 60, no. 6, pp. 2072–2082, 2014.

[285] X. Shao, B. Huang, and J. M. Lee, "Constrained bayesian state estimation–a comparative study and a new particle filter based approach," *Journal of Process Control*, vol. 20, no. 2, pp. 143–157, 2010.

[286] Z. Zhao, A. Tulsyan, B. Huang, and F. Liu, "Estimation and identification in batch processes with particle filters," *Journal of Process Control*, vol. 81, pp. 1–14, 2019.

[287] P. Jampana, S. Shah, R. Kadali, and D. Kihas, "Computer vision based interface level control in a separation cell," *IFAC Proceedings Volumes, 17th IFAC World Congress*, vol. 41, 2008.

[288] R. Li, N. M. Jan, B. Huang, and V. Prasad, "Constrained ensemble kalman filter based on kullback–leibler divergence," *Journal of Process Control*, vol. 81, pp. 150–161, 2019.

[289] S. Zhao and B. Huang, "Trial-and-error or avoiding a guess? initialization of the kalman filter," *Automatica*, vol. 121, p. 109184, 2020.

[290] A. Naik, R. Shariff, N. Yasui, and R. S. Sutton, "Discounted reinforcement learning is not an optimization problem," *arXiv preprint arXiv:1910.02140*, 2019.

[291] C. P. Robert, "Simulation of truncated normal variables," *Statistics and computing*, vol. 5, no. 2, pp. 121–125, 1995.

[292] R. Douc and O. Cappé, "Comparison of resampling schemes for particle filtering," in *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, pp. 64–69, IEEE, 2005.

[293] H. U. Sheikh and L. Bölöni, "Multi-agent reinforcement learning for problems with combined individual and team reward," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.

[294] M.-A. Chao, C.-Y. Chu, C.-H. Chao, and A.-Y. Wu, "Efficient parallelized particle filter design on cuda," in *2010 IEEE Workshop On Signal Processing Systems*, pp. 299–304, IEEE, 2010.

[295] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[296] W. Li, G. Wei, D. Ding, Y. Liu, and F. E. Alsaadi, "A new look at boundedness of error covariance of kalman filtering," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 2, pp. 309–314, 2018.

[297] R. V. Hogg and S. A. Klugman, "On the estimation of long tailed skewed distributions with actuarial applications," *Journal of Econometrics*, vol. 23, no. 1, pp. 91–102, 1983.

[298] J. Rezaie and J. Eidsvik, "Kalman filter variants in the closed skew normal setting," *Computational Statistics & Data Analysis*, vol. 75, pp. 1–14, 2014.

[299] R. B. Arellano-Valle, J. E. Contreras-Reyes, F. O. L. Quintero, and A. Valdebenito, "A skew-normal dynamic linear model and bayesian forecasting," *Computational Statistics*, vol. 34, no. 3, pp. 1055–1085, 2019.

[300] R. Chiplunkar and B. Huang, "Filtering and smoothing of hidden monotonic trends and application to fouling detection," *IFAC-PapersOnLine*, vol. 54, no. 3, pp. 427–432, 2021.

[301] R. Chiplunkar and B. Huang, "Latent variable modeling and state estimation of non-stationary processes driven by monotonic trends," *Journal of Process Control*, vol. 108, pp. 40–54, 2021.

[302] C. Flecher, D. Allard, and P. Naveau, "Truncated skew-normal distributions: moments, estimation by weighted moments and application to climatic data," *Metron*, vol. 68, no. 3, pp. 331–345, 2010.

[303] S. J. Julier, "Skewed approach to filtering," in *Signal and Data Processing of Small Targets 1998*, vol. 3373, pp. 271–282, International Society for Optics and Photonics, 1998.

[304] H. Zareifard and M. J. Khaledi, "Non-gaussian modeling of spatial data using scale mixing of a unified skew gaussian process," *Journal of Multivariate Analysis*, vol. 114, pp. 16–28, 2013.

[305] P. Zarrin, M. Maleki, Z. Khodadai, and R. B. Arellano-Valle, "Time series models based on the unrestricted skew-normal process," *Journal of Statistical Computation and Simulation*, vol. 89, no. 1, pp. 38–51, 2019.

[306] K. Kraus and N. Pfeifer, "Determination of terrain models in wooded areas with airborne laser scanner data," *ISPRS Journal of Photogrammetry and remote Sensing*, vol. 53, no. 4, pp. 193–203, 1998.

[307] J. Rezaie and J. Eidsvik, "A skewed unscented kalman filter," *International Journal of Control*, vol. 89, no. 12, pp. 2572–2583, 2016.

[308] H. Yu, J. Shang, and T. Chen, "On stochastic and deterministic event-based state estimation," *Automatica*, vol. 123, p. 109314, 2021.

[309] R. Krenek, J. Cha, B. R. Cho, and J. L. Sharp, "Development of statistical convolutions of truncated normal and truncated skew normal distributions with applications," *Journal of Statistical Theory and Practice*, vol. 11, no. 1, pp. 1–25, 2017.

[310] C. Xu, S. Zhao, Y. Ma, B. Huang, and F. Liu, "Robust filter design for asymmetric measurement noise using variational bayesian inference," *IET Control Theory & Applications*, vol. 13, no. 11, pp. 1656–1664, 2019.

[311] Y. Huang, Y. Zhang, P. Shi, Z. Wu, J. Qian, and J. A. Chambers, "Robust kalman filters based on gaussian scale mixture distributions with application to target tracking," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 10, pp. 2082–2096, 2017.

[312] G. González-Farías, J. Domínguez-Molina, and A. K. Gupta, "The closed skew-normal distribution," *Skew-elliptical distributions and their applications: a journey beyond normality*, pp. 25–42, 2004.

[313] O. Karimi, H. Omre, and M. Mohammadzadeh, “Bayesian closed-skew gaussian inversion of seismic avo data for elastic material properties,” *Geophysics*, vol. 75, no. 1, pp. R1–R11, 2010.

[314] P. Naveau, M. G. Genton, and X. Shen, “A skewed kalman filter,” *Journal of multivariate Analysis*, vol. 94, no. 2, pp. 382–400, 2005.

[315] M. Yue, C. An, and Z. Li, “Constrained adaptive robust trajectory tracking for wip vehicles using model predictive control and extended state observer,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 5, pp. 733–742, 2016.

[316] J. Yoo and K. H. Johansson, “Event-triggered model predictive control with a statistical learning,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 4, pp. 2571–2581, 2021.

[317] H. Jung, S. Heo, and J. H. Lee, “Model predictive control for amine-based co2 capture process with advanced flash stripper,” *Control Engineering Practice*, vol. 114, p. 104885, 2021.

[318] R. Cui, C. Yang, Y. Li, and S. Sharma, “Adaptive neural network control of auvs with control input nonlinearities using reinforcement learning,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 6, pp. 1019–1029, 2017.

[319] W. He, H. Gao, C. Zhou, C. Yang, and Z. Li, “Reinforcement learning control of a flexible two-link manipulator: An experimental investigation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 12, pp. 7326–7336, 2021.

[320] Z. He, L. Dong, C. Sun, and J. Wang, “Asynchronous multithreading reinforcement-learning-based path planning and tracking for unmanned underwater vehicle,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 5, pp. 2757–2769, 2022.

[321] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Looking back on the actor-critic architecture," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 40–50, 2021.

[322] A. Gonzalez-Garcia, D. Barragan-Alcantar, I. Collado-Gonzalez, and L. Garrido, "Adaptive dynamic programming and deep reinforcement learning for the control of an unmanned surface vehicle: Experimental results," *Control Engineering Practice*, vol. 111, p. 104807, 2021.

[323] O. Dogru, R. Chiplunkar, and B. Huang, "Reinforcement learning with constrained uncertain reward function through particle filtering," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 7, pp. 7491–7499, 2022.

[324] M. Yue, X. Hou, X. Zhao, and X. Wu, "Robust tube-based model predictive control for lane change maneuver of tractor-trailer vehicles based on a polynomial trajectory," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 12, pp. 5180–5188, 2020.

[325] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[326] O. Dogru, K. Velswamy, and B. Huang, "Actor-critic reinforcement learning and application in developing computer-vision-based interface tracking," *Engineering*, vol. 7, no. 9, pp. 1248–1261, 2021.

[327] D. H. Iversen, "Closed-skew distributions: Simulation, inversion and parameter estimation," Master's thesis, Institutt for matematiske fag, 2010.

[328] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan. corr abs/1701.07875 (2017)," *arXiv preprint arXiv:1701.07875*, 2017.

[329] Y. Chen, Z. Lin, and H.-G. Müller, "Wasserstein regression," *Journal of the American Statistical Association*, pp. 1–40, 2021.

[330] J. Nocedal and S. J. Wright, "Sequential quadratic programming," *Numerical optimization*, pp. 529–562, 2006.

[331] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[332] R. Fletcher, “Conjugate gradient methods for indefinite systems,” in *Numerical analysis*, pp. 73–89, Springer, 1976.

[333] S. Singer and J. Nelder, “Nelder-mead algorithm,” *Scholarpedia*, vol. 4, no. 7, p. 2928, 2009.

[334] C. G. Broyden, “The convergence of a class of double-rank minimization algorithms 1. general considerations,” *IMA Journal of Applied Mathematics*, vol. 6, no. 1, pp. 76–90, 1970.

[335] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, “Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization,” *ACM Transactions on mathematical software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[336] M. J. Powell, “A direct search optimization method that models the objective and constraint functions by linear interpolation,” in *Advances in optimization and numerical analysis*, pp. 51–67, Springer, 1994.

[337] A. R. Conn, N. I. Gould, and P. L. Toint, *Trust region methods.* SIAM, 2000.

[338] O. Dogru, N. Wieczorek, K. Velswamy, F. Ibrahim, and B. Huang, “Online reinforcement learning for a continuous space system with experimental validation,” *Journal of Process Control*, vol. 104, pp. 86–100, 2021.

[339] M. Broniatowski and S. Leorato, “An estimation method for the neyman chi-square divergence with application to test of hypotheses,” *Journal of multivariate analysis*, vol. 97, no. 6, pp. 1409–1436, 2006.

[340] A. Bhattacharyya, “On a measure of divergence between two multinomial populations,” *Sankhyā: the indian journal of statistics*, pp. 401–406, 1946.

[341] S. Verdú, “Total variation distance and the distribution of relative information,” in *2014 Information Theory and Applications Workshop (ITA)*, pp. 1–3, IEEE, 2014.

[342] Z. Yao, Z. Lai, and W. Liu, “A symmetric kl divergence based spatiogram similarity measure,” in *2011 18th IEEE International Conference on Image Processing*, pp. 193–196, IEEE, 2011.

[343] A. Chan, H. Silva, S. Lim, T. Kozuno, A. R. Mahmood, and M. White, “Greedification operators for policy optimization: Investigating forward and reverse kl divergences,” *arXiv preprint arXiv:2107.08285*, 2021.

[344] B. Fuglede and F. Topsoe, “Jensen-shannon divergence and hilbert space embedding,” in *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, p. 31, IEEE, 2004.

[345] D. A. Cieslak, T. R. Hoens, N. V. Chawla, and W. P. Kegelmeyer, “Hellinger distance decision trees are robust and skew-insensitive,” *Data Mining and Knowledge Discovery*, vol. 24, no. 1, pp. 136–158, 2012.

[346] G. Belitskii *et al.*, *Matrix norms and their applications*, vol. 36. Birkhäuser, 2013.

[347] M. L. Rizzo and G. J. Székely, “Energy distance,” *wiley interdisciplinary reviews: Computational statistics*, vol. 8, no. 1, pp. 27–38, 2016.

[348] T. Blevins, W. K. Wojsznis, and M. Nixon, *Advanced control foundation: tools, techniques and applications*. International Society of Automation (ISA), 2013.

[349] G. K. McMillan, *Process/industrial instruments and controls handbook*. McGraw-Hill Education, 1999.

[350] J. G. Ziegler, “Those magnificent men and their controlling machines,” *Journal of Dynamic Systems Measurement and Control-transactions of The Asme*, vol. 97, pp. 279–280, 1975.

[351] K. J. Åström, “Control system design,” 2002.

[352] D. E. Seborg, D. A. Mellichamp, T. F. Edgar, and F. J. Doyle III, *Process dynamics and control*. John Wiley & Sons, 2010.

[353] J. G. Ziegler, N. B. Nichols, *et al.*, "Optimum settings for automatic controllers," *trans. ASME*, vol. 64, no. 11, 1942.

[354] G. Cohen and G. Coon, "Theoretical consideration of retarded control," *Trans. ASME, 75 (1953), pp. 827-834*.

[355] S. Tjokro and S. L. Shah, "Adaptive pid control," in *1985 American Control Conference*, pp. 1528–1534, IEEE, 1985.

[356] D. E. Rivera, M. Morari, and S. Skogestad, "Internal model control: Pid controller design," *Industrial & engineering chemistry process design and development*, vol. 25, no. 1, pp. 252–265, 1986.

[357] I.-L. Chien and P. Fruehauf, "Consider imc tuning to improve controller performance," *Chemical Engineering Progress*, vol. 86, no. 10, pp. 33–41, 1990.

[358] S. Skogestad, "Simple analytic rules for model reduction and pid controller tuning," *Journal of process control*, vol. 13, no. 4, pp. 291–309, 2003.

[359] K. J. Åström and T. Hägglund, "Revisiting the ziegler–nichols step response method for pid control," *Journal of process control*, vol. 14, no. 6, pp. 635–650, 2004.

[360] C. Madhuranthakam, A. Elkamel, and H. Budman, "Optimal tuning of pid controllers for foptd, soptd and soptd with lead processes," *Chemical Engineering and Processing: Process Intensification*, vol. 47, no. 2, pp. 251–264, 2008.

[361] K. J. Åström and T. Hägglund, "Automatic tuning of simple regulators with specifications on phase and amplitude margins," *Automatica*, vol. 20, no. 5, pp. 645–651, 1984.

[362] K. M. Powell, D. Machalek, and T. Quah, "Real-time optimization using reinforcement learning," *Computers & Chemical Engineering*, vol. 143, p. 107077, 2020.

[363] Y. Li, W. Gao, W. Yan, S. Huang, R. Wang, V. Gevorgian, and D. W. Gao, "Data-driven optimal control strategy for virtual synchronous generator via deep reinforcement learning approach," *Journal of Modern Power Systems and Clean Energy*, 2021.

[364] Y. Bao, Y. Zhu, and F. Qian, "A deep reinforcement learning approach to improve the learning performance in process control," *Industrial & Engineering Chemistry Research*, 2021.

[365] L. A. Brujeni, J. M. Lee, and S. L. Shah, *Dynamic tuning of PI-controllers based on model-free reinforcement learning methods.* IEEE, 2010.

[366] A. El Hakim, H. Hindersah, and E. Rijanto, "Application of reinforcement learning on self-tuning pid controller for soccer robot multi-agent system," in *2013 joint international conference on rural information & communication technology and electric-vehicle technology (rICT & ICeV-T)*, pp. 1–6, IEEE, 2013.

[367] P. Kofinas and A. I. Dounis, "Online tuning of a pid controller with a fuzzy reinforcement learning mas for flow rate control of a desalination unit," *Electronics*, vol. 8, no. 2, p. 231, 2019.

[368] Q. Sun, C. Du, Y. Duan, H. Ren, and H. Li, "Design and application of adaptive pid controller based on asynchronous advantage actor–critic learning method," *Wireless Networks*, pp. 1–11, 2019.

[369] W. J. Shipman and L. C. Coetzee, "Reinforcement learning and deep neural networks for pi controller tuning," *IFAC-PapersOnLine*, vol. 52, no. 14, pp. 111–116, 2019.

[370] M. Sedighizadeh and A. Rezazadeh, "Adaptive pid controller based on reinforcement learning for wind turbine control," in *Proceedings of world academy of science, engineering and technology*, vol. 27, pp. 257–262, Citeseer, 2008.

[371] I. Carlucho, M. De Paula, S. A. Villar, and G. G. Acosta, "Incremental q-learning strategy for adaptive pid control of mobile robots," *Expert Systems with Applications*, vol. 80, pp. 183–199, 2017.

[372] N. P. Lawrence, G. E. Stewart, P. D. Loewen, M. G. Forbes, J. U. Backstrom, and R. B. Gopaluni, "Reinforcement learning based design of linear fixed structure controllers," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 230–235, 2020.

[373] N. P. Lawrence, M. G. Forbes, P. D. Loewen, D. G. McClement, J. U. Backstrom, and R. B. Gopaluni, "Deep reinforcement learning with shallow controllers: An experimental application to pid tuning," *arXiv preprint arXiv:2111.07171*, 2021.

[374] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[375] Z. Yu, J. Wang, B. Huang, and Z. Bi, "Performance assessment of pid control loops subject to setpoint changes," *Journal of Process Control*, vol. 21, no. 8, pp. 1164–1171, 2011.

[376] B. W. Bequette, *Process control: modeling, design, and simulation.* Prentice Hall Professional, 2003.

[377] T. Hägglund, "The one-third rule for pi controller tuning," *Computers & Chemical Engineering*, vol. 127, pp. 25–30, 2019.

[378] J. Pongfai, C. Angeli, P. Shi, X. Su, and W. Assawinchaichote, "Optimal pid controller autotuning design for mimo nonlinear systems based on the adaptive slp algorithm," *International Journal of Control, Automation and Systems*, vol. 19, no. 1, pp. 392–403, 2021.

[379] L. Wang, *PID control system design and automatic tuning using MATLAB/Simulink.* John Wiley & Sons, 2020.

[380] M. Huba, D. Vrancic, and P. Bistak, "Pid control with higher order derivative degrees for ipdt plant models," *IEEE Access*, 2020.

[381] S. Ulusoy, S. M. Nigdeli, and G. Bekdaş, "Novel metaheuristic-based tuning of pid controllers for seismic structures and verification of robustness," *Journal of Building Engineering*, vol. 33, p. 101647, 2021.

[382] I. Pandey, A. Panda, and P. Bhowmick, "Kalman filter and its application on tuning pi controller parameters," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1551–1556, IEEE, 2021.

[383] A. O'dwyer, *Handbook of PI and PID controller tuning rules.* World Scientific, 2009.

[384] M. Irshad and A. Ali, "A review on pid tuning rules for soptd inverse response processes," in *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, pp. 17–22, IEEE, 2017.

[385] S. Bharat, A. Ganguly, R. Chatterjee, B. Basak, D. K. Sheet, and A. Ganguly, "A review on tuning methods for pid controller," *Asian Journal For Convergence In Technology (AJCT)*, 2019.

[386] M. P. Dev, S. Jain, H. Kumar, B. Tripathi, and S. Khan, "Various tuning and optimization techniques employed in pid controller: A review," in *Proceedings of International Conference in Mechanical and Energy Technology*, pp. 797–805, Springer, 2020.

[387] R. P. Borase, D. Maghade, S. Sondkar, and S. Pawar, "A review of pid control, tuning methods and applications," *International Journal of Dynamics and Control*, pp. 1–10, 2020.

[388] U. M. Nath, C. Dey, and R. K. Mudi, "Review on imc-based pid controller design approach with experimental validations," *IETE Journal of Research*, pp. 1–21, 2021.

[389] O. A. Somefun, K. Akingbade, and F. Dahunsi, "The dilemma of pid tuning," *Annual Reviews in Control*, 2021.

[390] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, pp. 661–670, 2010.

[391] V. S. Borkar, *Stochastic approximation: a dynamical systems viewpoint*, vol. 48. Springer, 2009.

[392] F.-S. Wang, W.-S. Juang, and C.-T. Chan, "Optimal tuning of pid controllers for single and cascade control loops," *Chemical Engineering Communications*, vol. 132, no. 1, pp. 15–34, 1995.

[393] Y. Lee, S. Park, and M. Lee, "Pid controller tuning to obtain desired closed loop responses for cascade control systems," *Industrial & engineering chemistry research*, vol. 37, no. 5, pp. 1859–1865, 1998.

[394] S. Song, W. Cai, and Y.-G. Wang, "Auto-tuning of cascade control systems," *ISA transactions*, vol. 42, no. 1, pp. 63–72, 2003.

[395] J.-C. Jeng and M.-W. Lee, "Identification and controller tuning of cascade control systems based on closed-loop step responses," *IFAC Proceedings Volumes*, vol. 45, no. 15, pp. 414–419, 2012.

[396] O. Çakıroğlu, M. Güzelkaya, and İ. Eksin, "Improved cascade controller design methodology based on outer-loop decomposition," *Transactions of the Institute of Measurement and Control*, vol. 37, no. 5, pp. 623–635, 2015.

[397] N. Manh, V. Diep, and P. Hung, "A synthesis method of robust cascade control system," *Journal of Automation and Control Engineering*, vol. 4, no. 2, pp. 111–116, 2016.

[398] M. Khosravi, V. Behrunani, R. S. Smith, A. Rupenyan, and J. Lygeros, "Cascade control: Data-driven tuning approach based on bayesian optimization," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 382–387, 2020.

[399] H. Jung, K. Jeon, J.-G. Kang, and S. Oh, “Iterative feedback tuning of cascade control of two-inertia system,” *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 785–790, 2020.

[400] Z. Y. M. Yingkui, “Application of delta v dcs in process control experimental device [j],” *Electrical Automation*, vol. 4, 2011.

[401] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, “A quantization-friendly separable convolution for mobilenets,” in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pp. 14–18, IEEE, 2018.

[402] S. A. Bortoff, P. Schwerdtner, C. Danielson, S. D. Cairano, and D. J. Burns, “H-infinity loop-shaped model predictive control with hvac application,” *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 2188–2203, 2022.

[403] S. J. Qin and T. A. Badgwell, “An overview of industrial model predictive control technology,” in *AIche symposium series*, vol. 93, pp. 232–256, New York, NY: American Institute of Chemical Engineers, 1971-c2002., 1997.

[404] J. B. Rawlings, “Tutorial overview of model predictive control,” *IEEE control systems magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[405] J. B. Rawlings, D. Angeli, and C. N. Bates, “Fundamentals of economic model predictive control,” in *2012 IEEE 51st IEEE conference on decision and control (CDC)*, pp. 3851–3861, IEEE, 2012.

[406] M. Ellis, J. Liu, and P. D. Christofides, “Economic model predictive control,” *Springer*, vol. 5, no. 7, p. 65, 2017.

[407] A. Alessio and A. Bemporad, “A survey on explicit model predictive control,” in *Nonlinear model predictive control*, pp. 345–369, Springer, 2009.

[408] A. Bemporad and M. Morari, “Robust model predictive control: A survey,” in *Robustness in identification and control*, pp. 207–226, Springer, 1999.

[409] A. H. González and D. Odloak, “A stable mpc with zone control,” *Journal of Process Control*, vol. 19, no. 1, pp. 110–122, 2009.

[410] E. F. Camacho and C. B. Alba, *Model predictive control.* Springer science & business media, 2013.

[411] G. Shah and S. Engell, “Tuning mpc for desired closed-loop performance for mimo systems,” in *Proceedings of the 2011 American Control Conference*, pp. 4404–4409, 2011.

[412] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, “Multi-objective and interactive genetic algorithms for weight tuning of a model predictive control-based motion cueing algorithm,” *IEEE Transactions on Cybernetics*, vol. 49, no. 9, pp. 3471–3481, 2019.

[413] M. Sahin, “Optimization of model predictive control weights for control of permanent magnet synchronous motor by using the multi objective bees algorithm,” in *Model-Based Control Engineering* (U. Z. A. Hamid and A. A. M. Faudzi, eds.), ch. 5, Rijeka: IntechOpen, 2021.

[414] D. Oliveira, E. M. G. Rodrigues, R. Godina, T. D. P. Mendes, J. P. S. Catalão, and E. Pouresmaeil, “Mpc weights tunning role on the energy optimization in residential appliances,” in *2015 Australasian Universities Power Engineering Conference (AUPEC)*, pp. 1–6, 2015.

[415] H. Waschl, D. Alberer, and L. del Re, “Automatic tuning methods for mpc environments,” in *Computer Aided Systems Theory – EUROCAST 2011* (R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, eds.), (Berlin, Heidelberg), pp. 41–48, Springer Berlin Heidelberg, 2012.

[416] J. E. W. Santos, J. O. Trierweiler, and M. Farenzena, “Robust tuning for classical mpc through the multi-scenarios approach,” *Industrial & Engineering Chemistry Research*, vol. 58, no. 8, pp. 3146–3158, 2019.

[417] N. He, M. Zhang, and R. Li, "An improved approach for robust mpc tuning based on machine learning," *Mathematical Problems in Engineering*, vol. 2021, 2021.

[418] W. Wojsznis, J. Gudaz, T. Blevins, and A. Mehta, "Practical approach to tuning mpc* *based on practical approach to tuning mpc by wojsznis, gudaz, mehta, and blevins, published in the proceedings of the isa 2001 conference, september 10-13, 2001, houston, tx [11].," *ISA Transactions*, vol. 42, no. 1, pp. 149–162, 2003.

[419] C. Ionescu and D. Copot, "Hands-on mpc tuning for industrial applications," *Bulletin of the Polish Academy of Sciences. Technical Sciences*, vol. 67, no. 5, 2019.

[420] G. A. Bunin, F. F. Tirado, G. François, and D. Bonvin, "Run-to-run mpc tuning via gradient descent," in *22nd European Symposium on Computer Aided Process Engineering* (I. D. L. Bogle and M. Fairweather, eds.), vol. 30 of *Computer Aided Chemical Engineering*, pp. 927–931, Elsevier, 2012.

[421] C. Rowe and J. Maciejowski, "Tuning mpc using h/sub /spl infin// loop shaping," in *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, vol. 2, pp. 1332–1336 vol.2, 2000.

[422] S. Di Cairano and A. Bemporad, "Model predictive control tuning by controller matching," *IEEE Transactions on Automatic Control*, vol. 55, no. 1, pp. 185–190, 2010.

[423] M. Mehndiratta, E. Camci, and E. Kayacan, "Automated tuning of nonlinear model predictive controller by reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3016–3021, 2018.

[424] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Reinforcement learning of the prediction horizon in model predictive control," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 314–320, 2021.

[425] B. Zarrouki, V. Klos, N. Heppner, S. Schwan, R. Ritschel, and R. Vosswinkel, “Weights-varying mpc for autonomous vehicle guidance: a deep reinforcement learning approach,” in *2021 European Control Conference (ECC)*, pp. 119–125, 2021.

[426] O. Dogru, K. Velswamy, F. Ibrahim, Y. Wu, A. S. Sundaramoorthy, B. Huang, S. Xu, M. Nixon, and N. Bell, “Reinforcement learning approach to autonomous pid tuning,” *Computers & Chemical Engineering*, vol. 161, p. 107760, 2022.