A Feasibility Study of Elementary Reinforcement Learning-Based Process Control

by

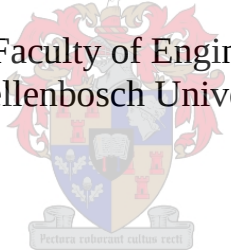
Edward Hendrik Bras

Thesis presented in partial fulfilment
of the requirements for the Degree

of

MASTER OF ENGINEERING
(CHEMICAL ENGINEERING)

in the Faculty of Engineering
at Stellenbosch University



Supervisor
Prof. T.M. Louw
Process Engineering
Stellenbosch University

Co-Supervisor(s)
Prof. S.M. Bradshaw
Process Engineering
Stellenbosch University

April 2022

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: April 2022

Plagiaatverklaring / Plagiarism Declaration

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is using ideas, material and other intellectual property of another's work and presenting it as my own.
2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.
3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.
4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly, all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel, of gedeeltelik, ingehandig het om enige kwaifikasie te verwerf nie.
   I declare that the work contained in this thesis/dissertation, except where otherwise stated, is my original work, and that I have not previously in its entirety, or in part, submitted this thesis/dissertation to obtain any qualification.

April 2022

ABSTRACT

The classical control paradigm is widely used in industry, has well-understood theoretical guarantees, and forms part of the foundational knowledge of chemical engineers. Challenging non-linear dynamics prevent its successful application in certain cases, while classical controllers cannot automatically accommodate changing closed-loop dynamics. Advances in computational capabilities have led to a significant research interest in the application of Reinforcement Learning (RL) to control processes. In RL, a computational agent interacts with an environment to maximise the cumulative scalar rewards received. It may be viewed as an alternative paradigm for control, as is done in this thesis, or as an approach to potentially enhancing the performance of classical controllers.

This simulation-based study's purpose is to investigate the feasibility of elementary RL techniques to automatically determine the final element adjustments in a single-loop RL-based control scheme. It places into context what the strengths and limitations are of using elementary RL to control processes and highlights nuances of RL-based control without trying to outperform classical control.

The control of a self-regulatory water tank model and the Van de Vusse reaction scheme model (used for benchmarking and requires advanced control solutions) were studied by applying two algorithms – Q-learning and SARSA – in a control scheme synthesized purely for theoretical study. Subsequently, these algorithms and the One-Step Actor-Critic algorithm were applied to the control of particle size in a qualitatively accurate grinding circuit model. All simulations leveraged the simplest possible RL design to allow interpretable and clear accounts of how these systems behave.

The results show that the use of elementary RL techniques to obtain interpretable RL-based controllers for simulation-based study worked well for the water tank and Van de Vusse reaction scheme models. This was not the case for the grinding circuit case study. Replacing the classical control paradigm is not likely using elementary RL. Significant safety concerns arise since large amounts of operational data may be required and insufficient training in certain regions of the state-action space leads to unpredictable control behaviour. The strengths and weaknesses of the algorithms studied were investigated. It is unlikely that a reduction of control loop specific tuning parameters in comparison to classical control will be realised in practical control problems by applying RL-based control. Where applicable, classical control outperformed the elementary RL-based controllers which stresses that algorithmic adjustments are required, as is recognised in state-of-the-art RL-based control approaches.

To conclude, the most practically feasible RL-based control solutions are likely to lie in the enhancement of existing control solutions by incorporating RL principles. The studied elementary RL-based control methods are not feasible for practical robust control. The control engineer must not be removed completely from the loop, and existing domain knowledge must be reconciled with computational thinking instead.

OPSOMMING

Die klassieke beheerparadigma word algemeen gebruik in industrie, het welbegrepe teoretiese waarborge, en vorm deel van die fundamentele kennis van chemiese ingenieurswese. Uitdagende nie-liniêre dinamika voorkom sy suksesvolle toepassing in sekere gevalle, terwyl klassieke beheerders nie outomaties veranderende geslote-lus dinamika kan akkommodeer nie. Vooruitgang in rekenvermoë het gelei tot ’n beduidende navorsingbelangstelling in die toepassing van versterkingsleer (RL) om prosesse te beheer. In RL reageer ’n rekenagent met ’n omgewing om die kumulatiewe skalaarbelongings ontvang, te maksimeer. Dit mag gesien word as ’n alternatiewe paradigma vir beheer, soos dit gedoen word in hierdie tesis, of as ’n benadering om die doeltreffendheid van klassieke beheerders potensieel te vergroot.

Hierdie simulasiegebaseerde studie se doel is om die uitvoerbaarheid van elementêre RL-tegnieke te ondersoek om outomaties die finale elementwysigings in ’n enkellus RL-gebaseerde beheerskema te bepaal. Dit plaas wat die sterkpunte en beperkinge is van die gebruik van elementêre RL om prosesse te beheer, in konteks, beklemtoon nuanse van RL-gebaseerde beheer sonder om die klassieke beheer te probeer oortref.

Die beheer van ’n self-regulerende watertenkmodel en die Van de Vusse-reaksieskemamodel (gebruik vir normstelling en vereis gevorderde beheeroplossings) is bestudeer deur twee algoritmes toe te pas – Q-leer en SARSA – in ’n beheerskema gesintetiseer uitsluitlik vir teoretiese studie. Vervolgens, is hierdie algoritmes en die “One-Step Actor-Critic”-algoritme toegepas op die beheer van partikelgrootte in ’n kwalitatiewe akkurate slypkringmodel. Alle simulasies het die eenvoudigste moontlike RL-ontwerp gebruik om interpreteerbare en duidelike rekenskap te gee van hoe hierdie sisteme hul gedra.

Die resultate het getoon dat die gebruik van elementêre RL-tegnieke om interpreteerbare RL-gebaseerde beheerders te verkry vir simulasiegebaseerde studie goed gewerk het vir die watertenk- en Van der Vusse-reaksieskemamodelle. Dit was nie die geval vir die slypkringgevallestudie nie. Om die klassieke beheerparadigma te vervang deur elementêre RL te gebruik, is nie waarskynlik nie. Beduidende veiligheidskommer kom voor aangesien groot hoeveelhede bedryfsdata vereis mag word en onvoldoende opleiding in sekere streke van die staataksiespasie na onvoorspelbare beheergedrag lei. Die sterkpunte en swakpunte van die algoritmes bestudeer is ondersoek. Dit is onwaarskynlik dat ’n reduksie van beheerlus spesifieke instemmingparameters, in vergelyking met klassieke beheer, gerealiseer sal word in praktiese beheerprobleme deur RL-gebaseerde beheer toe te pas. Waar toepaslik, het die klassieke beheer die elementêre RL-gebaseerde beheerders oortref, wat beklemtoon dat die algoritmiese wysigings vereis word, soos dit erken word in hipermoderne RL-gebaseerde beheerbenaderings.

Ter slotte, die mees praktiese uitvoerbare RL-gebaseerde beheeroplossings lê waarskynlik in die vergroting van bestaande beheeroplossings deur RL-beginsels te inkorporeer. Die bestudeerde RL-gebaseerde beheermetodes is nie uitvoerbaar vir praktiese robuste beheer nie. Die beheeringenieur moet nie heeltemal verwyder word uit die lus nie, en bestaande domeinkennis moet eerder met rekendenke versoen word.

ACKNOWLEDGEMENTS

The author would like to express his great appreciation to this project's contributors:

- My supervisors, Prof. Tobi Louw and Prof. Steven Bradshaw, for your guidance, advice, support, patience, constructive criticism, facilitation of my development as a postgraduate student, and believing in me throughout.

- Stellenbosch University's Department of Process Engineering and the Ernst and Ethel Eriksen Trust for financial assistance.

- Prof. Chris Aldrich for shedding further light on practical points regarding reinforcement learning for process control and the importance of relating results to a well-understood reference for context.

- Dr JP Barnard and Mr Gerhard van Wageningen for their assistance with the use of Stellenbosch University's high performance computing resources.

- My parents, Zelna and Johan, for their unwavering support and accommodation during all my endeavours. Also, for putting up with my reinforcement learning monologues during lockdown.

- My family for their support and motivation.

- My friends, Jason Pantony, Daneel Basson, Tristan Arendse, and Fabian Ferreira for providing support and combatting Covid-19 isolation throughout the pandemic.

- All academic contributors to the reinforcement learning field who have allowed me to appreciate, to an unprecedented extent, the wonders of mathematical modelling and computational thinking.

- Our Border Collie Dalmation mix, Benjamin, for reminding me that not all behaviours can be approximated computationally.

Contents

Nomenclature

Greek Symbols

α	step size hyperparameter
ε	the probability of taking a random action
$\eta(\boldsymbol{S}')$	the average number of time steps spent in the next observed state, $\boldsymbol{S}'$, per episode
γ	discount factor
$\boldsymbol{\phi}(\boldsymbol{S}), \boldsymbol{\phi}(\boldsymbol{S}, \boldsymbol{A})$	basis function vector that takes the currently observed state $\boldsymbol{S}$ as input (left), or the currently observed state $\boldsymbol{S}$ and action $\boldsymbol{A}$ as inputs (right)
$\mu(\boldsymbol{S})$	the fraction of time spent in each available observed state comprising the state space's approximation, $\boldsymbol{S} \in \mathcal{S}$
μ	sampling mean (normal distribution)
∇	a gradient with respect to a parameter vector
π	a policy (maps the observed state to an action)
ρ	liquid density
σ	standard deviation of a distribution
$\boldsymbol{\theta}$	parameter vector for the actor
τ	time constant
υ	rock fraction in the ore (grinding circuit model)

Other Symbols

$\boldsymbol{A}$	agent action (more than one component)
$\mathcal{A}$	action space
A_R, A_{tank}	surface area of reactor (left), or cross-sectional area of the tank (right)
$b(\boldsymbol{S})$	baseline function
C	concentration
$\mathfrak{C}$	a constant used to decay the probability of selecting a random action
C_p	liquid heat capacity
c_v	valve discharge coefficient
$E(t)$	error signal of control system at current time t
F_{in}	inlet volumetric flow rate (water tank model)
G	discounted return
g	gravitational acceleration
$G_d(s)$	disturbance transfer function in the Laplace domain
$G_p(s)$	process transfer function in the Laplace domain
H	height of liquid in the tank (water tank model)
ΔH	reaction enthalpy change (Van de Vusse reaction scheme model)
$h(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{\theta})$	preference function
J	parameter for slip jump
$J(\boldsymbol{\theta})$	scalar performance objective for policy gradient and actor-critic control algorithms
$\mathscr{k}$	lumped parameter proportional to the product of fraction valve opening and valve discharge coefficient
K_c	controller gain
k_1, k_2, k_3	specific reaction rates for reactions 1, 2, and 3
K_{PG}	the average length of an episode
M_T	arithmetic mean of the Bellman targets observed at the current state-action coordinate $(\boldsymbol{S}, \boldsymbol{A})$
m_C	coolant mass
$\dot{m}$	mass flow rate
$n(\mathcal{S}), n(\mathcal{A})$	cardinality of the state space (left), or cardinality of the action space (right)
$\mathscr{n}$	indicates Bellman target comprising more than two successive time steps' data

$\mathcal{O}(N)$	scaling complexity of code when evaluating N parameters
P_{mill}	mill power draw
$Q_\pi(\boldsymbol{S}, \boldsymbol{A})$	action-value function for policy π
R	scalar reward obtained by the agent
S	parameter for deadband plus stickband
$\boldsymbol{S}$	observed state
$\mathcal{S}$	state space
T	discrete time step in a Markov Decision Process
t	Bellman target for the action-value function, where the target is written using two successive discrete time steps
$\mathcal{T}$	outlet temperature of the process stream (Van de Vusse reaction scheme model)
ΔT	sampling period
$\mathcal{T}_C$	outlet temperature of coolant (Van de Vusse reaction scheme model)
$u(t)$	control signal generated by a control law written for analogue control
U	heat transfer coefficient between reactor and coolant (Van de Vusse reaction scheme model)
V	volumetric flow rate (grinding circuit model)
$\dot{V}$	volumetric flow rate (Van de Vusse reaction scheme model)
$V_\pi(\boldsymbol{S})$	state-value function for policy π
V_R	reactor volume (Van de Vusse reaction scheme model)
v_{mill}	volume of the mill (grinding circuit model)
$\boldsymbol{w}$	critic parameter vector
x	fraction valve opening
$\boldsymbol{X}(\boldsymbol{S}, \boldsymbol{A})$	basis function vector for the actor
$\mathcal{X}_i(\boldsymbol{S}, \boldsymbol{A})$	an individual element of $\boldsymbol{X}(\boldsymbol{S}, \boldsymbol{A})$

Acronyms

A2C	Advantage Actor-Critic
BC	steel ball consumption
CFF	cyclone feed flow
CSTH	Continuously Strirred Tank Heater
CV	controlled variable
DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
DQN	Deep Q-Network
DS-d	Direct Synthesis for Disturbances
DV	disturbance variable
FOPTD	First-Order-Plus-Time-Delay
FP	fines production
GP	Gaussian Process
GPI	Generalised Policy Iteration
GPPSTD	Gaussian Process Posterior Sampling Temporal Difference
GPTD	Gaussian Process Temporal Difference
HVAC	Heating, Ventilation, and Air Conditioning
IAE	integral of the absolute error
IMC	Internal Model Controller
ITAE	integral of time multiplied with the absolute error
MDP	Markov Decision Process
MFB	mill feed balls
MFLC	Model-Free Learning Control
MFS	mill feed solids
MFW	mill feed water
MIMO	multiple-input, multiple-output
MSE	Mean Squared Error
MV	manipulated variable
MW	molecular weight
NFQCA	Neural Fitted Q-Iteration with Continuous Actions
pdf	probability density function
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PILCO	probabilistic inference for learning control

pmf	probability mass function
PSE	particle size estimate
RBF	Radial Basis Function
RC	rock consumption
RL	Reinforcement Learning
SAG	semi-autogenous grinding
SANE	Symbiotic, Adaptive Neuro-Evolution
SARSA	State-Action-Reward-State-Action
SFW	sump feed water
SISO	single-input, single-output
SP	set point
$SVOL$	sump volume
TD	temporal difference
$TD(0)$	an algorithm that applies the principles of the SARSA algorithm to approximate the state-value function
TV	total MV variation (final element)
VSD	variable speed drive

CHAPTER 1

INTRODUCTION

The formulation of control laws for industrial servo (tracking of set point) and regulator (disturbance attenuation) problems is often based on the well-established classical framework. This foundation for control design has led to a stark contrast between control algorithms that are considered to be predominantly theoretical and those that are implemented by industrial practitioners. This is illustrated well by the ubiquity of the Proportional-Integral-Derivative (PID) and the Proportional-Integral (PI) control algorithms in industrial applications despite more advanced controllers developed in the academic literature. Model Predictive Control (MPC) is also widely implemented.

The identification of control objectives and available process handles, mapping of process variables to the Laplace domain (analytically or empirically), and subsequent control law tuning and implementation are steps that are often used when designing and deploying feedback controllers. These design steps have been covered extensively by Marlin (2000) and Skogestad and Poslethwaite (2005). Generally applicable controller design specifications that may be extracted from classical control theory have been summarized by Hafner and Riedmiller (2011). These are the necessity of nonlinear control laws, control law precision, coping with the effects of external variables, regulating long-term process dynamics, satisfactory servo problem behaviour, and robustness to model uncertainty or time-varying dynamics.

In chemical engineering, dynamic models of processes are useful tools when evaluating the theoretical and practical feasibilities of proposed control solutions. These dynamic process models combine fundamental laws of nature with assumptions that may be justified from an engineering perspective to approximate the time-dependent cause-effect relationships between the inputs and outputs of a chemical process. Mathematically, these are often developed using a combination of differential and algebraic equations and/or empirical modelling with time as the independent variable (McAvoy et al., 1972; Marlin, 2000; Le Roux et al., 2013; Felder et al., 2017).

Adaptive control is attractive to deal with issues arising from phenomena such as altered closed loop dynamics with root causes which may be challenging to find (Olsson and Åström, 2001; Shoukat Choudhury et al., 2005; Wakefield et al., 2018), difficulty in tuning many control loops (Marlin, 2000; Skogestad and Poslethwaite, 2005; Shipman and Coetzee, 2019), and challenging dynamics such as non-linearity (Wright and Kravaris, 2001; Hermansson and Syafiie, 2015) and inverse response (Marlin, 2000), within a unified framework.

This thesis uses the above framework as a starting point to conduct a simulation-based feasibility study of elementary, Reinforcement Learning – based control. The model-free Reinforcement Learning solution methodology was followed systematically to identify potential issues and nuances in the application of Reinforcement Learning to process control problems.

1.1 The Reinforcement Learning–Based Process Control Problem

Reinforcement Learning (RL) is a technique whereby a computational agent interacts with its environment (the RL environment) to maximize the expected value of a discounted return. This interaction process is known as training the RL agent. The agent is a probabilistic model for sequential decision making and is most often realised through the design, and updates to the parameters of, its RL representation. This representation can be anything from a discretized table to a deep neural network (Sutton and Barto, 2018). Mathematically, the vast majority of RL algorithms are founded on a discrete-time Markov Decision Process (MDP) (Section 2.14.1) approximation of the Dynamic Programming (DP) framework described by Bellman (1972). The agent's parameters become more suitable to solving a problem through experience. Its decision making is inherently non-linear and adaptive. These properties, accompanied by the availability of enhanced computational resources, allow for its application to both servo and regulator process control problems. This has resulted in an accompanying substantial research interest within the process control community.

Practical application using a purely online RL agent training approach without obtaining a first-guess policy offline will compromise the seven major categories of control objectives described by Marlin (2000). These are safety, environmental protection, protection of equipment, smoothness of production and operation of a plant, quality of the product, profit maximization, and monitoring and fault diagnosis. If an offline estimate of the RL agent's parameters is not generated, the RL agent will select random final element adjustments until it has gained sufficient experience. Therefore, generating an offline estimate of the RL agent's parameters must be investigated.

Elementary RL refers to the foundations of the field, and are the building blocks used as starting points for the development of state-of-the-art control approaches. The use of elementary RL algorithms is not consistent with the current trend in literature, which is to leverage modern computational capabilities to build highly adaptive RL representations and to study the best performance of the resulting agents. This involves advances in the RL field where the elementary building blocks are used and adapted to accommodate these RL representations. This thesis takes the stance that the following aspects of RL-based controller design must be prioritised:

1. How the steps in classical control design (identification of control objectives and available handles, mathematical modelling and transformation, control algorithm implementation) may be reconciled with the RL viewpoint.

2. Investigating the core limitations of elementary RL algorithms to establishing a reasonable and physically realisable adaptive control framework. This includes identification of subtleties arising and the practical limitations thereof. This is useful for stakeholders without a machine learning background and may hold implications for state-of-the-art applications that inherently rely on the foundations of RL.

1.2 Outcomes of this Work

The classical control paradigm motivates approaching the vast majority of process control problems, which are defined in the time domain, by first approximating a suitable linearized model of the process. This reduced complexity model is then mapped to the Laplace domain, where analytical and empirical procedures may be applied to solve pertinent tuning and stability problems. To investigate the feasibility of elementary RL-based control, the problem may be viewed in a similar light.

In RL, there are functions known as value functions and policies, which are used to transform time domain control data to domains for sequential decision making. The design of RL representations (Section 1.1) allows one to determine the degree of time domain complexity retained during data transformation. The RL algorithms relevant to this study are the State-Action-Reward-State-Action (SARSA), Q-learning, and One-Step Actor-Critic algorithms.

The aim of the study is to evaluate the feasibility of applying elementary RL techniques to automatically determine the optimal control actions for process control systems.

The objectives of this simulation-based study are as follows:

1. Synthesize a purely RL-based control methodology for single-input, single-output (SISO) problems that enables the testing of elementary RL techniques on simplified RL environments.

2. Investigate control behaviour as well as the effect of parameter selection and state-action space discretization fineness for tabular SARSA by studying a self-regulatory water tank (Case Study 1) and the control of desired product concentration in the Van de Vusse reaction scheme (Case Study 2).

3. Evaluate the feasibility of tabular Q-learning, tabular SARSA, and One-Step Actor-Critic agents for the control of product particle size in a ball mill grinding circuit (Case Study 3).

Objective 1 is aimed at establishing a synthesized controller that is used for investigation throughout the thesis. To validate the methodology used for control, the controller must adhere to two criteria. Firstly, application to a case study representing an easily controlled process must allow for a full description of the mechanism by which qualitatively reasonable control performance may be achieved. Secondly, application to an appropriate benchmark process must yield satisfactory results. The results of other researchers should be replicated to the extent possible. This validation forms the first part of objective 2. Objective 2 further aims to better understand which parameters (including both tuning parameters and RL hyperparameters) need to be tuned specifically for the process being controlled and the effects of state-action discretization used in the synthesized controller.

Objective 3 exposes the synthesized controller to a simulated system representative of a robust control problem often encountered in the mineral processing industry. The term “robust” is used in its capacity to describe uncertainty about the nominal plant model (Skogestad and Poslethwaite, 2005). In addition to

SARSA, Q-learning (for training on historical data) and One-Step Actor-Critic algorithms are also applied to this problem. As a well-understood quantitative reference, all three control laws generated by the elementary RL agents are compared to a continuous PID algorithm. This aids with obtaining a well-rounded understanding of the elementary aspects of RL-based process control, formalising qualitative comparisons between the studied RL agents, placing in perspective the challenges that prevent industrial RL-based control at present, and proposing directions for future research efforts.

To achieve the objectives, the following questions must be answered:

1. What are the limits of tabular Q-learning and tabular SARSA in RL-based process control?
2. What are the limits of One-Step Actor-Critic in RL-based process control?
3. Can control problem complexity be practically reduced for an RL agent by simplifying the RL environment, e.g. by discretizing the state-action space?

1.3 Scope

At this stage, it is essential to clearly define what is considered relevant to the application of elementary RL in this thesis. The study does not attempt to outperform classical control for the grinding circuit model that serves as case study, to develop a new practical control approach, or to extensively explore the application of the possible algorithmic adjustments to the most elementary RL algorithms. Rather, the study will explore the properties of the most interpretable and simplest RL agents when applied to process control problems to provide an account of the feasibility, nuances, and challenges of elementary RL-based process control.

The properties of SARSA are particularly amenable to mathematical manipulation and this algorithm naturally forms a central part of the aim, objectives, and key questions. Therefore, the focus on the One-Step Actor-Critic algorithm is markedly less than the focus on SARSA. Both algorithms play important roles in the grinding circuit simulation study conducted.

The deep learning paradigm applied to Actor-Critic algorithms has enjoyed much attention in the field of RL-based process control (Chapter 3). Significantly less attention has been paid to the elementary algorithms of RL. In the opinion of the author, this may mostly be attributed to known algorithmic deficiencies of the latter and the attention drawn to deep learning methods owing to the advances in computing capabilities coupled with their ability to cope with very complex and high-dimensional problems in an efficient manner. While these are valid reasons for pursuing state-of-the-art methods, a significant gap in the literature is brought about by the shortage of texts that focus on gradually building up complexity, and providing a well-rounded understanding of the nuances of RL-based process control that arise at the field's foundations. Such research provides both a framework where impediments to industrial implementation may be identified and a point of departure from which new research directions may be identified.

In this thesis, discrete action selections are made available to SARSA, Q-learning, and One-Step Actor-Critic agents. In addition to the points mentioned above, the path of increased resistance from the perspective of the RL-based control research community is also purposefully chosen so that algorithms are studied that improve interpretability for wider adoption. The word interpretable is used in its capacity to describe a control solution for which the output may be predicted by the designer through the use of logical statements and domain knowledge (Zhang et al., 2021). Low interpretability of state-of-the-art approaches contributes to industrial scepticism regarding RL-based control. Despite the study of interpretable RL agents, the envisaged interactions between production engineers and these elementary RL-based controllers remain challenging since the behaviour of such controllers are not always easily predictable and these interactions require significant field-specific knowledge.

1.4 Thesis Contributions

Through the application of systematic model-free RL solution methodology to selected simulated process control case studies, this thesis contributes the following to the field of RL-based process control:

1. Synthesis and validation of a reasonable model-free RL control design for SISO control loop pairing. *Reasonability* is defined for this thesis as the property of being suitable for the study of the behaviour of an RL agent, irrespective of the complexity of the agent, when applied to the control of a *simulated* process.

2. Characterization of tuning parameters, RL hyperparameters, and RL representation design for a tabular SARSA agent. A SARSA agent interacting with a discretized state-action space has unique theoretical properties that are worth exploring. Doing so provides a clear relation of what is purely *theoretically* achievable versus what is *practically* achievable with RL-based controller tuning.

3. Critical assessment of the ability of a tabular SARSA agent to achieve physically realisable feedback control and the implications of using Q-learning for training on historical data. In the presence of instrumentation constraints in real industrial processes, it is vital to consider the extent to which algorithmic deficiencies are detrimental to such a controller.

4. Drawing qualitative comparisons between SARSA, Q-learning, and One-Step Actor-Critic agents. This requires a clear understanding of the qualitative properties of the learning processes of each agent, which are also investigated.

5. Using the lessons learned by studying elementary RL-based control to make recommendations for future research.

1.5 Thesis Outline

Chapter 2 describes the necessary theoretical framework for the work presented. Subsequently, Chapter 3 reviews pertinent literature which lays the foundation for the methodology used to achieve the aim and

objectives of the study (Chapter 4 and Chapter 5). Chapter 6 conveys the results of the analyses and the critical assessment thereof. The culmination of the arguments and findings of this thesis is conveyed in Chapter 7 which also relates the author's recommendations for future work in the field.

CHAPTER 2

THEORETICAL FRAMEWORK

Of particular relevance to the current chapter are the algorithmic principles of value-based, Policy Gradient, and Actor-Critic control methods. In RL-based control, we are interested in identifying the parameters that best describe the action-value function for value-based RL (SARSA and Q-learning), or in directly parameterizing the optimal policy- and critic functions for Actor-Critic RL (One-Step Actor-Critic).

Appendix A provides a worked example where SARSA is applied to the control of liquid level in a mixing tank. The author recommends working through the material in Appendix A after reading Section 2.3 to provide a practical example and after reading this chapter to reconcile theory with the example.

2.1 Overview of the RL-Based Control Process

A generic illustration of RL-based control where an agent only selects one action at each time step is given in Figure 1. The process may be described as an RL agent interacting with an RL environment and occurs during the progression of discrete time steps. At each discrete time step, the agent applies an action to the RL environment which then undergoes a state transition and gives the agent a scalar reward. This establishes a causality between the agent's action selections and the responses of the RL environment.

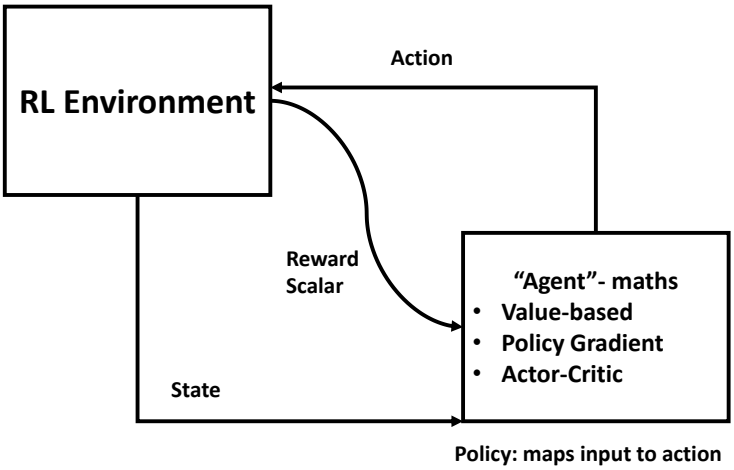


Figure 1: Generic flow diagram depicting agent-environment interaction for RL-based control

The agent's only goal is to maximise the cumulative rewards received from the RL environment by adjusting the parameters of its RL representation. By doing so, a policy is learned which dictates how each observed state (input) is mapped to the next selected action (output).

The agent is mathematically formulated so that its behaviour improves during the interaction and may establish a policy in one of three ways. The first is through learning a unique value function (value-based), the second by maximizing a scalar performance objective in the parameter space (Policy Gradient), and the third combines these two approaches to improve the learning properties of the second approach (Actor-Critic).

As will be seen during the literature review of Chapter 3, there are primarily three avenues of researching the RL-based controller problem. The first requires the agent to select the actions to control the plant when presented with the measurements provided by the available instrumentation (all sources of raw operational information provided), the second requires the agent to automatically tune classical controllers to enable control, and the third involves reducing the problem size through simplification. The third avenue is selected in this study.

Aspects such as operational data requirement and ease of tuning cannot easily be improved practically by applying deep learning techniques with potentially excessive numbers of parameters or providing all available plant measurements to the agent.

In the deep learning paradigm, a particular concern preventing industrial RL-based control is the interpretability of deep neural networks (Zhang et al., 2021). This further motivates the study of alternative approaches to deep neural networks to represent the required policies and value functions for process control problems. Applying elementary RL to process control problems is a natural step in the search for more interpretable RL-based process control solutions.

2.2 Preliminary RL Terminology

Unless stated otherwise, these definitions may be found in the work of Sutton and Barto (2018).

With the exception of the RL agent, all exogenous inputs, disturbances, and signal processing that are relevant to the process control problem constitute the *RL environment*. *Update rules* adjust the parameters that make up the model used for sequential decision making and are derived so as to always improve the agent's solution to the control problem as training progresses. The process control problem as a whole is formulated as an *MDP* (Section 2.14.1) with each *discrete time step* denoted by T. At each time step T, the agent is allowed to apply an *action* $(\boldsymbol{A})$ to the RL environment. The agent then moves from the current *state* $(\boldsymbol{S})$ which is observed at time step T to the *next state* $(\boldsymbol{S}')$ at time step $T+1$. All references to states refer to the observed states that are actually available to the agent.

Based on the new state $\boldsymbol{S}'$, a scalar signal known as the *reward* (R) is provided to the agent. The reward is used to perform the algorithmic update(s) that adjusts the probability of selecting the same action at the previous state $\boldsymbol{S}$. Through this stochastic and iterative interaction process, a parameter search is instantiated.

For process control problems, the final element adjustments applied to the process are determined by the actions selected by the RL agent. In general, $\boldsymbol{A}$ may be a scalar or a vector depending on the number of process inputs that must be manipulated by the agent at each time step T. Similarly, $\boldsymbol{S}$ and $\boldsymbol{S}'$ may be either scalars or vectors and must be defined to inform the agent of its position in the process operating window relative to where it needs to be to solve the process control problem. The *state space* is denoted by $\mathcal{S}$ and the *action space* by $\mathcal{A}$. The *state-action space* comprises the state space augmented to include information about the action space. These spaces refer to the vector spaces underlying the available states and actions, and are modelled either as discretized or continuous when defining possible values of $\boldsymbol{S}$ and $\boldsymbol{A}$ (Deisenroth et al., 2020).

An agent may be trained using an on-policy or an off-policy algorithm. An *on-policy* algorithm interacts directly with the RL environment, while an *off-policy* algorithm adjusts RL representation parameters using historical data. On-policy training is therefore online, while off-policy training is offline.

Exploration-exploitation refers to the need for any computational agent to maintain a balance between encountering new regions of the state-action space that have unknown consequences and utilising knowledge already gained. The agent achieves sequential decision making through the application of exploration-exploitation rules to the RL representation. This statistical model which maps the currently observed state $\boldsymbol{S}$ to the appropriate action $\boldsymbol{A}$ to take is the *policy*, $\pi(\boldsymbol{S}) \rightarrow \boldsymbol{A}$. An RL agent has a *stationary policy* π *if* update rules are not applied to it, or if the parameters of the RL representation have already converged (Engel, 2011). The term "*exploration strategy*" refers to the approach used to ensure that exploration and exploitation are balanced.

Using a *table* as an RL representation implies that the states and actions are discrete. Depending on the numbers of components defining the states and actions (i.e., depending on whether $\boldsymbol{S}$ and $\boldsymbol{A}$ are scalars or vectors), such a table may take the form of a hypervolume. A hypervolume as RL representation is therefore simply a table in higher dimensions.

Each RL problem has underlying *value functions*, and the RL representation constrains to what extent the required *optimal value function* may be learned after sufficient training. The underlying value-functions are known as the state-value function and the action-value function. The state-value function contains the values associated with each possible observed state, while the action-value function contains the values associated with taking each available action at each possible observed state. The values provided by these functions are the relative potentials for increased reward yield. During an RL agent's training, each value function instance depends on the corresponding instance of the policy.

2.3 Formulising an RL Agent's Objective

Irrespective of the type of RL agent used, the agent must maximise the cumulative rewards received during its interaction with the RL environment. The discounted return, denoted by G, is used to help the RL agent reach increased rewards. It is sampled at every time step of the MDP and is defined in the time domain by

Equation [1], where $t = i\Delta T$ for $i \in \mathbb{N}$ and sampling period ΔT (Sutton and Barto, 2018). The time step of sampling is shown in the subscript of G in Equation [1]. The sampling period between measurements, ΔT, is the actual time in the RL environment associated with a transition between two successive time steps in the MDP.

$$G_t \doteq \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+(i-1)\Delta T} \qquad [\ 1\ ]$$

When indicating the time at which a reward R is received, the convention used in this thesis is that each reward's subscript denotes the time at which it was received by the RL agent, which corresponds to the time at which the RL agent leaves the current time step. Therefore, R_t in Equation [1] denotes the scalar reward received by the RL agent when $i = 1$. Equation [1] can be rewritten in terms of the discrete time steps of an MDP, as shown in Equation [2]. The sampling period ΔT is the same between successive time steps in Equation [2] (a transition from T to $T + 1$), and any one of the discrete time steps of the MDP may be referred to as an instance of T. The reward R obtained at the MDP time step associated with the instance of T may be denoted by R_T.

$$G_T \doteq R_T + \gamma R_{T+1} + \gamma^2 R_{T+2} + \cdots \qquad [\ 2\ ]$$

The hyperparameter $\gamma \in [0,1]$ is known as the discount factor (Sutton and Barto, 2018). The properties of the value functions (Section 2.5) rely on contraction mapping theory which also ensures that the value functions are unique for each policy (Hunter and Nachtergaele, 2005; Szepesvári, 2010; Sutton and Barto, 2018). As $\gamma \to 0$, the agent prioritizes immediate rewards. Long-term rewards are prioritised if $\gamma \to 1$. The agent's policy π determines the actions applied to the RL environment which, in turn, affect the rewards obtained during agent-RL environment interaction (Sutton and Barto, 2018).

Any RL-based control solution therefore relies on the designer to decide when certain rewards are provided to the agent based on the observed state $\boldsymbol{S}$ (reward function), which RL agent is applied, and how the policy π must be instantiated using the RL representation(s). The outcome of these design decisions must be that the problem is solved by the agent to the satisfaction of the designer. To solve the control problem, the agent must be provided with sufficient and appropriate operational data.

These design attributes indicate an opportunity, albeit limited, to inject domain knowledge in an RL-based controller even in the model-free context. Importantly, model-free methods do not facilitate the incorporation of such knowledge in the update rules applied during the parameter search of a given agent type. Rather, the RL representation's design before training, the definitions of $\boldsymbol{S}$ and $\boldsymbol{A}$, and the reward function allow for some extent of domain knowledge to be incorporated in the design.

2.4 The Bellman Equation Written for the Optimal Value Functions

The starting point for RL-based control is the Bellman equation for the true, optimal state-value function which is central to the DP methods described by Bellman (1972). Jaakkola et al. (1994) summarise the qualitative motivation for the Bellman equation. Assume, for now, that the probabilistic model $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$

describing the transition from $\boldsymbol{S}$ at T to $\boldsymbol{S}'$ at $T+1$ as a consequence of each available action in the action space, $\boldsymbol{A} \in \mathcal{A}$, is known. If both the action space $\mathcal{A}$ and the state space $\mathcal{S}$ are modelled using discretized actions and states (to prevent intractable computation), Bellman's equation for optimal state-value relates the reward received from the RL environment to the optimal state-value function $\left(V_\pi^*(\boldsymbol{S})\right)$ evaluated at the current state $\boldsymbol{S}$ and is given in a very accessible form by Jaakkola et al. (1994).

This form is shown in Equation [3] and is evaluated for a stationary policy π. Such a policy does not necessarily imply the optimal policy π^*, merely that the policy evaluated must not change with time. Such a policy could be obtained by evaluating a classical control law with constant tuning parameters provided that the incorporation of historical information through the integral mode is not perceived as a non-stationary policy when each state and action is mapped to the discretized state-action space of the RL representation, stopping the training process of an RL agent, or evaluating the optimal policy π^*.

$$V_\pi^*(\boldsymbol{S}) = \max_{\boldsymbol{A} \in \mathcal{A}} [R_T + \gamma \textstyle\sum_{\boldsymbol{S}' \in \mathcal{S}} Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})\, V_\pi^*(\boldsymbol{S}')] \qquad [\,3\,]$$

The second term in the operand of Equation [3] is the expected discounted return when following the stationary policy after the current time step. The random variable $\boldsymbol{S}' \sim Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ is the input to the optimal state-value function at the next state, $V_\pi^*(\boldsymbol{S}')$, and selecting the maximum numerical value of the operand across all available actions at each time step T in Equation [3] ensures that the entries of $V_\pi^*(\boldsymbol{S})$ correspond to the maximum accumulation of rewards.

The contribution of the most recent agent action to the discounted return G is largely captured by the reward which is added in the operand, while the second term is discounted through multiplication with γ. Hence, consistency with Equations [1] and [2] is ensured – only time steps T and $T+1$ under stationary policy π are relevant in Equation [3] and this requires the incorporation of γ^1. While the stationary policy is not necessarily discretized, the evaluation of $V_\pi^*(\boldsymbol{S})$ is defined in terms of the discretized state-action space to ensure computationally tractable summation in Equation [3].

The Bellman equation applied to the optimal action-value function for policy π, $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$, is defined as the operand of Equation [3] (Jaakkola et al., 1994; Sutton and Barto, 2018). This is shown in Equation [4]. It can be observed that information regarding the action $\boldsymbol{A}$ selected by the agent is explicitly incorporated in the definition, making it suitable for solving value-based *control* problems. By substituting Equation [4] into Equation [3], Equation [5] is obtained (Jaakkola et al., 1994). Equation [4] and Equation [5] clarify that the properties of state-value functions arising from contraction mapping theory directly extend to action-value functions.

$$Q_\pi^*(\boldsymbol{S}, \boldsymbol{A}) = R_T + \gamma \textstyle\sum_{\boldsymbol{S}' \in \mathcal{S}} Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})\, V_\pi^*(\boldsymbol{S}') \qquad [\,4\,]$$

$$V_\pi^*(\boldsymbol{S}) = \max_{\boldsymbol{A} \in \mathcal{A}} [Q^*(\boldsymbol{S}, \boldsymbol{A})] \qquad [\,5\,]$$

In process control problems, irrespective of the RL representation's design, solving Equation [4] quickly increases computational expense to an unreasonable extent owing to the number of possible states comprising the state space $\mathcal{S}$. This computational intractability, and the unreasonable assumption of obtaining a suitable probabilistic model $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ beforehand, are two key motives for approaching the solution of the Bellman equations using stochastic algorithms that are executed iteratively. Such stochastic approximation to DP principles is what gives rise to RL algorithms. The optimal value functions exist, are unique, and convergence is guaranteed after deriving a suitable RL update rule(s) (Engel, 2005; Hunter and Nachtergaele, 2005; Szepesvári, 2010).

2.5 Representing Future Rewards

While Equations [4] and [5] describe the optimal value functions corresponding to the optimal policy π^* (or another stationary policy), value functions exist for any given non-stationary policy π. Equation [6] – as described by Sutton and Barto (2018) – shows that the action-value function $\left(Q_\pi(\boldsymbol{S}, \boldsymbol{A})\right)$ for policy π is the expected value of the discounted return sampled at time step T (Equation [2]) conditioned to the current state-action coordinate $(\boldsymbol{S}, \boldsymbol{A})$ for the policy instance π. The state and action are sampled at MDP time step T, and are denoted by $\boldsymbol{S}_T$ and $\boldsymbol{A}_T$, respectively. The value-based agent approximates the action-value function numerically, and thereby tries to maximise the cumulative rewards it receives. In the tabular case, the RL agent stores the estimated discrete action-value function in a table format $\left(\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})\right)$.

$$Q_\pi(\boldsymbol{S}, \boldsymbol{A}) \doteq \mathbb{E}_\pi[G_T | \boldsymbol{S}_T = \boldsymbol{S}, \boldsymbol{A}_T = \boldsymbol{A}] \; \forall \; \boldsymbol{S} \in \mathcal{S} \; and \; \boldsymbol{A} \in \mathcal{A} \qquad [6]$$

The uniqueness of the action-value function ensures that, for a given exploration strategy, the action-value function $Q_\pi(\boldsymbol{S}, \boldsymbol{A})$ for the optimal policy π^* is associated with a maximization of cumulative rewards across the state-action space, and is therefore the optimal action-value function $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$. Different discretizations of a tabular RL representation provide different resolutions to the optimal action-value function $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$. This is because, when the RL agent's parameters converge, the attributes of the unique $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$ for optimal policy π^* are captured to an extent dependent upon the RL representation design used.

Recall from Section 2.4 that the action-value function differs from the state-value function by augmenting the state space to include information about the different actions available to the agent. The state-value function is defined as shown in Equation [7]. For model-free, value-based control, $Q_\pi(\boldsymbol{S}, \boldsymbol{A})$ *must* be used. Both value functions are used to represent future rewards that are associated with following a policy π.

$$V_\pi(\boldsymbol{S}) \doteq \mathbb{E}_\pi[G_T | \boldsymbol{S}_T = \boldsymbol{S}] \; \forall \; \boldsymbol{S} \in \mathcal{S} \qquad [7]$$

During training, the RL agent seeks to update the policy π towards the optimal policy π^* as the approximated value function is updated. The optimal policy π^* ultimately learned by a value-based agent is the one obtained by using the *approximation* of the unique optimal action-value function $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$ as a

map from the current state-action coordinate to the most beneficial action to apply to the RL environment. This is achieved by applying an exploration strategy (which is also essential during agent training) to the approximation $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$. The optimal policy π^{*} obtained in this fashion is characteristic of value-based RL control methods.

The statistical conditioning of Equation [6] implies that the expected numerical values of G are dependent on the state-action coordinates encountered. Consequently, the frequencies with which these coordinates within the process operating window are experienced during training, and the RL representation's design, have very significant impacts on the range of the state-action space for which the agent can make sensible decisions. The fraction of the state-action space for which the RL representation has fully converged may be referred to as the *coverage* of the value-based algorithm.

Formally, the RL representation of a value-based method is the data transformation applied to each instance of $(\boldsymbol{S}, \boldsymbol{A})$ to obtain an appropriate real, scalar approximation to Equation [6] or Equation [7]. Equation [6] and Equation [7] are not solved analytically at any point.

Recall from Section 2.4 that a stochastic approximation approach is sought in RL as $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ is not known. This problem is addressed in RL by iteratively solving the DP problem posed by the Bellman equations for the optimal value functions, Equations [4] and [5], by sampling a target function at each discrete time step T. The target function is an approximation of G (Equation [2]) and is generated using available information. Equations [6] and [7] make this approach possible.

2.6 The Policy Improvement Theorem

The policy improvement theorem described by Sutton and Barto (2018) is of interest when studying value-based RL control. When we consider a deterministic policy π at time step T and its updated version π' at time step $T + 1$, the policy improvement theorem states that π' will be associated with a larger state-value evaluated at the currently observed state $\boldsymbol{S}$ ($V_{\pi}(\boldsymbol{S}) \leq V_{\pi'}(\boldsymbol{S})$). This implies that π' is a better policy for sequential decision making when interacting with the RL environment, and that it aids with the maximisation of the cumulative rewards received by the RL agent. If the value functions – state-value function $V_{\pi}(\boldsymbol{S})$ and action-value function $Q_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ – were considered for the same deterministic policy π, they would be related through Equation [8] with $\pi(\boldsymbol{S}) \rightarrow \boldsymbol{A}$.

$$V_{\pi}(\boldsymbol{S}) = Q_{\pi}\big(\boldsymbol{S}, \pi(\boldsymbol{S})\big) \qquad [\,8\,]$$

For RL-based control, the implication of the theorem is that improving the approximation to the unique $Q_{\pi}^{*}(\boldsymbol{S}, \boldsymbol{A})$ is guaranteed to improve the agent's behaviour for a tabular RL representation. The policy improvement theorem may readily be illustrated by using recursive substitutions of Equation [10] into the inequality shown in Equation [9]. This is true irrespective of the reward function used since it is the relative magnitudes of the rewards obtained that are important to the RL agent. The policy improvement theorem does not, strictly speaking, hold true when an RL representation other than a table is used. This

is because the specific form of the policy π becomes dependent on the design of the RL representation – the designer may unknowingly prevent certain regions of $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$ to be approximated (Sutton and Barto, 2018). The summation $R_T + \gamma V_\pi(\boldsymbol{S}')$ in Equation [10] is an approximation of the discounted return sampled at time step T (G_T in Equation [2]).

$$V_\pi(\boldsymbol{S}) \leq Q_\pi\big(\boldsymbol{S}, \pi'(\boldsymbol{S})\big) \qquad [\ 9\ ]$$

$$Q_\pi(\boldsymbol{S}, \boldsymbol{A}) = \mathbb{E}_\pi[R_T + \gamma V_\pi(\boldsymbol{S}') | \boldsymbol{S}_T = \boldsymbol{S}, \boldsymbol{A}_T = \pi'(\boldsymbol{S})] \qquad [\ 10\ ]$$

2.7 Policy Iteration, Value Iteration, and Generalized Policy Iteration

The fact that G may be computed recursively, accompanied by $V_\pi^*(\boldsymbol{S})$ as a γ-contraction, allows for iterative approximation of $V_\pi(\boldsymbol{S})$ or $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ to be used in prediction or control problems, respectively (Jaakkola et al., 1994; Szepesvári, 2010; Sutton and Barto, 2018). The prediction problem refers to computing an approximate state-value function for a stationary policy π, while the control problem refers to approximating the optimal policy π^*. In DP, where knowledge of $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ is required, there are two key approaches to leveraging recursive computation to approximate the optimal policy π^* (Sutton and Barto, 2018). The RL environment needs to be modelled as finite to ensure computational tractability in DP – $\mathcal{S}$, $\mathcal{A}$, and $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ of the RL environment all need to be represented as finite for DP methods.

The first DP approach is policy iteration which entails generating successive improved instances of $V_\pi(\boldsymbol{S})$ and π by repeatedly applying policy evaluation and policy improvement steps, both of which directly rely on knowing $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$. Policy evaluation updates $V_\pi(\boldsymbol{S})$ to correspond to π, while policy improvement updates π to π' using the current state-value estimate $V_\pi(\boldsymbol{S})$. After this, policy evaluation is applied again to estimate $V_{\pi'}(\boldsymbol{S})$, after which policy improvement is applied again. This process is continued until the optimal policy π^* and its corresponding value function $V_\pi^*(\boldsymbol{S})$ are approximated. Starting each policy evaluation step with the newest $V_\pi(\boldsymbol{S})$ helps with preventing excessive computational complexity if the changes in the state-value functions during successive policy evaluation steps are little.

The parameters used to describe $V_\pi(\boldsymbol{S})$ need not converge during each policy evaluation step to ensure ultimately finding the optimal value function $V_\pi^*(\boldsymbol{S})$ and the corresponding optimal policy π^*. This leads to the second DP approach to control, value iteration. This approach alleviates the use of multiple sweeps through the state space $\mathcal{S}$ by truncating policy evaluation. This is achieved by applying Equation [3] as an update rule and only performing one sweep through $\mathcal{S}$ per policy evaluation step, and therefore only one parameter adjustment per possible instance of $\boldsymbol{S}$ (Jaakkola et al., 1994; Sutton and Barto, 2018).

In contrast to DP approaches to control, RL control methods establish decision making without knowledge of $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ through Generalized Policy Iteration (GPI), and can deal with both finite and non-finite MDPs. At each time step T of an RL control problem (or at the end of each instance of a fixed batch of training steps for an RL agent based on the principles of Monte Carlo analysis), $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ is altered to better

approximate the true action-value function, $Q_\pi(\boldsymbol{S},\boldsymbol{A})$, at the current policy π. After this, π is improved with regard to the updated $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$. Note that, for a fixed exploration strategy and a value-based RL agent, the update to $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$ implicitly ensures that the current policy π is more suited to the current $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$. This is because π is instantiated by applying an exploration strategy to $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$ in value-based control. When both $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$ and π do not change during sequential updates, the optimal action-value function $Q_\pi^*(\boldsymbol{S},\boldsymbol{A})$ has been approximated within the constraints of the RL representation design used and the coverage permitted by the training of the agent.

2.8 Value-Based RL Control Algorithms

2.8.1 SARSA

A numerical approximation of the optimal action-value function $Q_\pi^*(\boldsymbol{S},\boldsymbol{A})$ may be obtained through the application of SARSA as an on-policy, value-based RL control method. This RL algorithm illustrates the principles of value-based RL control well at an elementary level.

The training process follows the episodic viewpoint of RL agent training, where a fixed number of transitions from T to $T+1$ are predefined as an episode. Each episode starts with $T=1$. The key properties of the episodic viewpoint for value-based control are that it is not assumed beforehand that returns may be identified and averaged for each individual state, and that it is assumed that the RL problem is finite in the number of MDP time steps (Sutton and Barto, 2018). The interaction between the agent and the RL environment therefore consists of a number of episodes, and each episode consists of a number of steps. A SARSA agent solves the RL control problem using the method described below (Poole and Mackworth, 2010; Sutton and Barto, 2018).

Let t denote the Bellman target, Equation [11], updated at time step T. The Bellman target t is a biased estimate of the discounted return G, Equation [2], and is updated based on the scalar action-value sample Q_{T+1}. The target considered is known as a one-step return, since only two adjacent MDP time steps are used to calculate the target value at each instance of T and $T+1$. The Bellman target is not stationary and this is a characteristic of RL problems – each GPI iteration changes the target's numerical value during the agent's interaction with the RL environment. For sufficiently large T, we may use the arithmetic mean of all available t observed at the state-action coordinate $(\boldsymbol{S},\boldsymbol{A})$ as an unbiased estimator of G at the current time step. This mean is given by Equation [12], and the initial manipulations are given by Equations [13] through [16]. Each subscript in Equations [11] through [16] may be converted to the time domain by multiplying it with ΔT.

$$t(\boldsymbol{S},\boldsymbol{A}) = R_T + \gamma Q_{T+1} \qquad [\ 11\ ]$$

$$M_T(\boldsymbol{S},\boldsymbol{A}) = \frac{\left(t_1(\boldsymbol{S},\boldsymbol{A}) + t_2(\boldsymbol{S},\boldsymbol{A}) + \cdots + t_T(\boldsymbol{S},\boldsymbol{A})\right)}{T} \qquad [\ 12\ ]$$

$$TM_T(\boldsymbol{S},\boldsymbol{A}) = \sum_{i=1}^{T} R_i + \gamma \sum_{i=1}^{T} Q_{i+1} \qquad [\ 13\ ]$$

$$TM_T(\boldsymbol{S},\boldsymbol{A}) = (R_T + \gamma Q_{T+1}) + \left(\sum_{i=1}^{T-1} R_i + \gamma \sum_{i=1}^{T-1} Q_{i+1}\right) \quad [\ 14\ ]$$

$$TM_T(\boldsymbol{S},\boldsymbol{A}) = (R_T + \gamma Q_{T+1}) + (T-1)M_{T-1}(\boldsymbol{S},\boldsymbol{A}) \quad [\ 15\ ]$$

$$M_T(\boldsymbol{S},\boldsymbol{A}) = \left(\frac{R_T + \gamma Q_{T+1}}{T}\right) + \left(1 - \frac{1}{T}\right) M_{T-1}(\boldsymbol{S},\boldsymbol{A}) \quad [\ 16\ ]$$

By letting $\alpha_T = \frac{1}{T}$ and rearranging, we may write Equations [17] and [18].

$$M_T(\boldsymbol{S},\boldsymbol{A}) = \alpha_T(R_T + \gamma Q_{T+1}) + (1-\alpha_T)M_{T-1}(\boldsymbol{S},\boldsymbol{A}) \quad [\ 17\ ]$$

$$M_T(\boldsymbol{S},\boldsymbol{A}) = M_{T-1}(\boldsymbol{S},\boldsymbol{A}) + \alpha_T\big([R_T + \gamma Q_{T+1}] - M_{T-1}(\boldsymbol{S},\boldsymbol{A})\big) \quad [\ 18\ ]$$

It should be realised that $M_T(\boldsymbol{S},\boldsymbol{A})$ is updated recursively, which implies that it may be interpreted as an update rule and may be written in terms of the average corresponding to the $(\boldsymbol{S},\boldsymbol{A})$ coordinate encountered during time step T. Further, $Q_{T+1} = Q(\boldsymbol{S}',\boldsymbol{A}')$ for table entry of the RL representation $Q(\boldsymbol{S}',\boldsymbol{A}')$ observed at the next state-action pair $(\boldsymbol{S}',\boldsymbol{A}')$ since the Bellman target t bases its biased estimate of the return G on the immediate reward and the action-value sample at the next MDP time step (weighted by the discount factor γ).

Herewith, the update rule used in the SARSA algorithm reported by Sutton and Barto (2018), is obtained – Equation [19]. Since α_T decreases as T increases, updates occurring later during a training episode carry less weight. It is, however, logical that the importance of information obtained later during an episode does not diminish naturally.

$$Q(\boldsymbol{S},\boldsymbol{A}) \leftarrow Q(\boldsymbol{S},\boldsymbol{A}) + \alpha_T[R_T + \gamma Q(\boldsymbol{S}',\boldsymbol{A}') - Q(\boldsymbol{S},\boldsymbol{A})] \quad [\ 19\ ]$$

Therefore, α_T is replaced by a constant α, which changes the update rule of Equation [19] to that of Equation [20]. The result of this is that the true average $M_T(\boldsymbol{S},\boldsymbol{A})$ is not approximated across the state-action space. This does not compromise the RL agent's training, since changes in the training process generating the value estimates are still tracked (Poole and Mackworth, 2010). Further, any scaled instance of the numerical approximation to $Q_\pi^*(\boldsymbol{S},\boldsymbol{A})$ is sufficient to solve an RL control problem provided that suitable training data is used.

$$Q(\boldsymbol{S},\boldsymbol{A}) \leftarrow Q(\boldsymbol{S},\boldsymbol{A}) + \alpha[R_T + \gamma Q(\boldsymbol{S}',\boldsymbol{A}') - Q(\boldsymbol{S},\boldsymbol{A})] \quad [\ 20\ ]$$

The update rule of SARSA is an instance of stochastic semi-gradient descent of the Mean Squared Error (MSE) objective function written in terms of the difference between t and $Q(\boldsymbol{S},\boldsymbol{A})$ (Poole and Mackworth, 2010; Sutton and Barto, 2018; Theodoridis, 2020).

Equation [20] is written under the assumption that each coordinate of the discretized state-action space has a discrete, independent entry $Q(\boldsymbol{S},\boldsymbol{A})$. For a tabular method, the values of $Q(\boldsymbol{S},\boldsymbol{A})\ \forall\ \boldsymbol{S} \in \mathcal{S}, \boldsymbol{A} \in \mathcal{A}$ comprise the parameter vector $\boldsymbol{w}_q$ of the approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$. The scalar output

of the approximation $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ may be expressed in linear form as $\boldsymbol{w}_q^T \boldsymbol{\phi}_q(\boldsymbol{S}, \boldsymbol{A})$, where $\boldsymbol{\phi}_q(\boldsymbol{S}, \boldsymbol{A})$ is a vector that only contains a numerical value of one at the current state-action coordinate, with all the other entries equal to zero. The symbol T in the exponent of a mathematical expression refers to the transpose operation. While the linear form helps with understanding the general expression of $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$, only the table entries $Q(\boldsymbol{S}, \boldsymbol{A}) \; \forall \; \boldsymbol{S} \in \mathcal{S}, \boldsymbol{A} \in \mathcal{A}$ are of interest when applying tabular SARSA. Conceptually, the table entries correspond to the entries of $\boldsymbol{w}_q$.

Table 1 shows the steps used to implement SARSA. At each state $\boldsymbol{S}$ encountered, a SARSA agent has to consider the output of the approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ for each available $\boldsymbol{A} \in \mathcal{A}$ and determine the action selection that is associated with the maximum approximated action-value at the present time step T. This is known as the greedy action. An agent following the ε-greedy exploration strategy selects a random action with probability ε, otherwise it selects the greedy action (Sutton and Barto, 2018).

Table 1: Pseudocode for the SARSA algorithm used to update $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ with fixed episode lengths (Sutton and Barto, 2018)

1. $Initialize\ hyperparameters\ \{\alpha, \varepsilon, \gamma\}$
2. $Initialize\ Tabular\ Action - Value\ Function\ \tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A}) = 0 \; \forall \; \boldsymbol{S} \in \mathcal{S}, \boldsymbol{A} \in \mathcal{A}$
3. $for\ Episode\ Counter = 1\ to\ Number\ of\ Episodes$
    a. $Initialize\ Observed\ State\ \boldsymbol{S}$
    b. $Select\ Action\ \boldsymbol{A}\ Using\ \pi\ \left(Exploration\ Strategy\ Applied\ to\ \tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})\right)$
    c. $for\ stepCntr = 1\ to\ Number\ of\ Steps$
        i. $Apply\ \boldsymbol{A}\ to\ RL\ Environment; Obtain\ Reward\ R\ and\ \boldsymbol{S}'$
        ii. $Select\ \boldsymbol{A}'\ Using\ \pi$
        iii. $Update\ \tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})\ using\ Equation$ [20]
        iv. $\boldsymbol{S} \leftarrow \boldsymbol{S}'; \boldsymbol{A} \leftarrow \boldsymbol{A}'$
    d. end
4. end

In Equation [20], the term in brackets is known as the temporal difference (TD). Updating the estimate at $(\boldsymbol{S}, \boldsymbol{A})$ based on information at $(\boldsymbol{S}', \boldsymbol{A}')$ is an application of bootstrapping (James et al., 2013; Sutton and Barto, 2018). Since $\boldsymbol{S}'$, and therefore $\boldsymbol{A}'$, cannot be known at time step T before $T + 1$ has been experienced, the agent must always trail behind true process time when trained online.

The one-step return may be generalised to n-step returns where G is written out further before adding $\gamma^n Q(\boldsymbol{S}_{T+n}, \boldsymbol{A}_{T+n})$ to the Bellman target. As $n \to \infty$, Monte Carlo control methods are achieved. In Monte Carlo methods, the full return is therefore used as an unbiased Bellman target, and many episodes are utilised to ensure that the estimated value of G is accurate from the perspective of resampling statistics (Simon, 1997; Sutton and Barto, 2018).

The use of one-step returns and a tabular RL representation have the advantages of providing control approaches that are easily interpretable, having a lower operational data requirement than methods using n-step backups with large n, and avoiding unnecessary complexity for a feasibility study. The sampling period ΔT determines both the time interval between updating agent action selections and the bootstrapping interval when using one-step returns (Sutton and Barto, 2018).

2.8.2 Q-learning and Off-Policy Convergence

A SARSA agent must interact directly with the RL environment to adjust its estimates of the optimal parameters in the RL representation. The initialisation of the policy π will result in random action selections at each time step initially. As training progresses, the action selections will become more appropriate to solving the problem as experience is gained by the agent. Watkins (1989) developed Q-learning, the counterpart of SARSA aimed at developing an initial guess of the optimal RL representation parameter values based on historical data generated by a behavioural policy π_b. Exactly the same theoretical framework for value-based RL control given in Section 2.8.1 is applicable to this algorithm, and the corresponding update rule is shown in Equation [21].

$$Q(\boldsymbol{S}, \boldsymbol{A}) \leftarrow Q(\boldsymbol{S}, \boldsymbol{A}) + \alpha \left(R_T + \gamma \max_{\boldsymbol{a} \in \mathcal{A}} Q(\boldsymbol{S}', \boldsymbol{a}) - Q(\boldsymbol{S}, \boldsymbol{A}) \right) \qquad [\ 21\ ]$$

Note that the only difference between Equation [20] and Equation [21] is that the Bellman target is expressed in terms of the maximum action-value at $T + 1$ for Q-learning (the action-value corresponding to a greedy policy π at the next state $\boldsymbol{S}'$). This makes sense intuitively since, at each T, driving the policy π towards making action selections made by the behavioural policy π_b which provides a feasible solution is expected to provide a good initial estimate of behaviour that will solve the control problem.

The use of function approximation (Section 2.9.6), as opposed to tabular methods, is known to render Q-learning's convergence properties unreliable without certain algorithmic adjustments (Precup et al., 2001; Sutton and Barto, 2018). This unreliability of function approximation for Q-learning is illustrated in the counterexample of (Baird, 1995). Mnih et al. (2015) state that the unreliability persists

in the case of nonlinear function approximation for $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ as a result of data observation sequence correlations, the sensitivity of the policy π and consequently operational data distribution to $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$, as well as the correlations existing between the entries of the RL representation and the greedy Bellman target.

Guidelines for preventing unintended divergence while executing value-based prediction owing to RL representation design are summarised by Sutton and Barto (2018) and are assumed to be applicable to value-based control as a first approximation. Unstable training may, according to these guidelines, be avoided if only two of the following three aspects are present in the design: function approximation, bootstrapping, and off-policy training.

2.8.3 Graphical Overview of the SARSA Agent

The policy π maps the inputs provided to the agent at the current time step T to the action deemed appropriate by the agent based on the knowledge it has gained thus far (Sutton and Barto, 2018). For the SARSA agent, the relationship between the descriptive terms of the algorithms is given in Figure 2. Recall that for SARSA, the RL representation is a table (or, more generally, a discrete hypervolume) with scalar weights that indicate how beneficial each available action is at the current state ("current state" indicated by the horizontal box in Figure 2, where each row of the table is a possible state). The larger the weight, the more beneficial the action associated with that column is considered.

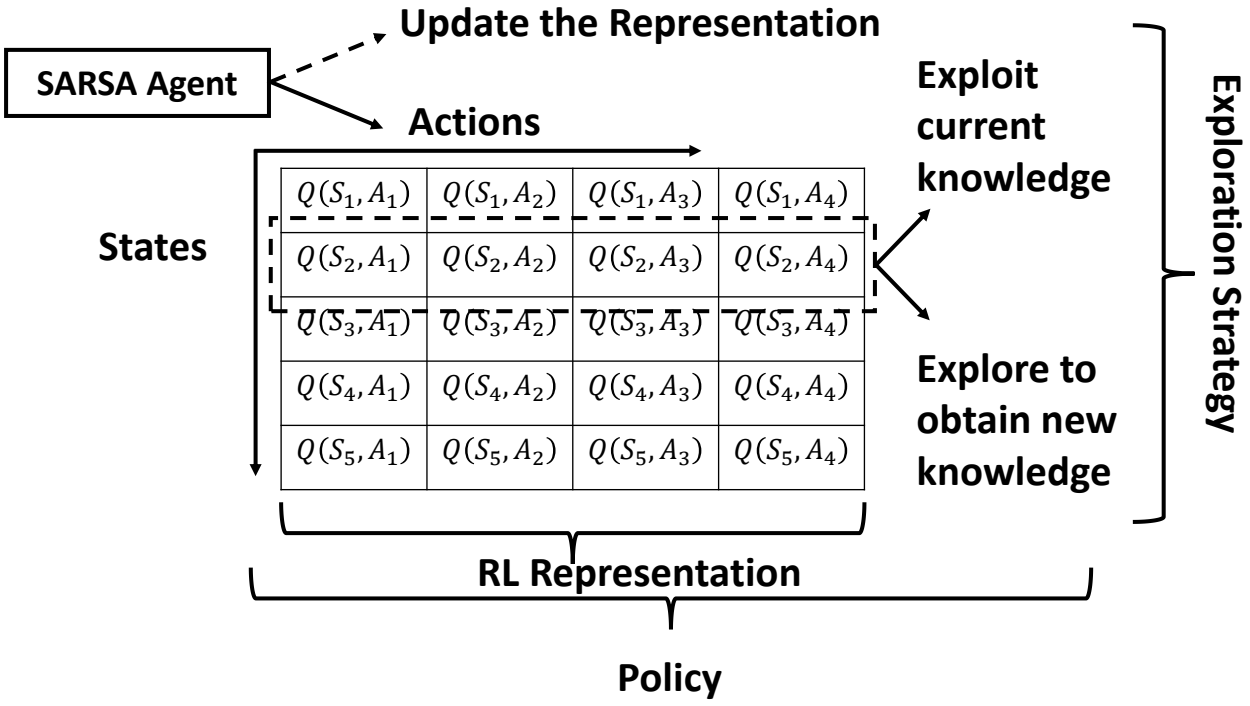


Figure 2: Relationship between SARSA agent, RL representation, policy, and exploration strategy

The table in Figure 2 therefore provides a "map" that contains the cumulative knowledge of the RL agent. To gain new knowledge, the agent must find a balance between taking the actions deemed most beneficial and exploring the application of different actions. This is done in a probabilistic fashion and is the

exploration strategy. Without exploration during training, the process becomes trivial as the agent will always select the action corresponding to the largest parameter value in the relevant row of the table (assuming that non-zero rewards were already received by the agent). The table coupled with the exploration strategy constitutes the policy π.

2.9 Policy Gradient and Actor-Critic RL Control

This section is a concise statement of theory described by Sutton and Barto (2018).

2.9.1 Problem Context

Policy Gradient theory forms the basis of any Actor-Critic approach. Recall from Section 2.3 that, in Policy Gradient and Actor-Critic RL control methods, the goal is still to maximize the expectation of the discounted return. The parameters of a model-free RL method reside in the $n(\mathcal{S}) \times n(\mathcal{A})$ – dimensional space, where $n(\mathcal{S})$ and $n(\mathcal{A})$ are the cardinalities of the state space $\mathcal{S}$ and the action space $\mathcal{A}$, respectively. As the discretization of these spaces become finer in a tabular method, the parameter space becomes greater. In Policy Gradient and Actor-Critic methods, the dimensionality of the parameter vector(s) is fixed despite the fact that the cardinality of the state-action space may become infinite. In these methods, the entries of a parameter vector defined in the $\mathcal{p}$-dimensional parameter space, $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{p}}$, are updated which enables the agent to select actions without directly consulting an approximation to $Q_\pi(\boldsymbol{S}, \boldsymbol{A})$ or $V_\pi(\boldsymbol{S})$ across the entire state space $\mathcal{S}$.

The vector $\boldsymbol{\theta}$ parameterizes a probability distribution describing the probability of selecting an action when the agent observes the current state $\boldsymbol{S}$, as shown in Equation [22]. This is the parameterized policy $\pi_{\boldsymbol{\theta}}$. The agent selects an action using Equation [23], meaning that the action at time step T is sampled from the probability distribution describing π. The conditional probabilities of Equation [22] and Equation [23] express that the probability of selecting an action at time step T is dependent on the state $\boldsymbol{S}$ and the instance of $\boldsymbol{\theta}$ available at T. Recall that the value functions are unique for a given policy π. The optimal value functions are unique for an RL problem, while the optimal policy π^* is not necessarily (even though the discounted returns across the state-action space will be equal for the different optimal policies). The optimal policy may be understood conceptually in process control problems, Section 2.11.

$$\pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta}) = Pr[\boldsymbol{A}_T = \boldsymbol{A}|\boldsymbol{S}_T = \boldsymbol{S}, \boldsymbol{\theta}_T = \boldsymbol{\theta}] \qquad [\ 22\ ]$$

$$\boldsymbol{A}_T \sim \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta}) \qquad [\ 23\ ]$$

2.9.2 Stochastic Gradient Ascent and the Policy Gradient Theorem

To obtain suitable entries for the elements of $\boldsymbol{\theta}$ given a selected parametric function which is used to estimate an optimal policy π^*, the stochastic gradient ascent algorithm shown in Equation [24] is applied as an update rule. The function $J(\boldsymbol{\theta})$ is a scalar performance objective that measures the suitability of the current estimate of $\boldsymbol{\theta}$ and needs to be maximised. The gradient of $J(\boldsymbol{\theta})$ in the parameter space is $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

When applying the episodic approach to RL agent training, $J(\boldsymbol{\theta})$ is defined as the state-value of the non-random starting state of each episode, $\boldsymbol{S}_0$, for parameterized policy $\pi_{\boldsymbol{\theta}}$, Equation [25].

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad [\ 24\ ]$$

$$J(\boldsymbol{\theta}) \doteq V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S}_0) \qquad [\ 25\ ]$$

Note that the term policy π is still used generically in this thesis rather than specifying $\pi_{\boldsymbol{\theta}}$ or $\pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta})$ for Policy Gradient and Actor-Critic control policies since the term describes the map from state to action learned by an RL agent by adjusting the parameters of its RL representation(s), $\pi(\boldsymbol{A}|\boldsymbol{S})$. By maximising this performance objective $J(\boldsymbol{\theta})$ through the adjustment of the entries in the vector $\boldsymbol{\theta}$ during interaction of the agent with the RL environment, we are ensuring that an optimal policy is approximated during training. While the use of the starting state $\boldsymbol{S}_0$ might seem arbitrary, it is chosen because it is often the only state one has control over (for example an initial steady state condition of a plant).

The derivative of π with respect to $\boldsymbol{\theta}$ must be finite and real. An often-used parameterization is the soft-max/Boltzmann distribution of Equation [26], where the normalization in the denominator ensures that a true probability distribution is achieved and that each individual probability lies between zero and one. The function $h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta})$, as defined in Equation [27], is a preference function that is expressed similarly to the linear form $\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S},\boldsymbol{A})$ in value-based methods (Section 2.8.1). An increased preference results in an increased probability of selecting $\boldsymbol{A}$ again when the same state is encountered later during the training process. By sampling an action from Equation [26] when required, exploration is ensured.

$$\pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta}) = \pi = \frac{\exp\left(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta})\right)}{\sum_{\boldsymbol{b}} \exp\left(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta})\right)} \in [0,1] \; and \; \boldsymbol{b} \in \mathcal{A} \qquad [\ 26\ ]$$

$$h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{X}(\boldsymbol{S},\boldsymbol{A}) \qquad [\ 27\ ]$$

In Equation [27], $\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})$ refers to a vector in which each entry contains a function evaluated at the coordinate $(\boldsymbol{S},\boldsymbol{A})$ within the state-action space. As $h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta})$ becomes sufficiently large, the policy becomes deterministic. This means that an output of $\pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta}) \rightarrow 1$ is obtained if $\boldsymbol{A}$ is the action which the agent becomes very confident in applying to the RL environment given the currently observed state $\boldsymbol{S}$.

At each T, the update rule used for the actor must ensure that the adjustment of the elements in $\boldsymbol{\theta}$ causes improved decision making by the RL agent. The appropriate direction for adjusting $\boldsymbol{\theta}$ in the $\mathcal{p}$ – dimensional parameter space is $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. The theoretical expression for the policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is shown in Equation [28] with K_{PG} equal to the average length of an episode and the fraction of time spent in each $\boldsymbol{S} \in \mathcal{S}$ is given by $\mu(\boldsymbol{S})$ (this is unknown). The derivation of Equation [28] is shown by Sutton and Barto (2018). Insufficient information of the training process is available to solve Equation [28] analytically at each time step since this requires information on the frequencies with which the states will be visited during training. A sample gradient for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ must therefore be derived and subsequently used to execute Equation [24] at each time step T.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = K_{PG} \sum_{\boldsymbol{S} \in \mathcal{S}} \mu(\boldsymbol{S}) \sum_{\boldsymbol{A} \in \mathcal{A}} Q_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S}, \boldsymbol{A}) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta}) \quad [\ 28\ ]$$

The expected value of the sample gradient must be proportional to the actual policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ described by Equation [28]. Equation [28] is sufficient to derive such a sample gradient, since the constant K_{PG} may be absorbed into the step size hyperparameter $\alpha_{\boldsymbol{\theta}}$ of Equation [24] when writing the update rule for π. The summation across $\boldsymbol{A} \in \mathcal{A}$ in Equation [28] may be viewed as a discrete approximation to the action-value of an event associated with adjusting $\boldsymbol{\theta}$ using the gradient of π. The on-policy distribution $\mu(\boldsymbol{S})$ describes the probabilities with which states will be experienced if training occurs online. We can then write $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ as Equation [29] after sampling $\boldsymbol{S}_T$, the state at time step T. The operand of Equation [29] may also be written as an expectation with respect to the instance of the policy π. Firstly, $1 = \frac{\pi(\boldsymbol{A}|\boldsymbol{S}_T, \boldsymbol{\theta})}{\pi(\boldsymbol{A}|\boldsymbol{S}_T, \boldsymbol{\theta})}$ is multiplied in the operand, followed by sampling $\boldsymbol{A}_T$ which is, for on-policy training, experienced according to $\pi(\boldsymbol{A}|\boldsymbol{S}_T, \boldsymbol{\theta})$. The result is Equation [30], which may be simplified to Equation [31] by using Equations [2] and [6] to write Equation [30] in terms of the discounted return sampled at time step T, denoted by G_T (Equation [2]).

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi}[\sum_{\boldsymbol{A} \in \mathcal{A}} Q_{\pi}(\boldsymbol{S}_T = \boldsymbol{S}, \boldsymbol{A}) \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta})] \quad [\ 29\ ]$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi}\left[\sum_{\boldsymbol{A} \in \mathcal{A}} \left(\frac{\pi(\boldsymbol{A}|\boldsymbol{S}_T, \boldsymbol{\theta}) Q_{\pi}(\boldsymbol{S}_T = \boldsymbol{S}, \boldsymbol{A})}{\pi(\boldsymbol{A}|\boldsymbol{S}_T, \boldsymbol{\theta})}\right) \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta})\right]$$

$$= \mathbb{E}_{\pi}\left[Q_{\pi}(\boldsymbol{S}_T = \boldsymbol{S}, \boldsymbol{A}_T = \boldsymbol{A}) \frac{\nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}\right] \quad [\ 30\ ]$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi}\left[G_T \frac{\nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}\right] \quad [\ 31\ ]$$

2.9.3 The Eligibility Vector

The vector $\frac{\nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}$ of Equation [31], which is written in terms of the sampled action $\boldsymbol{A}_T = \boldsymbol{A}$ and state $\boldsymbol{S}_T = \boldsymbol{S}$, is the direction in $\mathbb{R}^{\mathcal{P}}$ that increases the probability of selecting the same action $\boldsymbol{A}$ during future visits to the same observed state $\boldsymbol{S}$. The inverse proportionality of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ to $\pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})$ prevents Equation [31] from being biased with respect to more frequently experienced actions. Appendix B describes how an analytical expression for $\frac{\nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}$ is obtained for the parametrized policy form of Equation [26].

2.9.4 The Baseline Function

In Policy Gradient methods a function of state, $b(\boldsymbol{S})$, may be subtracted from $Q_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ in Equation [28]. This function is often referred to as a baseline. The form of $b(\boldsymbol{S})$ must be only a function of $\boldsymbol{S}$ to ensure that the expected value of training is not dependent on the choice of $b(\boldsymbol{S})$. The simplification of the resulting expression is related in Equations [32] through [34]. Since $\nabla_{\boldsymbol{\theta}}(\sum_{\boldsymbol{A} \in \mathcal{A}} \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta})) = \nabla_{\boldsymbol{\theta}}(1) = 0$, Equation [32] is equivalent to Equation [28]. This, in turn,

implies that the expectation with respect to the current policy π in Equation [31] is not influenced by the subtraction of a baseline.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = K_{PG} \sum_{\boldsymbol{S} \in \mathcal{S}} \mu(\boldsymbol{S}) \sum_{\boldsymbol{A} \in \mathcal{A}} \big(Q_{\pi}(\boldsymbol{S}, \boldsymbol{A}) - b(\boldsymbol{S})\big) \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta}) \quad [\ 32\ ]$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = (K_{PG} \sum_{\boldsymbol{S} \in \mathcal{S}} \mu(\boldsymbol{S})) \big[\sum_{\boldsymbol{A} \in \mathcal{A}} \big(Q_{\pi}(\boldsymbol{S}, \boldsymbol{A}) \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta})\big) - b(\boldsymbol{S}) \nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{A} \in \mathcal{A}} \big(\pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta})\big)\big] \quad [\ 33\ ]$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = (K_{PG} \sum_{\boldsymbol{S} \in \mathcal{S}} \mu(\boldsymbol{S})) \big[\sum_{\boldsymbol{A} \in \mathcal{A}} \big(Q_{\pi}(\boldsymbol{S}, \boldsymbol{A}) \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}|\boldsymbol{S}, \boldsymbol{\theta})\big)\big] \quad [\ 34\ ]$$

Theodoridis (2020) highlights that the variance in the updates used in a stochastic approximation algorithm, and therefore also the speed of parameter convergence, are disadvantaged by the use of an observation set to approximate targets and parameter space gradients. Sutton and Barto (2018) indicate that subtracting a baseline aids with reducing this effect.

Qualitatively, the role $b(\boldsymbol{S})$ would play in an update rule may be understood. The target remains a numerical point estimate of discounted return (Equation [2]), and the direction of parameter adjustment remains $\frac{\nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T, \boldsymbol{\theta})}$. The function $b(\boldsymbol{S})$ serves as a way of providing the agent with information regarding which updates are more worthwhile in terms of optimising the expected value of the discounted return. If the action selection $\boldsymbol{A}_T = \boldsymbol{A}$ at T results in low discounted return yield in comparison to other known selections, an adjustment to $\boldsymbol{\theta}$ that increases the probability of repeating the selection $\boldsymbol{A}_T = \boldsymbol{A}$ at $\boldsymbol{S}_T = \boldsymbol{S}$ is likely to be less beneficial than a different selection. In this sense, $b(\boldsymbol{S})$ may be used to increase algorithm efficiency by providing "critique" to the agent.

It is important to understand that the equivalence of Equations [28] and [32] implies that the expected value of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, and therefore the expected value of the sample gradient, are not affected by the subtraction of $b(\boldsymbol{S})$. The *trajectory* that the entries comprising $\boldsymbol{\theta}$ follow during training is. A specific choice of $b(\boldsymbol{S})$ is known as a critic, as clarified in Section 2.9.5. The "critique" provided by a critic is known to enable significant reduction in the variance present in the updates to the components of $\boldsymbol{\theta}$ when compared to an agent that only differs by not incorporating a baseline function $b(\boldsymbol{S})$.

2.9.5 The One-Step Actor-Critic Algorithm

The state-value function $V_{\pi}(\boldsymbol{S})$ is a natural choice for a baseline $b(\boldsymbol{S})$. As is the case with SARSA, a one-step return (only information at T and $T+1$ used in the update rule(s)) may be used during algorithmic updates instead of a numerical approximation to the full discounted return. To learn $V_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S})$, the expression $\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{S})$ for the approximated state-value $\left(\tilde{V}_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S})\right)$ may be used (also see Section 2.8.1), and the state-value form of the Bellman target may be used for this RL prediction problem. The vector $\boldsymbol{w}$ contains the parameters relevant to the RL representation for the value function. The vector $\boldsymbol{\phi}(\boldsymbol{S})$ contains basis functions used in function approximation, Section 2.9.6. The RL prediction counterpart to SARSA is an algorithm known as $TD(0)$, and the resulting update rule is used to update the parameter vector $\boldsymbol{w}$ of $\tilde{V}_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S})$

is given in Equation [35]. The derivation of Section 2.8.1 may be used to understand Equation [35]. The gradient in the parameter space of $\boldsymbol{w}$ is $\nabla_{\boldsymbol{w}}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})\right) = \boldsymbol{\phi}(\boldsymbol{S})$. The only differences between $TD(0)$ and tabular SARSA (Section 2.8.1) are that function approximation (Section 2.9.6) is used, and that the action space is not accounted for in the RL representation.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\boldsymbol{w}}\left(R_T + \gamma \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S}') - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})\right)\boldsymbol{\phi}(\boldsymbol{S}) \qquad [\ 35\ ]$$

The Bellman target written in terms of state value may be used to replace G_T in Equation [31] when calculating the sample gradient for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Substituting the resulting sample gradient into Equation [24] yields Equation [36]. In Equation [36], the discount factor γ must be equal to unity since it does not incorporate the effect of $\gamma < 1$ on the frequencies with which certain states are visited by the RL agent (Sutton and Barto, 2018).

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}}(\gamma)\left(R_T + \gamma \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S}') - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})\right)\frac{\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}\ if\ \gamma = 1 \qquad [\ 36\ ]$$

Recall from Section 2.9.2 that K_{PG} is the average length of an episode. From the proof of the policy gradient theorem presented by Sutton and Barto (2018), it is known that $K_{PG} = \sum_{\boldsymbol{S}' \in \mathcal{S}} \eta(\boldsymbol{S}')$. The symbol $\eta(\boldsymbol{S}')$ denotes the average number of time steps spent in $\boldsymbol{S}'$ per episode. Further, this can be expanded as shown in Equation [37], where $Pr(\boldsymbol{S}' = \boldsymbol{S}_0)$ is the probability that an episode starts in state $\boldsymbol{S}'$. This may be simplified by realising that $Pr(\boldsymbol{S}' = \boldsymbol{S}_0) = 0$ as we require a non-random starting state – each episode starts in $\boldsymbol{S}_0$ and therefore the probability of starting in $\boldsymbol{S}'$ is zero.

$$K_{PG} = \sum_{\boldsymbol{S}' \in \mathcal{S}} \eta(\boldsymbol{S}') = \sum_{\boldsymbol{S}' \in \mathcal{S}}[Pr(\boldsymbol{S}' = \boldsymbol{S}_0) + \sum_{\boldsymbol{S} \in \mathcal{S}} \eta(\boldsymbol{S}) \sum_{\boldsymbol{A} \in \mathcal{A}} \pi(\boldsymbol{A}|\boldsymbol{S})\, Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})]\ if\ \gamma = 1 \qquad [\ 37\ ]$$

To generalize Equation [37] to the case where $0 < \gamma < 1$, it needs to be realised that the frequencies with which instances of $\boldsymbol{S}'$ are encountered during training depend on their importance with regard to the accumulation of reward, and therefore the discount factor γ. From the definition of the discounted return, Equation [2], it is known that each possible instance of $\boldsymbol{S}'$ that follows the starting state is associated with the reward received multiplied with γ^{T-1}. Later steps in an episode contribute less to the approximation of $J(\boldsymbol{\theta}^*) = V^*_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S}_0)$, where $\boldsymbol{\theta}^*$ is the optimal actor parameter vector and $V^*_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{S}_0)$ is the optimal state value evaluated at the starting state of each episode $\boldsymbol{S}_0$, and this relative importance is expressed through discounting. The result is shown in Equation [38] for $0 < \gamma < 1$ and γ^{T-1} therefore needs to be incorporated in the sample gradient, and consequently in the update rule of the actor, Equation [36]. Generalizing Equation [36] in this way leads to Equation [39] as the update rule for the actor.

$$K_{PG} = \sum_{\boldsymbol{S}' \in \mathcal{S}} \eta(\boldsymbol{S}') = \gamma^{T-1} \sum_{\boldsymbol{S}' \in \mathcal{S}}[0 + \sum_{\boldsymbol{S} \in \mathcal{S}} \eta(\boldsymbol{S}) \sum_{\boldsymbol{A} \in \mathcal{A}} \pi(\boldsymbol{A}|\boldsymbol{S})\, Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})] \qquad [\ 38\ ]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}}\gamma^{T-1}\left(R_T + \gamma \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S}') - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})\right)\frac{\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}\ if\ 0 < \gamma < 1 \qquad [\ 39\ ]$$

It is important to realise that any RL algorithm relies on the properties of value functions. This is because the definition of $J(\boldsymbol{\theta})$ only ensures that increased cumulative reward is pursued by the agent as it maximises $J(\boldsymbol{\theta})$ because of the existence and uniqueness of the state-value function for the optimal policy, $V_{\pi_{\boldsymbol{\theta}}}^{*}(\boldsymbol{S})$. More specifically, $V_{\pi_{\boldsymbol{\theta}}}^{*}(\boldsymbol{S}_0)$ must be fixed by the uniqueness of $V_{\pi}^{*}(\boldsymbol{S})$ for the performance objective $J(\boldsymbol{\theta})$ to work. When ascending the sample gradient does not change the approximation to the policy π, then the optimal policy π^* has been approximated.

It should be noted that the policy parameter vector $\boldsymbol{\theta}$ still exists without the value function. The Bellman target $R_T + \gamma \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{S}')$ influences asymptotic performance by introducing bias through bootstrapping. This bias is known to often benefit learning through update variance reduction and to promote learning acceleration as was the case with SARSA. The bootstrapping property is required for a choice of $b(\boldsymbol{S})$ to be deemed a critic. Table 2 provides pseudocode for the One-Step Actor-Critic algorithm that includes the appropriate analytical eligibility vector form from Appendix B.

Table 2: Pseudocode for the One-Step Actor-Critic algorithm used to update $\tilde{V}_\pi(\boldsymbol{S})$ and $\pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta})$, the numerical approximations to $V_\pi(\boldsymbol{S})$ and parametric policy π with fixed episode lengths (on-policy) (Sutton and Barto, 2018)

1. *Initialize hyperparameters and RL representation parameters* $\{\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{w}}, \gamma, \boldsymbol{\theta}, \boldsymbol{w}\}$
2. *Input*: $\pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta}) = \frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}$ *with* $h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})$
3. *Input*: $\tilde{V}_\pi = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})$
4. *for Episode Counter* $= 1$ *to Number of Episodes*
    a. *Initialize* $\boldsymbol{S}$ *in accordance with Equation* [25]
    b. $I \leftarrow 1$
    c. *for stepCntr* $= 1$ *to Number of Steps*
        i. *Select* $\boldsymbol{A}_T \sim \pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta})$
        ii. *Apply* $\boldsymbol{A}_T = \boldsymbol{A}$ *to RL Environment; Obtain Reward* R *and* $\boldsymbol{S}'$
        iii. $\delta \leftarrow R + \gamma\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S}') - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})$
        iv. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\boldsymbol{w}}\delta\nabla_{\boldsymbol{w}}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S})\right)$
        v. $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} I\delta\begin{bmatrix} X_1(\boldsymbol{S},\boldsymbol{A}) - \frac{\sum_{\boldsymbol{b}}\left[\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)\mathcal{X}_1(\boldsymbol{S},\boldsymbol{b})\right]}{\sum_b \exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)} \\ \vdots \\ X_{\mathcal{p}}(\boldsymbol{S},\boldsymbol{A}) - \frac{\sum_{\boldsymbol{b}}\left[\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)\mathcal{X}_{\mathcal{p}}(\boldsymbol{S},\boldsymbol{b})\right]}{\sum_b \exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)} \end{bmatrix}$$
        vi. $I \leftarrow \gamma I$
        vii. $\boldsymbol{S} \leftarrow \boldsymbol{S}'$
    d. end
5. end

2.9.6 Function Approximation and Generalization

Fasshauer (2007) explains the use of radial basis function (RBF) interpolation as a method of approximating functions across a real domain. A least squares solution is determined to reconstruct a test

function after interpolation by multiplication of an interpolation matrix with a vector of parameters. The interpolation matrix contains the values of the basis functions evaluated at the data points in the columns. After matrix multiplication is performed, the interpolated function has been constructed.

Both the approximations to the state-value function $\tilde{V}_\pi(\boldsymbol{S})$ and the action-value function $\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A})$ may be written in the linear form described in Sections 2.8.1 and 2.9.5. This is shown in Equations [40] and [41]. By letting each entry in the column vector $\boldsymbol{\phi}(\boldsymbol{S})$ or $\boldsymbol{\phi}(\boldsymbol{S},\boldsymbol{A})$ be the output of a function that maps from $\boldsymbol{S}$ or $(\boldsymbol{S},\boldsymbol{A})$ to $\mathbb{R}^1$, one allows an interdependence of the different elements of the parameter vector $\boldsymbol{w}$ which needs to be determined for each type of problem (prediction or control problem, Section 2.7). The vectors $\boldsymbol{\phi}(\boldsymbol{S})$ and $\boldsymbol{\phi}(\boldsymbol{S},\boldsymbol{A})$ are basis function vectors. This approach to generating an RL representation is known as function approximation.

$$\tilde{V}_\pi(\boldsymbol{S}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S}) \qquad [\ 40\ ]$$

$$\tilde{Q}_\pi(\boldsymbol{S},\boldsymbol{A}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{S},\boldsymbol{A}) \qquad [\ 41\ ]$$

An often used form for each element in a basis function vector is the RBF form described by Sutton and Barto (2018). By applying this form, Equation [42] may be used to express each entry $\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A})$ in the column vector $\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})$ of Equation [27] for the preference function $h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A}) = \sum_i \theta_i\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A})$ for Actor-Critic control. The column vector containing $\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A})$ for $i \in 1,2,\ldots\mathcal{p}$ is therefore the basis function vector for the actor, $\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})$. Equation [42] contains a term for every state component comprising $\boldsymbol{S}$, and a term for every action component comprising $\boldsymbol{A}$. Each constant $\boldsymbol{c}$ is specific to the basis function $\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A})$, while the scale variable σ may be kept constant.

$$\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A}) \doteq \exp\left(-\left(\frac{|\boldsymbol{S}-\boldsymbol{c}_{S,i}|^2+|\boldsymbol{A}-\boldsymbol{c}_{A,i}|^2}{2\sigma^2}\right)\right) \qquad [\ 42\ ]$$

For $\tilde{V}_\pi(\boldsymbol{S})$, each element of $\boldsymbol{\phi}(\boldsymbol{S})$ is parameterized as in Equation [42], the only difference being that actions are not appended. Each element θ_i in $\boldsymbol{\theta}$ is associated with an $\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A})$, while each element w_i in $\boldsymbol{w}$ is associated with a $\phi_i(\boldsymbol{S})$ when implementing the algorithm given in Table 2.

Figure 3 and Figure 4 show an example of an RBF in $\mathbb{R}^2$. Each RBF has a perceptive field and the larger the overlap between the fields of the RBFs used, the stronger the interdependence between those parameters. This leads to generalization across the state space or state-action space and enhanced coverage during training. Generalization therefore describes the behaviour of an RL representation where different elements of the parameter vector are updated differently depending on their positions within the state space (RL prediction) or the state-action space (RL control).

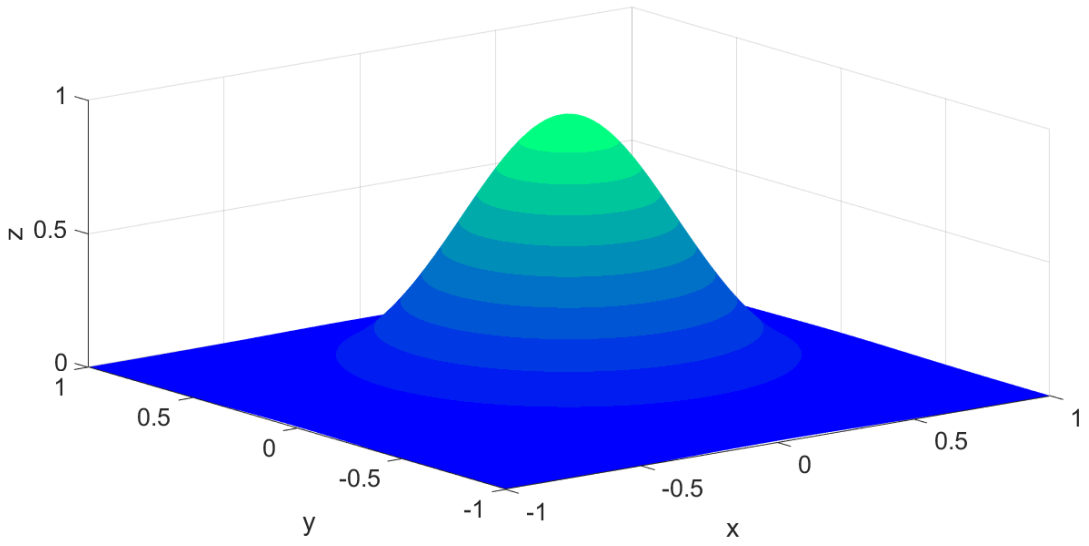


Figure 3: Example of an RBF in $\mathbb{R}^2$ with $\sigma^2 = 0.1$

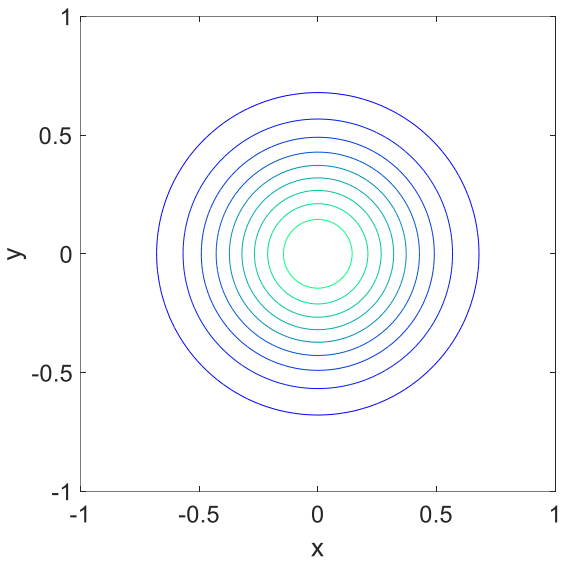


Figure 4: Top view of the RBF shown in Figure 3

2.9.7 Dimensionality of the Action Space

The actions available to a One-Step Actor-Critic agent may be discrete or continuous. In the discrete case, for each instance of $\boldsymbol{S}$, the agent is provided with a probability mass function (pmf) given by $\pi(\boldsymbol{A}|\boldsymbol{S},\boldsymbol{\theta})$. The actions available to the RL agent are defined by the designer as discrete selections and comprise the discretized action space $\mathcal{A}$. Each action selection may have an identifying number and the respective numbers should be scaled consistently with the components of the state $\boldsymbol{S}$ to prevent unintended distortion of the RBFs' shapes when applying the update rules given by Equations [35] and [36]. Each number that the RL agent may select as an action must map to a corresponding input applied to the RL environment.

Generalization using the RBFs placed in the state-action space allows enhanced coverage over tabular methods to be achieved while at the same time benefitting from the policy gradient theorem and the direct learning of an optimal policy π^*. While discretized actions would not achieve the best possible performance during training for problems that have continuous action spaces, it is natural to consider the discretized case when targeting efficiency in the use of operational hours and considering only a small increase in RL representation complexity in comparison to tabular methods.

It is worth noting that an Actor-Critic algorithm may be adapted to naturally accommodate large action spaces and even continuous action spaces, by applying function approximation to learn the parameters of a probability density function (pdf). The parameter vector $\boldsymbol{\theta}$ is then again used with a basis function vector that contain basis functions placed in the state space. The parameters describing the pdf are expressed in terms of $\boldsymbol{\theta}$ (Sutton and Barto, 2018). In Chapter 3 this will be observed to be a significant driving force in the development of state-of-the-art RL algorithms that utilise sophisticated function approximation such as deep neural networks.

2.9.8 Graphical Overview of the One-Step Actor-Critic Agent

In contrast to SARSA, the One-Step Actor-Critic agent's RL representation consists of a probability distribution that is created for the current state. By sampling from the probability distribution, exploration is automatically ensured. This is because sampling from a distribution inherently has an extent of randomness associated with it. The probability mass function shown in Figure 5 is described by the parameter vector $\boldsymbol{\theta}$. The parameter vector $\boldsymbol{w}$ associated with $\tilde{V}_{\pi}(\boldsymbol{S})$ captures the cumulative knowledge of the agent regarding how beneficial the states of the RL environment are, while $\boldsymbol{\theta}$ captures the policy of the agent based on cumulative knowledge.

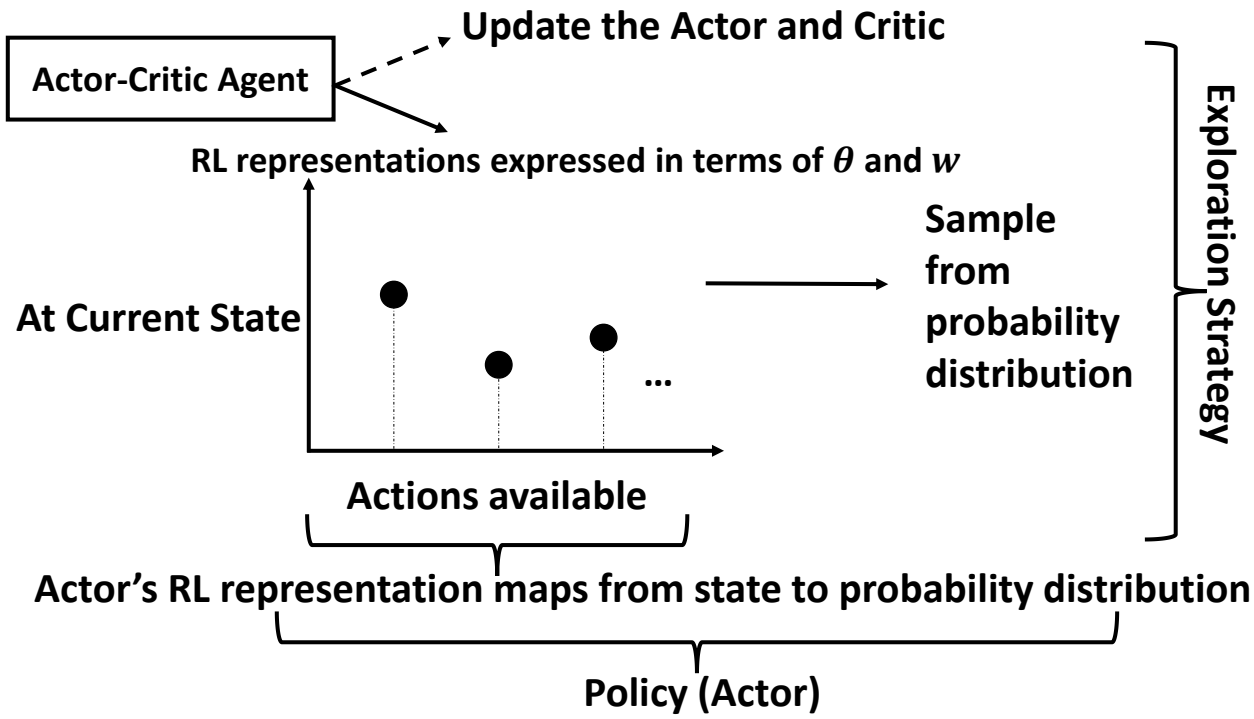


Figure 5: Relationship between Actor-Critic Agent, actor's RL representation, policy, and exploration strategy

2.10 Practicalities of Approximating a Stochastic Optimal Policy

Any on-policy RL agent starts training by selecting actions randomly. Recall that exploration ensures that the agent experiences new state-action coordinates where it does not yet have a notion of what the potential is for obtaining increased rewards. For value-based control (SARSA), Equation [26] may be rewritten in terms of the action-value function's table entries, as shown in Equation [43] (Tan, 1993). The temperature parameter may be used to adjust the degree of determinism in the action selections. At each $\boldsymbol{S}$, π is a pmf – a vector containing the probability masses of each action. The most likely actions are then sampled from this vector to select the next action. Using Equation [43] with a value-based method after approximating $Q_{\pi}^{*}(\boldsymbol{S}, \boldsymbol{A})$, the designer may obtain different stochastic policies by adjusting the "temperature" parameter.

$$\pi(\boldsymbol{A}|\boldsymbol{S}) = \frac{\exp\left(\frac{Q(\boldsymbol{S},\boldsymbol{A})}{temperature}\right)}{\sum_{\boldsymbol{b}} \exp\left(\frac{Q(\boldsymbol{S},\boldsymbol{b})}{temperature}\right)} \; where \; \pi \in [0,1] \; and \; \boldsymbol{b} \in \mathcal{A} \qquad [\,43\,]$$

If an ε-greedy exploration strategy is applied after the parameters of the RL representation (the table entries of SARSA) have converged, a deterministic policy is obtained. Importantly, *the designer decides* whether a deterministic (ε-greedy) or stochastic (Boltzmann, Equation [43]) policy is appropriate.

In contrast, an Actor-Critic RL agent automatically adjusts the entries of the actor's parameter vector $\boldsymbol{\theta}$ which determine the degree to which the learned policy π is deterministic. This is possible because adjusting the entries of $\boldsymbol{\theta}$ drives the preference function $h(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{\theta})$, Equation [27], to larger values in appropriate regions of the state-action space. The preference function $h(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{\theta})$ is free of the constraints

of the optimal action-value function across the state-action space, $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$, since the performance objective $J(\boldsymbol{\theta})$ only depends on the optimal state-value at the starting state of an episode, $V_\pi^*(\boldsymbol{S}_0)$.

2.11 Understanding an Optimal Policy in RL-Based Process Control

In the field of RL-based process control, a process with fixed physical properties and design will have one unique optimal policy π^* that results in the best possible control behaviour given the available instrumentation and equipment. The design of an RL-based control approach simply aims to approximate this optimal policy in the presence of limited information provided by the available instrumentation. Since the RL algorithms are based on different principles (value-based, Policy Gradient, or Actor-Critic) and different RL representations and reward functions may be designed, different approximations to the underlying optimal policy are obtained during training.

2.12 Model-Based RL Control

In a typical industrial control setting, each transition from $\boldsymbol{S}$ to $\boldsymbol{S}'$ is associated with uncertainty. This uncertainty results mainly from the limited plant information that can be gathered using the available instrumentation, and the presence of noise in plant measurements. Recall from Section 2.4 that such probabilistic information is contained in the unknown state-transition probabilities $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$.

The model $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ therefore contains the probabilities of moving to each possible instance of $\boldsymbol{S}'$ within the RL representation from the state-action coordinate encountered at T. Such a model may be used to characterise the RL problem, inform the sequential assignment of discounted returns to agent behaviours, and improve the decision making performed (Jaakkola et al., 1994; Potapov and Ali, 2003). RL approaches that develop an approximation to $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ are known as model-based methods. There are challenges specific to model-based methods, such as computational complexity and the dependence of achievable agent behaviour on the accuracy of $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$.

In contrast, model-free methods do not approximate $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$ as a statistical model for the RL environment, and decision making is therefore based solely on experience. The agent remains a statistical model for sequential decision making irrespective of whether model-based or model-free RL is studied, but is referred to as an agent to avoid ambiguity in the use of the word "model".

2.13 Accounting for the Bias-Variance Trade-Off

Sections 2.8 and 2.9 clarified that RL control may be viewed as a regression problem where semi-gradient descent and/or gradient ascent are used to minimize a cost function $J(\boldsymbol{w})$ in the case of value-based methods, or maximise a performance objective $J(\boldsymbol{\theta})$ in the case of Policy Gradient and Actor-Critic methods. An often-encountered concern in regression problems is the possibility of fitting the parameter vector to an instance of noise provided by the training data. As we increase the number of parameters to be trained, the degrees of freedom of our approximator increases at the expense of a larger risk of fitting

the parameters to noise present in the data (Bishop, 2006; James et al., 2013). In this section, $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ is used as an example.

Bishop (2006) shows that the mean-squared error expected loss function $J(\boldsymbol{w})$ can be decomposed as illustrated qualitatively in Equation [44] by taking the expectation of the squared error between the unknown underlying value function and the approximation $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ with respect to the observed data set (instead of with respect to the policy π).

$$J(\boldsymbol{w}) = (bias)^2 + variance + noise \qquad [\ 44\ ]$$

The bias in Equation [44] refers to the difference between the expected prediction provided by $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ and the true underlying value function for policy π. The variance expresses how sensitive the values of $\boldsymbol{w}$ are to the sampled data set. The series of $(\boldsymbol{S}, \boldsymbol{A}, R, \boldsymbol{S}')$ tuples are a stochastic process defined by $Pr(\boldsymbol{S}', R_T|\boldsymbol{S}, \boldsymbol{A})$ such that G_T is the sampled return from the stochastic process $\sum_{i=1}^{T} \gamma^{(i-1)} R_{i\Delta T} Pr(\boldsymbol{S}', R_T|\boldsymbol{S}, \boldsymbol{A}) Pr(\boldsymbol{A}|\boldsymbol{S}) Pr(\boldsymbol{S})$ defined in the time domain for each episode. This stochastic process is defined in terms of the full return (Equation [2]), and therefore the factor γ^{T-1} is required for MDP time step T (similar to Equation [39] of Section 2.9.5) rather than γ^1.

The probabilities comprising the stochastic processes are the true and unknown state-transition probabilities of the RL environment relevant to the state-action coordinates, $Pr(\boldsymbol{S}', R_T|\boldsymbol{S}, \boldsymbol{A})$, the action probabilities conditioned to the corresponding instances of $\boldsymbol{S}$, $Pr(\boldsymbol{A}|\boldsymbol{S})$, which would be instantiated by the different policies generated during training, and the probability of encountering the respective states during training, $Pr(\boldsymbol{S})$. The variance around the expectation of the Bellman target used to sample G_T results from process stochasticity, is unmodelled, and constitutes the noise. In a deterministic system, the noise would be zero.

There remains a trade-off between bias and variance. This means that as an RL representation is equipped with a larger parameter space with d dimensions, $\boldsymbol{w} \in \mathbb{R}^d$, it can fit better to data and thus has a larger probability of having high variance. This increases the risk of fitting the parameters to training data noise (James et al., 2013). As the size of the parameter space is decreased, there is a greater risk of the RL representation becoming biased.

The size of the parameter space for SARSA application may be selected to ensure that the parameter space is much smaller than the state-action space through discretization. It should be kept in mind that the true state-action space of a process control problem is, in reality, often continuous. In addition, many different data points are experienced during agent training, and a discretized action space may be used. The actor and critic basis function models described in this chapter have predefined shapes and positions. Overfitting may therefore reasonably be assumed not to be a concern for the theory described in this chapter, and it is much more likely in deep neural networks (Sutton and Barto, 2018). For these reasons, regularisation techniques described by Bishop (2006) and James et al. (2013) are not incorporated for the RL representations considered.

2.14 Challenges Posed by Stationary RL Environments

2.14.1 The MDP Assumption

Section 1.1 mentioned that the vast majority of RL algorithms are based on the MDP mathematical formulation. The Markov property is an assumption with regard to the definition of $\boldsymbol{S}$. The probability of each instance of state $\boldsymbol{S}'$ and reward R at time step $T+1$ is assumed to depend only on the values of $\boldsymbol{S}$ and $\boldsymbol{A}$ at T (Engel, 2005; Szepesvári, 2010; Sutton and Barto, 2018).

Any process can be transformed to an MDP by introducing additional state components in the definition of $\boldsymbol{S}$. If excessive information is included in $\boldsymbol{S}$, the RL agent will still be able to learn a suitable optimal policy, but the operational data requirement is more likely to become unrealistic. If insufficient information is incorporated in $\boldsymbol{S}$ to satisfy the assumption of a Markovian environment, the MDP is partially observable and including an extent of historical information in the components of $\boldsymbol{S}$ may improve the control behaviour learnt (Moerland et al., 2021). For an extent of historical information included in $\boldsymbol{S}$, the optimal definition of the state space $\mathcal{S}$ for a particular process lies between the extremes of including many state components and including insufficient state components. An RL environment that is not truly Markovian may still be modelled as such to varying degrees of success.

2.14.2 Reward Function Design and Assignment of Rewards

There are no fixed rules with regard to how the goals envisaged by the designer should be translated to a state-dependent reward function. The rewards may be sparse or continuous (Shipman and Coetzee, 2019). Further, associating a reward received with the correct change(s) in action may pose a considerable challenge. Adjustment of the sampling period ΔT is a straightforward initial approach to addressing this problem.

2.14.3 The Dimensionality of the State-Action Space

In process control problems, the operating window is a connected set. Final element adjustments are often continuous, that is $\mathcal{A}$ is continuous. This does not mean that the designer must always model $\mathcal{S}$ and $\mathcal{A}$ as continuous when designing the RL representation. In general, there are two ways to deal with these vector spaces. The first is described by Potapov and Ali (2003) as aggregation, where $\mathcal{S}$ and $\mathcal{A}$ are mapped to discretized states and actions, thereby establishing a tabular approach. Fully continuous or intermediate application may be achieved through the use of function approximation, which was discussed in Section 2.9.7. As such, continuous modelling of $\mathcal{S}$ may be combined with continuous modelling of $\mathcal{A}$. Alternatively, both vector spaces may be modelled as discrete, or only $\mathcal{A}$ may be modelled as discrete.

Another important consideration is the curse of dimensionality. Lillicrap et al. (2016) summarise this concept as the exponential increase in the size of the modelled action space $\mathcal{A}$ as the degrees of freedom in a system are increased. In a seven degree of freedom system, with each degree of freedom allowing for

three discretized actions, the size of the discretized action space $\mathcal{A}$ is 3^7. As we increase the fineness of $\mathcal{A}$'s discretization, the size increases severely.

This has led to alternative approaches to dealing with large action spaces rather than applying algorithms such as Deep Q-Network (DQN) with discretized actions – Chapter 3. Exploration of large action spaces with a DQN agent is difficult. It is important, however, that Lillicrap et al. (2016) focused on application to problems where it is considered vital not to discard any information about the structure of $\mathcal{A}$. Therefore, concerns about the curse of dimensionality may potentially be relaxed when targeting simplification of the problem and efficiency in the use of plant hours during training.

2.14.4 RL Environment Stochasticity and Its Dependence on Industry

Recall from Section 2.10 that an agent's policy π may be deterministic or stochastic (Engel, 2005; Sutton and Barto, 2018). For a deterministic policy, the RL representation and the exploration strategy are designed so that, at a particular state $\boldsymbol{S}$, the selection of the action $\boldsymbol{A}$ is always the same. The only exception occurs if the agent selects an action with the purpose of exploring (for example, ε-greedy).

A deterministic RL environment has a sparse matrix comprising $Pr(\boldsymbol{S}', \boldsymbol{S}, \boldsymbol{A})$, where having a particular action $\boldsymbol{A}$ applied to it at state $\boldsymbol{S}$ inevitably leads to the same next state $\boldsymbol{S}'$ at $T + 1$. This is often not the case in chemical processes, with the mineral processing industry being a very good example. This is because an insufficient set of sensors is typically available to provide complete information to the agent about the achieved state (Whitehead and Lin, 1995). This has an influence on the extent to which the MDP assumption of Section 2.14.1 is only an approximation as opposed to a strictly valid assumption. Industrial processes are therefore typically partially observable. As a result, the optimal policy π^* that would provide the best solution to a process control problem is a stochastic policy.

2.15 Chapter Summary

Chapter 2 provided the reader with the necessary theoretical background to understand the RL algorithms studied in this thesis. An in-depth focus on the underlying theory is aligned well with the aim and objectives of the study. While the algorithms may be argued to be relatively "simple" by experienced RL practitioners, readily accessing the necessary roots of the algorithms without significant background in machine learning is a considerable challenge to the wider community of stakeholders in its own right.

The author used this chapter to lay some of the foundations for the methodology followed to conduct the feasibility study. This methodology is described in Chapter 4 and Chapter 5. The theory of this chapter was aimed at addressing two key problems. The first is that of estimating suitable instances of the table entries for SARSA, Q-learning, and suitable entries for the critic's parameter vector $\boldsymbol{w}$ in the case of a One-Step Actor-Critic agent. The second problem is estimating a suitable parameter vector $\boldsymbol{\theta}$ for the actor of a One-Step Actor-Critic agent.

CHAPTER 3

LITERATURE REVIEW

As will be seen in this chapter, it is evident that the majority of the RL-based control methods found in literature were applied in the context of simulation studies. Hoskins and Himmelblau (1992) applied principles of RL to control long before the advent of deep learning in RL-based process control. It is observed that deep learning has enjoyed much attention in state-of-the-art RL-based process control research in recent years, and to a much smaller extent the building blocks of the field. All three branches of RL control (value-based, Section 2.8, Policy Gradient, and Actor-Critic, Section 2.9) are represented, and there is relatively little focus on methods that utilise a probabilistic model of the RL environment (model-based methods, Section 2.12).

The prevailing theme in RL-based process control research is applying available computational resources to improve asymptotic performance of the trained policy π. The literature selected for review aims to contrast the use of elementary and state-of-the-art RL-based control methods. The former refers to the use of RL representations that do not apply deep learning and aligns with the aim and objectives of this thesis (Section 1.2), while the latter refers to the use of deep neural network-based function approximation.

Interestingly, application of model-based approaches to process control problems may involve work on adaptations to MPC through the use of the probabilistic model of the RL environment in certain locations of the state space $\mathcal{S}$ (Lee, 2004; Lee and Lee, 2005).

3.1 Parallels Between RL-Based Control and Classical Process Control

Consider the often-encountered simplified time-domain block diagram of a SISO control system depicted in Figure 6. The continuous feedback controller utilises the error signal ($E(t)$) calculated as the difference between the set point ($SP(t)$) and the output of the sensor measuring the controlled variable ($CV_m(t)$). A control law is applied to the error signal after which the dynamics of the final element determines the manipulated variable ($MV(t)$) adjustments made as an input to the process. The control law output is denoted by $u(t)$. In the interest of more concise notation, the '(t)' will be omitted when referring in the text to control laws that operate in continuous time, with the exception of $E(t)$ and $u(t)$. The objective of the control system is to maintain the actual CV close to SP. This includes finding an adequate trade-off between minimising the deviation of CV from SP during transient responses, limiting excessive or physically unrealisable MV adjustment, and ensuring zero offset at steady-state.

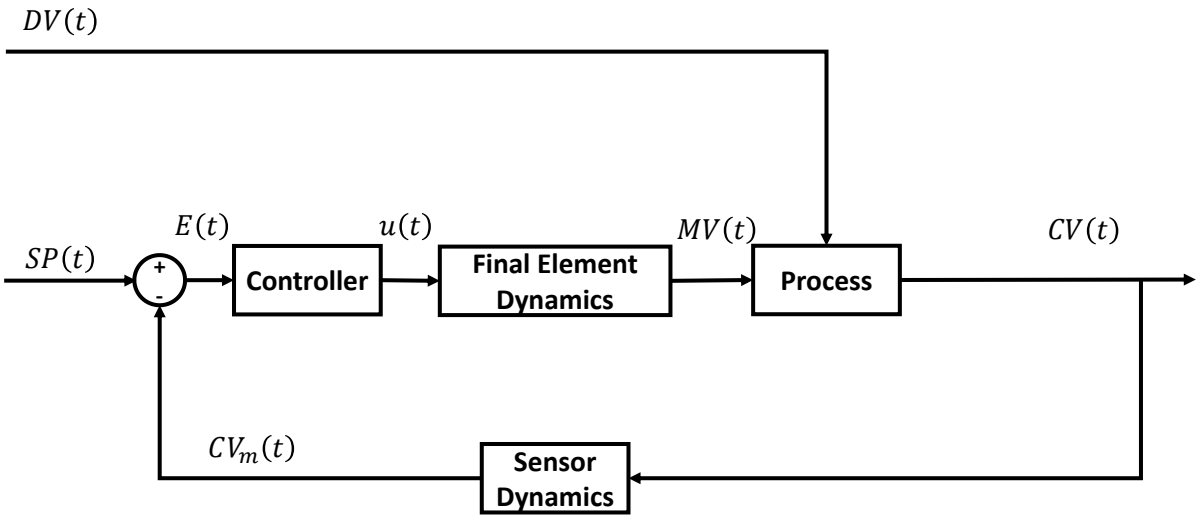


Figure 6: Classical feedback controller time domain block diagram, adapted from Marlin (2000)

In classical feedback control, the problem is typically approached by using linearization or empirical modelling, and Laplace domain modelling to determine an appropriate set of tuning constants. These tuning constants are applied to a fixed mathematical expression that maps $E(t)$ data to control signals that are subject to final element dynamics before being applied to the process. For a fixed process description, the tuning constants are typically determined to favour the type of control problem.

Skogestad and Poslethwaite (2005) describe a regulator control problem as compensating for the effects of uncontrolled exogenous input perturbations, the disturbance variables (DV). The servo problem describes the achievement of appropriate MV behaviour to ensure that the CV follows changes to the SP. A classical control law is the PID control algorithm, which is shown in recommended form in Equation [45] (Marlin, 2000). PI control is obtained by omitting the third term in the parentheses. The '$bias$' is the value of $u(t)$ required when $E(t) = 0$ to maintain the initial steady state of the process.

$$u(t) = K_c \left(E(t) + \frac{1}{\tau_I} \int_0^t E(t)\, dt - \tau_d \frac{dCV(t)}{dt} \right) + bias \qquad [45]$$

An RL-based controller, as shown in Figure 7, determines $u\big(i(\Delta T)\big)$ for $i \in \mathbb{N}$ and time domain sampling period ΔT through its policy π. The agent algorithm receives information regarding the RL environment, which is the measured state information. In contrast to Equation [45], the agent's control law is expressed in terms of the parameters of an RL representation. Iterative training of the agent requires the identification of appropriate measurements to represent the essential characteristics of the RL environment, which includes the process.

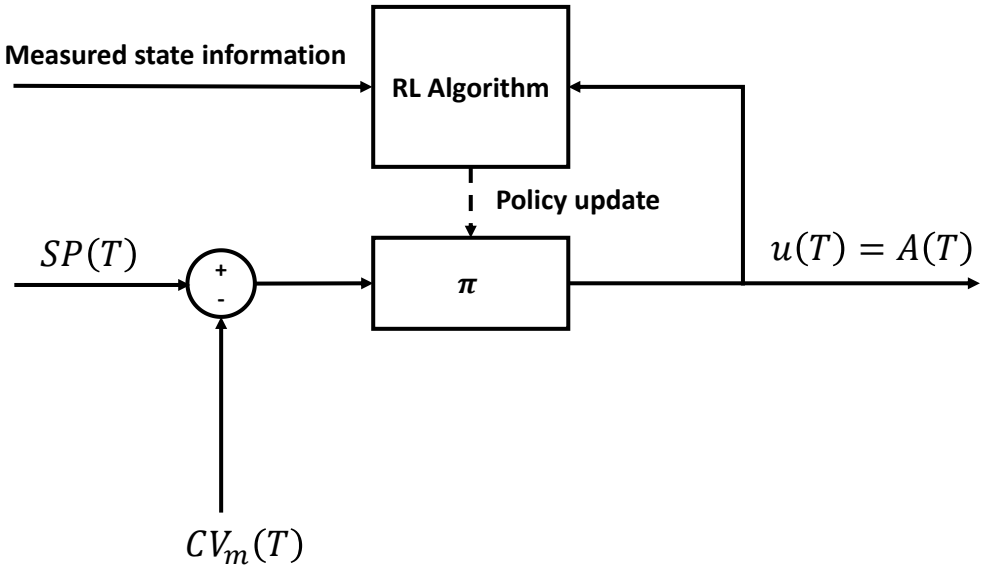


Figure 7: Generic agent that represents the control law in an RL-based controller design

3.2 Model-Free, Value-Based Applications

Value-based RL control, Section 2.8, allows the studying of tabular RL representations and the influences of hyperparameters at their highest levels. Recall from Section 2.5 that different approaches to modelling the state-action space may be used to approximate the same optimal action-value function $Q_\pi^*(\boldsymbol{S}, \boldsymbol{A})$ at different resolutions, once the RL representation's parameters have converged. The use of a discretized state-action space clearly constitutes a low-dimensional viewpoint, but holds merit in terms of understanding the interaction of RL agents with RL environments of interest to chemical process control. It also aids in the training of interpretable RL representations.

Since elementary RL is applied in this work, the review of Section 3.2.1 is the most extensive. The literature reviewed in this chapter is used to identify themes relevant to the feasibility study conducted. These themes are summarised in Section 3.8.

3.2.1 Elementary Application

Syafiie et al. (2008) approach SISO, RL-based control with a tabular RL representation and Q-learning in their Model-Free Learning Control (MFLC) approach. The authors select $0 < \alpha \leq 1$, with $\alpha = 0.1$ being typical and state that this range avoids oscillatory behaviour during value updates. The discount factor γ is set close to one with representative examples of $\gamma = 0.9$ for buffer tank control and $\gamma = 0.98$ for laboratory scale pH control in their study. The use of γ close to one while showing cognisance of its theoretical constraints is intuitive, since an RL agent must incorporate the long-term dynamic effects of its

behaviour while making decisions which are relevant to a process control problem (Hafner and Riedmiller, 2011).

In MFLC, RL representation design is approached from a minimalistic point of view. Since the available handles to be used as actions are fewer for process control problems than is the case with many other RL applications such as those found in robotics, the curse of dimensionality described by Lillicrap et al. (2016) is argued by the authors not to prevent the application of tabular RL representations. The feedback control error is discretized, which is also the only component comprising the scalar state, S. It is stated by Syafiie et al. (2008) that, without compromising generality, the goal state in MFLC is assumed to be at the centre of the symmetrically discretized states. The authors state that the fineness of the state discretization establishes a trade-off between improved efficiency in the agent's use of operating hours and the description of all the characteristics of the action-value function of a particular control problem. This trade-off is incorporated through the selection of a parameter for the equal number of symmetrically discretized states available for the positive and negative feedback control errors, respectively. The goal state contains $E(T) = 0$.

Reward functions typically used in MFLC may be binary with a positive reward of $+1$ if S' is in a predefined goal state and a penalty of -1 if S' is outside the goal band (Syafiie et al., 2008). Alternatively, a penalty of -1 is used at intermediate values of S' and a large penalty of -10 might be used if the feedback control error is excessively large in its magnitude. A reward of $+10$ may in this case be received if S' is within the goal band. Syafiie et al. (2008) hereby manage to illustrate that the control objectives of a particular process control problem need not be communicated explicitly in the reward function when designing RL-based controllers. The rewards can communicate to the agent that it must ensure CV is maintained close to SP irrespective of whether a servo or regulator problem is encountered. The relative magnitudes of penalties and rewards need to be sensible to enable efficient learning.

In the work of Syafiie et al. (2008), the action space $\mathcal{A}$ is represented by a discrete set of scalar actions that depend on the instance of S encountered. The sampling period ΔT is implemented by setting the control law output at T, $u(T)$, equal to $u(T-1) + k(a_w - A)$ for action selection A. This form for the control law output is very close to a discrete time approximation of a continuous low-pass filter where the filter time constant multiplied with the derivative of the filtered output with respect to time is set equal to the filtered output subtracted from the unfiltered output. The parameter k is used to adjust how aggressive the controller is, while a_w is called the wait action. The wait action is the action corresponding to maintaining the previous control signal as $(a_w - A) = 0$ if $A = a_w$. A more aggressive controller implies that the action selection frequencies of the agent have a more immediate effect on the current control law output $u(T)$. When S is within the goal band, a_w is provided to the agent as a way of forcing the agent to adjust its policy π to establish zero steady-state offset within the control tolerance and the incremental change in the control law's output.

The significance of the expression for $u(T)$ used by Syafiie et al. (2008) is twofold. Firstly, the authors illustrate that the agent's discrete action selections may be processed in the RL environment to achieve intermediate selections. Secondly, the process time represented by a transition from T to $T+1$, the sampling period ΔT, does provide an additional handle in RL-based control design. This handle provides an interesting approach which may presumably allow the designer to reduce operational data requirement at the expense of a "coarser" control law, in addition to the primary goal envisaged by the authors – manipulating how aggressive the controller is.

Syafiie et al. (2008) have shown that reasonable feedback control may be achieved through the use of value-based RL control methods with tabular RL representations. An ε-greedy exploration strategy was used with $\varepsilon = 0.10$. Sutton and Barto (2018) indicate that Q-learning is an off-policy algorithm, which implies that it is suitable for training on historical data (or to a less practical extent training in parallel to an existing control law). The Q-learning update rule, Equation [21], is very intuitive and is equivalent to SARSA if a greedy policy is followed by the SARSA agent.

The use of $E(T)$ as the scalar state S in MFLC is likely inspired by the use of $E(t)$ in classical PI/PID control. From a machine learning perspective, the definition of scalar components contained in a vector $\boldsymbol{S}$ comprising the state space $\mathcal{S}$ must be made in a manner which improves the accuracy of the MDP model's description of the RL environment. The use of $E(t)$ is a reasonable first approach to understand the behaviour of a tabular RL agent applied to the control of certain processes.

Brujeni et al. (2010) applied SARSA with a k-nearest neighbour generalisation technique for continuous state space $\mathcal{S}$ modelling in combination with quadratic function approximation to the tuning of a PI controller (in an RL-based adaptive control scheme reminiscent of gain scheduling), and compared the resulting PI controller to the control performance achieved using an Internal Model Controller (IMC). The process studied was a Continuous Stirred Tank Heater (CSTH) which is known to be controlled well by PI and PID controllers.

Their training scheme is comprised of two stages. Firstly, an empirically identified Laplace domain model is used as the RL environment in on-policy training. Secondly, the first-guess policy is refined by allowing the RL agent to interact with the experimental setup while adhering to physical constraints with regard to final element adjustment. Servo and regulator problem behaviours were illustrated by the authors. The feedback control error defined the state S, and the components of the action $\boldsymbol{A}$ were defined as the proportional and integral gains. A penalty of $-\big(E(T)\big)^2$ was used during training.

An ε-greedy exploration strategy was used throughout, with ε decaying from unity in a $\frac{1}{T}$ fashion. The discrete action sets for the RL-based and IMC controllers were the same – illustrating that controllers need to be limited to the same extent for sensible comparison. A step size $\alpha = 0.25$ was used throughout. Different SP values changing within a fixed symmetric range were used during each episode of training. After obtaining an initial estimate of a suitable policy π, it was refined through training on laboratory

equipment with the same training scheme. The presence of a sufficiently quantitatively accurate process model renders potential concerns regarding operational data requirement irrelevant.

Interestingly, the use of RL to establish an alternative controller tuning paradigm was pointed out by Brujeni et al. (2010) as an approach that would have a less severe impact on process performance during the application of exploratory actions to the RL environment. Presumably, this is because including a PI control law in the RL environment serves a dimensionality reduction role, the problem faced by the RL algorithm is then significantly simplified, and unintended behaviour that is not safe may be removed by adjusting the ranges of the tuning constants that the RL agent may apply to the RL environment. By training the RL agent initially on an empirically identified first-order-plus-time-delay (FOPTD) model of the process, the authors are stressing that domain knowledge must be leveraged even when applying model-free RL control methods. This specific example limits the agent to simulated experience, but is considerably better than training the agent from scratch provided that a suitably accurate model is available.

Brujeni et al. (2010) illustrated in their simulation results that a PI controller tuned using RL approximates the aggressive IMC controller tuning when trained sufficiently. This is significant as the ability to understand RL-based control from the perspective of another control method is a step towards the development of RL-based controllers that are more easily accessible outside the machine learning community, and display improved interpretability. The focus on SP adjustment during agent training illustrates that, as a minimum, adaptive RL-based control needs to be exposed to perturbations in its definition of the state space during training. The initial plant-model mismatch when connecting the first estimate of the policy π to the real-world system caused oscillations about the desired SP.

Ramanathan et al. (2018) applied Q-learning on-policy (in the same vein as the work of Syafiie et al. (2008)) to the control of level in a conical tank with a constant SP. Such process control problems display significantly non-linear dynamics which are caused by the time-varying cross-sectional area and the requirement to achieve precise control. A discretized state-action space was used, and the action-value table was trained initially on a model of the process to prevent damage to the experimental setup when connecting the RL agent to the experimental feedback control loop. Approaches to the control of conical tanks often entail linearized modelling of system parameters, fuzzy control, or neural networks. The authors assumed that an MDP sequential decision making framework is adequate for the process for their definition of the state-action space. The outlet valve is kept at a constant setting during training, while discrete inlet valve openings are available to the agent as actions.

The authors continue to emphasize the importance of preventing the use of an excess of parameters that need to be tuned and set $\gamma = 0.99$ for Q-learning. It is important to realise that long-term rewards must be prioritised, and the use of $\gamma = 1$ does not compromise the properties of the value functions provided that termination of each episode is ensured when following each policy π instantiated during RL agent training (Jaakkola et al., 1994; Sutton and Barto, 2018). While this may in general be achieved by defining a fixed state that terminates a training episode, it may also be ensured by predefining the length of a training

episode. The definition of S as the position of liquid in the tank is only feasible for a constant SP application as was the case in their work, since error signal feedback is required for control to work.

The authors' graphical depiction of how scalar states are mapped to scalar actions for the trained agent illustrates how the agent selects discrete actions at frequencies that bound the optimal selections within the constraints of its discretization. If the conical tank's height is significantly above SP, the inlet valve must be closed frequently, and vice versa. Between these extremes the agent is uncertain as to what constitutes ideal behaviour and it bounds the ideal final element adjustment by selecting actions across the range of available actions. The reward function used by Ramanathan et al. (2018) was the negative of the absolute value of the error signal.

Testing for disturbance attenuation involved changing the inlet flow rate or removing liquid while the constant SP was maintained. A trade-off between discretization fineness, performance, and operational data requirement was noted by the authors. The authors manage to illustrate that even a minimalistic state space definition with discretized state-action space modelling and deterministic policy can show a level of robustness to SP and DV changes. The authors mention that including more actions and training at different outlet flow rates may help with improving robustness while allowing for a larger operational data requirement.

The trained agent consistently achieves heights that oscillate about the desired SP. This is a result of direct application of a discretized state-action space when interacting with the conical tank system, potentially accompanied by plant-model mismatch.

3.2.2 State-of-the-Art

Hafner and Riedmiller (2011) developed the Neural Fitted Q-Iteration with Continuous Actions (NFQCA) algorithm that extends concepts of value-based RL control to a continuously modelled action space $\mathcal{A}$. The greedy Bellman target is used, while a neural network RL representation is used for the critic. A continuous penalty function is applied and is an extension of the use of a binary reward function. In the continuous penalty function, the error signal $E(T)$ is scaled using a hyperparameter, after which a binary cost is multiplied with the square of the hyperbolic tangent applied to the scaled $E(T)$. A dataset for batch learning, $\mathcal{D}$, is created online and contains entries consisting of $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{S}')$ tuples obtained during transitions from T to $T+1$. Two phases alternate during training of the RL agent. Firstly, the recent policy is used to control the process and expand $\mathcal{D}$. This is followed by updates to the critic and actor.

The NFQCA algorithm applies a fitted Actor-Critic method since the policy network of the actor is used to facilitate Q-learning rather than apply a Policy Gradient method. The most recent instance of the neural network representation of the policy π is assumed to provide the greedy action at time step T. This provides a promising framework for value-based control in continuous state-action spaces where finding the greedy action may not be possible without intractable computational expense. Learning is performed with different initial input values to aid in the prevention of local minima when applying the non-linear

parametric function approximation, and computational efficiency is promoted by the approach to Bellman target evaluation (Hafner and Riedmiller, 2011).

The authors applied NFQCA to the control of a Heating, Ventilation, and Air Conditioning (HVAC) task which was represented by a non-linear process model. The coupled dynamics of each unit of the process makes this a difficult task for classical control owing to the necessity of variable gains. The authors successfully trained the agent within 163 interaction trajectories and a total of 32 600 MDP steps. Every 200 steps were considered as a batch of data to include in $\mathcal{D}$. The agent's state $\boldsymbol{S}$ consisted of seven components (boiler feed water flow rate, temperature of inlet boiler water, outlet and inlet air temperatures, temperature of outlet boiler water, air inlet flowrate, and error signal $E(T)$). Valve state was provided to the agent as action, and deep neural networks were applied to represent both $\tilde{Q}_{\pi}(\boldsymbol{S}, A)$ and π.

Mnih et al. (2015) tackle the problem of learning off-policy from sensory inputs without pre-identified, task-specific characteristics/features of the state-space, and a discrete action space. Their algorithm was tested in a benchmark game theory RL environment. Their ε-greedy Deep Q-Network (DQN) algorithm addresses the off-policy instability issues of Q-learning, Section 2.8.2, by introducing experience replay for value-based control to the RL field, and defining separate target and behavioural policies. Experience replay is instantiated through the generation of a buffer of previous training data tuples. From the replay buffer, a random minibatch of $(\boldsymbol{S}, \boldsymbol{A}, R, \boldsymbol{S}')$ tuples are sampled at each training step, and gradient descent updates based on Q-learning are performed on the behavioural parameter vector.

In DQN, the target network is only updated periodically by setting its parameters equal to the parameters of the current convolutional network representing $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$. Correlations in observation sequences are removed and those arising from the sensitivity of π to $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ are smoothed by the random sampling from the replay buffer. Sampling from a replay buffer removes the assumption that network training samples are independently and identically distributed, and improves algorithm computational efficiency. The data distribution assumption does not hold during exploration (Lillicrap et al., 2016), and therefore a solution was sought during the development of DQN (Mnih et al., 2015). The slower updating of the target network reduces correlations between the target and the action-values.

Hessel et al. (2018) highlight their concern about the limitations of ε-greedy exploration in the DQN algorithm. Their predominant concern is that it is difficult to learn alternative policies that accommodate rewards only far in the future. More specifically, the concern was exemplified by a game studied by Mnih et al. (2015) that required many actions before obtaining the first reward. Various different extensions to DQN were proposed in the approach of Hessel et al. (2018).

It is worth pointing out that DQN was initially designed and tested on RL environments where perceptive pixels were provided as input without clear definition of RL environment characteristics by the designer. That is, the score in a game, sensory input, and available actions were provided along with the scalar penalty signal.

3.3 Model-Free Policy Gradient and Actor-Critic Applications

The success of DQN inspired the extension of some of its principles to a Policy Gradient method for MDPs that are assumed to have a fully observable state. This is not a limitation, but refers to the use of a deterministic policy with noise added for the purpose of exploration. This led to the development of the Deep Deterministic Policy Gradient (DDPG) algorithm by **Lillicrap et al. (2016)**. The techniques of target and behavioural networks, and replay buffer sampling (using a uniform distribution in DDPG) are incorporated in the algorithm in addition to batch normalization. Direct application of DQN to continuous action spaces is not feasible as the evaluation of the greedy Bellman target becomes intractable.

Lillicrap et al. (2016) state that components of the state-action space can have ranges that differ significantly owing to the use of different units. This is an impediment to proper generalization and parameter learning owing to the distortion of basis functions. The basis functions can be scaled manually. An additional problem in the training of neural networks is a shift in the training data population distribution (and indirectly of the data sent to the different network layers) known as covariance shift (Ma et al., 2019). To address the scaling problem while minimizing covariance shift, Lillicrap et al. (2016) introduce batch normalization where each dimension of the minibatch data sample is normalized to have unit mean and variance. A soft-max parametric policy π neural network and a separate $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ neural network are trained. Both networks' targets are updated at much slower rates than the behavioural networks to ensure training stability.

Ma et al. (2019) applied DDPG to the control of a simulated multi-input nonlinear semi-batch polymerization process with large time-delay. The authors stress that the application of RL methods to process control problems required problem-specific design specifications. Average molecular weight (MW) is the CV of the process, and the MDP assumption is inadequate to achieve acceptable performance for this specific control problem if the definition of the state space $\mathcal{S}$ does not incorporate historical information. Therefore, the first component of $\boldsymbol{S}$ is defined to include historical MW measurements. Corresponding MW targets constitute the second component of $\boldsymbol{S}$. Policy gradients are implemented through an inverting gradient method that removes the necessity of saturating action selections when continuous actions outside of the available range are selected during training. Each action $\boldsymbol{A}$ sampled from the continuously modelled action space $\mathcal{A}$ consists of two components, namely the initiator and monomer flow rates.

The authors define their reward function as a piecewise continuous function that is a reward linear in time if the MW is within the tolerance range. Otherwise, a squared penalty in terms of MW deviation is added to a linear time-dependent penalty. Their intention was clearly to communicate control objectives such as control within a tolerance band with a continuous reward function that prevents reward sparsity and causes more severe reinforcement signals to occur as T increases. The authors define the sampling period ΔT as 2.5 min with the goal of incorporating the sampling delay present in the measurement of the CV.

Shipman and Coetzee (2019) used Advantage Actor-Critic (A2C) as an on-policy method to tune a classical PI controller for a stochastic instance of a discrete first-order dynamic model. The process model's gain and time constant were kept within predefined ranges. This algorithm's state-value target is expressed using the same principles as those in the One-Step Actor Critic algorithm of Section 2.9.5 (Mnih et al., 2016). The authors used a continuous saturating reward function scaled to lie between zero and one. In the work of Shipman and Coetzee (2019), the reward function was developed to encode concepts of squared-error deviation from SP and penalizing excessive MV manipulation (excluding proportional mode responses to DV and SP changes). The discrete action space modelling of A2C was modified to accommodate continuous actions by representing the probability distribution used to select each action by a mean value and a variance (pdf) rather than generating discrete action probabilities (pmf). The policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ was approximated using the method of Mnih et al. (2016) and was accompanied by regularization to prevent overfitting, Section 2.13.

The training scheme included random changes of plant dynamics and either exposing the agent to the full complexity for which it was trained or a gradual increase in task complexity. The state definition included the noisy CV measurement of the SISO process, the SP, the control law signal $u(T)$, and tuning constants. The MDP time steps of the agent are purposefully scaled with respect to the true process, presumably to improve the ease with which the agent observes the causal relationship between its actions and the rewards received.

A total of 100×10^6 training time steps were used in all the training schemes of Shipman and Coetzee (2019). The sampling period ΔT for the agent was defined as 60 seconds. The sense of the controller is assumed known in the controller gain ranges provided to the agent. The agent time steps of 60 seconds is very large in comparison to the time constant τ_p of the model studied which has ranges of 0.5 to 15.5, 10 to 45, and 1 to 4 seconds during training.

Marlin (2000) states that, for a FOPTD model, the time required to reach 63.2 % of the final CV value for a step change in the input is the process time constant τ_p. This implies that the agent was only fully exposed to the RL-environment's dynamics for an unknown fraction of the training time steps, while the remainder of the agent's training comprised of steady-state optimization of the process. This does not compromise the value of the results generated by Shipman and Coetzee (2019) in any way. The observation made helps to identify an important aspect of RL-based control, namely that the sampling period ΔT should be selected in a way that ensures that the RL agent is sufficiently exposed to the dynamics of the process that it needs to control.

3.4 Model-Free, Bayesian RL

An alternative function approximation paradigm was proposed by **Engel (2005)** through the application of Gaussian Processes (GPs) coupled with ε-greedy exploration to establish a SARSA variant where the approximation $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$ is determined online based on Bayes' rule. Variants of this concept are coined

Gaussian Process Temporal Difference (GPTD) learning methods, and may be implemented with a parameter vector $\boldsymbol{w}$. Stochastic and deterministic GPTD implementations have been developed.

Ghavamzadeh et al. (2015) have surveyed model-free applications of value-based, Policy Gradient, and Actor-Critic methods that use Bayesian inference. GPTD was adapted with posterior sampling by **Fan et al. (2018)** to yield Gaussian Process Posterior Sampling Temporal Difference (GPPSTD). **Deisenroth and Rasmussen (2011)** developed the probabilistic inference for learning control (PILCO) model-based RL control algorithm that promotes efficiency during training for different control tasks by calculating analytic policy gradients through GP function approximation.

The small extent to which Bayesian, model-free RL methods have been applied to process control problems is noteworthy. This may be attributed in part to the ability of frequentist Actor-Critic parametric function approximators to incorporate agent uncertainty naturally despite not applying Bayesian principles. The selection of a prior distribution $p(\boldsymbol{w})$ relies on a coupling of domain knowledge and statistical intuition with no "routine" procedure that may be used for standardising the approach. Different designers will make significantly different assumptions regarding the data distribution of RL agent training and such assumptions would further rely on the specific process considered.

Evaluation of $\boldsymbol{w}$ in the GPTD methods of Engel (2005) may potentially require the inversion of matrices that may be ill-conditioned despite approaching the control problem using a low-dimensional state-space. It is an important realization that the number of state components (the number of components comprising $\boldsymbol{S}$) is typically low. The concerns reiterated by Lillicrap et al. (2016) in the context of the PILCO algorithm, namely that it is impractical for high-dimensional problems, is only considered relevant in the process control setting if one agent with one GP function approximator is used to manipulate the final elements of many control loops on a large plant. This is substantiated by the success of applying tabular methods to certain SISO control problems, Section 3.2.1. Employing one agent to manipulate the final elements of many control loops would cause the number of components contained in $\boldsymbol{S}$ to increase significantly.

3.5 The Importance of Real-World Data

The use of dynamic process models is invaluable when conducting theoretical feasibility studies of new process control developments. The results of such work must be viewed conservatively since the extensive success of RL applied to problems posed by games (see, for example, Mnih et al. (2015)) may, in part, be attributed to the ability of the computational agent to discern "fixed" rules that are prevalent in such RL environments.

Engel (2005) emphasizes that both intrinsic and extrinsic uncertainty exists in an RL control problem. Uncertainty results from the mismatch between the RL design choices and their suitability to the process being controlled. Intrinsic uncertainty depends on whether the RL environment's state transitions are occurring randomly. In contrast, extrinsic uncertainty results from the subjective nature of RL representation design, the definition of states and actions and the reward function, and the RL algorithm

applied. Recall from Sections 2.14.1 and 2.14.4 that partial observability arises when a stochastic problem behaviour is induced by insufficient information contained in the definition of $\boldsymbol{S}$ (Sutton and Barto, 2018; Moerland et al., 2021).

Unfortunately, the use of dynamic process models as opposed to real-world data results in a significant bias with respect to uncertainty. The presence of covariance shift, Section 3.3, is likely also more prevalent when deep neural networks are used. Intrinsic uncertainty cannot be reduced, while the external uncertainty exacerbated by a limited operational data set can (Moerland et al., 2021).

This does not mean by any means that simulation-based study does not allow for a sensible feasibility study. It is essential to consider the operational data requirement for simulated systems with the assumption that, for many processes, this requirement would be further increased owing to further design uncertainty. New theoretical developments in RL-based process control also cannot be safely applied directly to real-world systems that pose a safety or economic risk.

3.6 Overview of Literature Included in the Review

Table 3 provides a high-level overview of the literature reviewed in this chapter. The entries of Table 3 follow the review of the chapter chronologically, where multiple-input, multiple-output (MIMO) control problems are indicated in brackets in the relevant cells.

Table 3: Characteristics of RL-based control approaches found in literature

Author	Tabular?	Value-based?	Policy Gradient?	Actor-Critic?	Bayesian RL?	Model-based RL?	SISO Problem for Agent?	Applied to Chemical Engineering?	Real world data?
Syafiie et al. (2008)	Yes	Yes	No	No	No	No	Yes	Yes	Yes
Brujeni et al. (2010)	Yes	Yes	No	No	No	No	Yes	Yes	Yes
Ramanathan et al. (2018)	Yes	Yes	No	No	No	No	Yes	Yes	Yes
Hafner and Riedmiller (2011)	No	Yes (fitted Actor-Critic, Section 3.2.2)	No	No	No	No	Yes	Yes	No
Mnih et al. (2015)	No	Yes	No	No	No	No	N/A	No	N/A
Hessel et al. (2018)	No	Yes	No	No	No	No	N/A	No	N/A
Lillicrap et al. (2016)	No	No	Yes	No	No	No	N/A	No	N/A
Ma et al. (2019)	No	No	Yes	No	No	No	No (MIMO)	Yes	No
Shipman and Coetzee (2019)	No	No	No	Yes	No	No	Yes	Yes	No
Engel (2005)	No	Yes	No	No	Yes	No	N/A	No	N/A

Author	Tabular?	Value-based?	Policy Gradient?	Actor-Critic?	Bayesian RL?	Model-based RL?	SISO Problem for Agent?	Applied to Chemical Engineering?	Real world data?
Ghavamzadeh et al. (2015)	No	Yes	Yes	Yes	Yes	Yes	N/A	No	N/A
Fan et al. (2018)	No	Yes	No	No	Yes	No	N/A	No	N/A
Deisenroth and Rasmussen (2011)	No	No	Yes	No	Yes	Yes	N/A	No	N/A
(Lee, 2004; Lee and Lee, 2005)	Yes (different discretization strategies considered)	Yes (focus on approximate DP and Q-learning)	N/A	N/A	No	No	Yes	Yes	No

3.7 Placing RL-Based Process Control in Context

This thesis is concerned with investigating elementary RL methods applied to single loop base layer control. The author would, however, like to encourage the reader to consult literature that clarifies the appropriate positioning of RL in process control. Two papers are highlighted in this section.

A recent review that includes a comparison of RL-based control with traditional optimal control methods, a summary of important developments in deep RL, current shortcomings of such methods, advantages and disadvantages, and other RL-based control approaches are given by Nian et al. (2020). Further, this paper provides detailed examples of applying Q-learning to track the SP output pressure of a pumping system by adjusting the rotations per minute of the pump's impeller. A particularly interesting field of application pointed out by Nian et al. (2020) is sequential anomaly detection via RL, where loss incidents on a plant are detected in advance to achieve proactive risk management. This may potentially help with planning and scheduling activities on a plant.

Shin et al. (2019) compare RL with MPC as an instance of mathematical programming and, alongside this comparison, suggest the use of RL and mathematical programming that use a receding horizon in a complementary fashion. Analogously, considering the incorporation of model-based RL techniques in RL-based control approaches may be beneficial.

3.8 Summary

Chapter 3 aimed to establish more thoroughly the niche of this thesis. Selected elementary, value-based control methods that have found application to process control problems were discussed. The review of state-of-the-art value-based, Policy Gradient, Actor-Critic algorithms, and the Bayesian viewpoint may seem excessive owing to the exclusivity with which elementary algorithmic principles have been applied in this work. This was, however, necessary to draw out commonly occurring themes, to form an opinion on the approach to a feasibility study, and identify gaps in the existing literature which are visited in the recommendations of Section 7.7.

The common themes identified include prioritizing the leveraging of computational resources to train sophisticated deep learning function approximators, and the explicit communication of control objectives through various continuous reward functions. The agent is required to approximate the optimal policy π^* without first considering what the minimum and most general definition of $\boldsymbol{S}$ is and where domain knowledge may be injected into the system to inform RL representation design. It is argued in this thesis that, to evaluate the present industrial feasibility of RL-based control, one has to formalise these concepts as simply and generally as possible and evaluate whether an agent can be trained within conservative operational data constraints. If reasonable operational data requirement cannot be achieved for the most elementary on-policy algorithm (SARSA, Table 1) when applied simply to challenging process control problems, state-of-the-art algorithms' operational data requirement should be critically assessed if the

number of parameters is much larger in the deep learning function approximation used, Section 6.5. Operational data requirement accompanies other feasibility criteria presented in Section 5.7.

The proposed RL methodology of Chapter 5 draws significantly on lessons learned by studying the work of Syafiie et al. (2008), Brujeni et al. (2010), Lillicrap et al. (2016), Ramanathan et al. (2018), Sutton and Barto (2018), Ma et al. (2019), and Shipman and Coetzee (2019). Overall, it is the opinion of the author that the intersection of elementary RL and process control has not received sufficient attention. There thus exists gaps in the literature in terms of identifying impediments to industrial RL-based control that emerge at this level. This is understandable given the recent advances in computational power and data science which motivate making the study of asymptotically achievable control performance a priority.

CHAPTER 4

CASE STUDIES

This chapter describes the simulated self-regulatory water tank model (Section 4.1), Van de Vusse reaction scheme model (Section 4.2), and grinding circuit model (Section 4.3) used to conduct the feasibility study of RL-based process control.

The contribution of including Case Study 1 is its simplicity from a process control perspective. The dynamics are approximated well by a FOPTD model and the system has an operating window that is very intuitive. This ensures that any potential complexity is prevalent only in the control design. Control concepts may therefore be tested without the engineer being concerned about nuances caused by the operating window's shape, or process dynamics that may result in instability when applying classical feedback control (Marlin, 2000).

The reactor simulated in Case Study 2 has dynamics and stability properties that prevent the successful application of classical PI and PID control. It is the prevalence of significant uncertainty about a nominal process model that poses a practical challenge to control in the mineral processing industry. Case Study 3 is a simulated investigation of a robust control problem.

4.1 Case Study 1: Self-Regulatory Water Tank

4.1.1 System Description and Model Summary

A model was derived for the level control problem depicted in Figure 8 by writing an overall material balance, as shown in Equation [46]. In the level control problem, the height of liquid in a tank needs to be maintained at the desired value (SP) while being subjected to changes in the flow rate of liquid entering the system (DV). This is achieved by manipulating the degree to which the valve is open (MV).

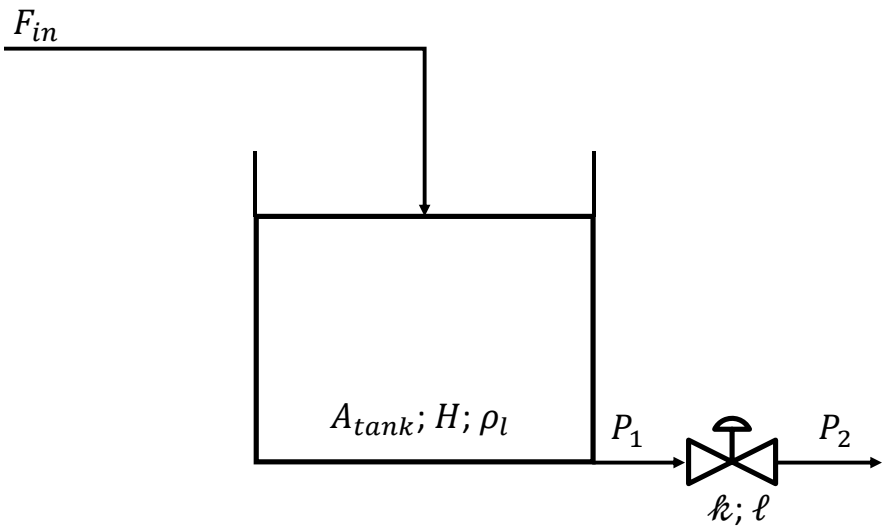


Figure 8: Illustration of simulated process used for Case Study 1

$$\frac{dm_{tank}}{dt} = \dot{m}_{in} - \dot{m}_{out} \qquad [\ 46\ ]$$

With the exception of $\dot{m}_{out}$, Equation [46] may be rewritten in terms of liquid density (ρ_l), the volumetric inlet flow rate of water (F_{in}), and the cross-sectional area of the water tank (A_{tank}), yielding Equation [47]. To obtain an expression for $\dot{m}_{out}$, it is assumed that turbulent flow conditions occur while water flows through the valve and that an orifice or nozzle discharge coefficient is applicable. Green and Perry (2008) state that the outlet mass flow rate may then be expressed in terms of the valve discharge coefficient, the liquid density, and the static pressure loss across the valve. In the water tank model, the characteristic valve discharge coefficient and the fraction of valve opening are represented by a lumped parameter $\mathcal{k}$. Substituting this outflow term into Equation [47] yields Equation [48].

$$A_{tank}\left(\frac{dH}{dt}\right)\rho_l = (\rho_l F_{in}) - \dot{m}_{out} \qquad [\ 47\ ]$$

$$A_{tank}\left(\frac{dH}{dt}\right)\rho_l = (\rho_l F_{in}) - \mathcal{k}\sqrt{\rho_l(P_1 - P_2)} \qquad [\ 48\ ]$$

Lastly, the pressure drop term in Equation [48] is approximated by expressing hydrostatic pressure in terms of fluid properties and incorporating a pressure loss parameter ℓ. There are thus two model parameters, $\mathcal{k}$ and ℓ, which may be used as handles to fit the model to a particular process, keeping in mind that simplified pressure drop modelling is used and that valve characteristics were assumed when writing the outflow term. The final model is given in Equation [49], where x is the fraction valve opening and c_v is the valve discharge coefficient. The value of c_v was selected so that a sensible range of valve openings are considered during Case Study 1. Note that the pressures before and after the valve (P_1 and P_2) and ρ_l are not part of Equation [49].

$$A_{tank}\left(\frac{dH}{dt}\right) = (F_{in}) - c_v x\sqrt{H}\ where\ x \in [0,1]\ and\ \tilde{\mathcal{k}} = \frac{\mathcal{k}}{x} = \frac{c_v}{\sqrt{g(1-\ell)}} \qquad [\ 49\ ]$$

Table 4 provides a summary of the variables of the water tank model. Table 5 summarises the parameters and Table 6 provides the steady state of the model.

Table 4: Summary of the variables in the water tank model

State variables	**Disturbance variable (*DV*)**	**Controlled variables (*CV*)**	**Manipulated variables (*MV*)**	**Measured variables**
H	F_{in}	H	$\not{k}$	H
$Degrees\ of\ freedom = 1 - 1 = 0$				

Table 5: Parameter list for the water tank model

Parameter	**Parameter name**	**Numerical value**	**Unit**
A_{tank}	Cross-sectional area of the water tank	1.8	m^2
g	Gravitational acceleration	9.81	$\frac{m}{s^2}$
ℓ	Pressure loss parameter	0.5	–

Table 6: Steady state of the water tank model

State variable	**Name of variable**	**Steady state**	**Unit**
F_{in}	Inlet volumetric flow rate	31.67×10^{-4}	$\frac{m^3}{s}$
H	Height of the liquid in the tank	1.5	m
$\not{k}$	Lumped parameter proportional to the product of fraction valve opening and valve discharge coefficient	1.17×10^{-3}	m^2
c_v	Valve discharge coefficient	0.01	$\frac{m^{2.5}}{s}$
x	Fraction valve opening	0.259	–

4.1.2 System Inputs

The input to the water tank model consisted of the inlet flow rate F_{in}, which was kept at a constant setting of 190 $\frac{L}{min}$ ($31.67 \times 10^{-4}\ \frac{m^3}{s}$). Recall from Section 2.8.1 that training was implemented using episodes, where each episode is a predefined number of time steps and its use is motivated by underlying assumptions

regarding the nature of the state-action space, the MDP, and the performance objective used to adjust RL representation parameters in the case of Policy Gradient and Actor-Critic methods. Each episode started with $H = 1.4\ m$, and a constant SP of $1\ m$ or $2\ m$ was maintained for the duration of an episode. The SP was obtained by sampling one of the two integers with equal probability.

4.1.3 Classical Controller for the System

A PI controller was tuned and applied to control the height of liquid in the water tank. The FOPTD tuning correlations given by Marlin (2000) were used. The tuning constants used are shown in Table 7, where the constants are expressed for the process transfer function $\frac{H(s)}{\not{k}(s)}$.

Table 7: PI controller tuning settings for the model of Case Study 1

Controller parameter	Numerical value
K_C	$-3.84\frac{m}{m^2}$
τ_I	$600\ s$
$bias$	$1.17 \times 10^{-3}\ m^2$

4.2 Case Study 2: Van de Vusse Reaction Scheme

4.2.1 System Description and Model Summary

The second simulated case study is the Van de Vusse reaction scheme. The reactor in which the reactions occur is illustrated in Figure 9. Four compounds are present in the reactor. The desired product, cyclopentenol (B), is produced by an electrophilic hydration process with an acid catalyst. The reagent is cyclopentadiene (A), and two unwanted side reactions occur. The first is the conversion of B to cyclopentanediol (C), while the second is a parallel conversion of A to dicyclopentadiene (D). The reaction scheme is given in Equations [50] and [51] (Chen et al., 1995).

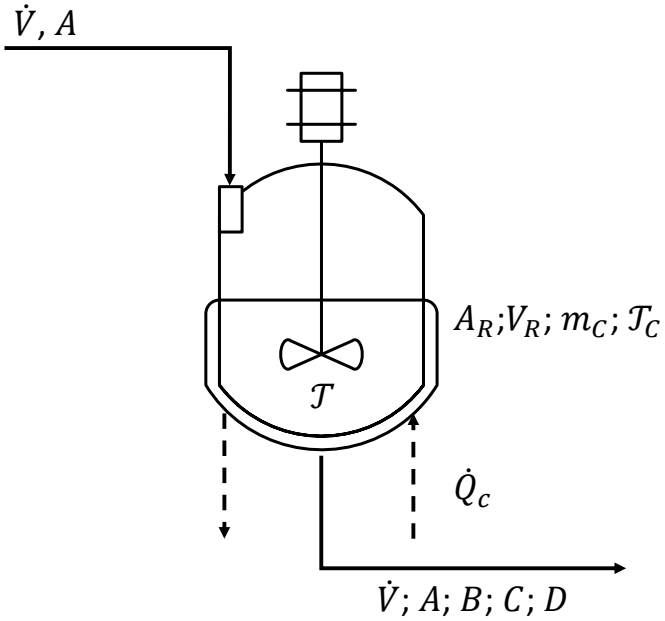


Figure 9: Illustration of the reactor used for Case Study 2, redrawn from Chen et al. (1995)

$$A \overset{k_1}{\rightarrow} B \overset{k_2}{\rightarrow} C \qquad [\ 50\ ]$$

$$2A \overset{k_3}{\rightarrow} D \qquad [\ 51\ ]$$

The process model is shown in Equations [52] through [55] and consists of mole balances performed on the reagent (A) which is fed to the reactor at inlet temperature $\mathcal{T}_0$, and the desired product (B), accompanied by energy balances for the process fluid in the reactor and for the coolant. Heat may be removed from the coolant by an external heat exchanger ($\dot{Q}_C$) in addition to the heat transfer between the process fluid and the coolant in Equation [55]. The heat removal rate $\dot{Q}_C$ is maintained at a constant value of $-1\ 113.5 \frac{kJ}{h}$ (Chen et al., 1995). The temperature-dependent reaction rates are assumed to follow the Arrhenius law, as shown for reaction i in Equation [56].

$$\frac{dC_A}{dt} = \frac{\dot{V}}{V_R}(C_{A0} - C_A) - k_1(\mathcal{T})C_A - k_3(\mathcal{T})C_A^2 \qquad [\ 52\ ]$$

$$\frac{dC_B}{dt} = -\frac{\dot{V}}{V_R}C_B + k_1(\mathcal{T})C_A - k_2(\mathcal{T})C_B \qquad [\ 53\ ]$$

$$\frac{d\mathcal{T}}{dt} = \frac{\dot{V}}{V_R}(\mathcal{T}_0 - \mathcal{T}) - \frac{1}{\rho C_p}(k_1(\mathcal{T})C_A\Delta H_{rAB} + k_2(\mathcal{T})C_B\Delta H_{rBC} + k_3(\mathcal{T})C_A^2\Delta H_{rAD}) + \frac{UA}{\rho C_p V_R}(\mathcal{T}_C - \mathcal{T}) \qquad [\ 54\ ]$$

$$\frac{d\mathcal{T}_C}{dt} = \frac{1}{m_C C p_C}\left(\dot{Q}_C + UA(\mathcal{T} - \mathcal{T}_C)\right) \qquad [\ 55\ ]$$

$$k_i(\mathcal{T}) = k_{i0}\exp\left(\frac{E_i}{\mathcal{T}(°C) + 273.15}\right) \qquad [\ 56\ ]$$

The problem of interest in this work is the control of the concentration of compound B by adjusting only the normalized inlet flow rate $\left(\frac{\dot{V}}{V_R}\right)$. The motivation behind this selection is twofold. Firstly, only adjusting normalized flow rate is known to accentuate the challenging properties of the problem (Chen et al., 1995). Secondly, single control loop pairing allows all focus to be placed on the performance of the control laws studied.

Table 8 summarises the key variables of the Van de Vusse reaction scheme model, while Table 9 and Table 10 provide the parameter values used and the corresponding steady state for the state variables of the model, respectively. As shown in Table 8, $\mathcal{T}_0$ was provided as a measured variable, in contrast to the benchmark rules prescribed by Chen et al. (1995). This is purposefully done so that information regarding the DV is available during the simulations.

Table 8: Summary of the key variables in the Van de Vusse reaction scheme model

State variables	**Constant external variables (EV)**	**Disturbance variables (DV)**	**Controlled variables (CV)**	**Manipulated variables (MV)**	**Measured variables**
C_A	C_{A0}	$\mathcal{T}_0$	C_B	$\frac{\dot{V}}{V_R}$	C_B
C_B					$\mathcal{T}_0$
$\mathcal{T}$					
$\mathcal{T}_C$					
$Degrees\ of\ freedom = 7 - 7 = 0$					

Table 9: Nominal parameter list for the Van de Vusse reaction scheme model (Chen et al., 1995)

Parameter	**Parameter name**	**Numerical value**	**Unit**
k_{10}	Specific reaction rate for $A \rightarrow B$	1.287×10^{12}	$\frac{1}{h}$
k_{20}	Specific reaction rate for $B \rightarrow C$	1.287×10^{12}	$\frac{1}{h}$
k_{30}	Specific reaction rate for $2A \rightarrow D$	9.043×10^{9}	$\frac{1}{mol\ A.h}$
E_1	Activation energy for $A \rightarrow B$	$-9\ 758.3$	K

Parameter	Parameter name	Numerical value	Unit
E_2	Activation energy for $B \rightarrow C$	$-9\,758.3$	K
E_3	Activation energy for $2A \rightarrow D$	$-8\,560$	K
ΔH_{RAB}	Reaction enthalpy for $A \rightarrow B$	4.2	$\frac{kJ}{mol\ A}$
ΔH_{RBC}	Reaction enthalpy for $B \rightarrow C$	-11	$\frac{kJ}{mol\ B}$
ΔH_{RAD}	Reaction enthalpy for $2A \rightarrow D$	-41.85	$\frac{kJ}{mol\ A}$
ρ	Density of process fluid	0.9342	$\frac{kg}{L}$
C_p	Heat capacity of process fluid	3.01	$\frac{kJ}{kg.K}$
U	Cooling jacket heat transfer coefficient	4 032	$\frac{kJ}{h.m^2.K}$
A_R	Cooling jacket surface area	0.215	m^2
V_R	Volume of reactor	10	L
m_C	Mass of coolant	5	kg
Cp_C	Heat capacity of coolant	2	$\frac{kJ}{kg.K}$

Table 10: Steady state of the Van de Vusse reaction scheme model

State variable	Name of variable	Steady state	Unit
C_A	Concentration of compound A	2.14	$\frac{mol}{L}$
C_B	Concentration of compound B	1.09	$\frac{mol}{L}$
$\mathcal{T}$	Temperature of the process fluid	114.2	$°C$
$\mathcal{T}_C$	Temperature of the coolant	112.9	$°C$
$\frac{\dot{V}}{V_R}$	Inlet flow rate divided by reactor volume	14.19	$\frac{1}{h}$

4.2.2 System Inputs

Two schemes were used to change the inputs to the Van de Vusse reaction scheme model. In the first, the SPs were sampled according to the logic described in Section 4.1.2, with SP values of $0.95\frac{mol}{L}$ and $1.09\frac{mol}{L}$. The DVs were changed with normally distributed periods between step changes, and each new DV value was sampled from a normal distribution. The mean and standard deviation of the normal distribution used for DV sampling during each training episode are shown in Table 11. In Table 11, the last two rows show the mean and standard deviation used for the sampling of the time steps at which the DV changes occur. The unit "steps" in this table refers to a multiple of the process time associated with each of the MDP time steps, i.e. with each transition from T to $T+1$. In the second scheme, SPs, DVs, and the times of these step changes were sampled from uniform distributions on bounded intervals given in Section 5.3.3 during each episode.

Table 11: Number of MDP steps and parameters describing the normal distribution used for DV and time step sampling for the Van de Vusse reaction scheme model

Parameter	Numerical value
Number of steps	400
μ_{DV}	$2\ °C$
σ_{DV}	$3\ °C$
μ_T	$120\ steps$
σ_T	$0.5\ steps$

Instrument lag was simulated by letting $\mathcal{T}_0(T+1) = \mathcal{T}_0(T)$ and $SP(T+1) = SP(T)$, while the only state component for which information regarding $T+1$ was made available during an update at time step T is $E(T+1)$.

4.2.3 Classical Controller for the System

The reactor is required to operate close to the point of optimal yield of compound B, as described by Chen et al. (1995). At this operating point, classical control of the reactor through PI/PID control becomes infeasible. The reactor displays non-minimum phase dynamics, meaning that the CV (C_B) initially follows a trajectory in the opposite direction to its final steady state change. The controller's sense would therefore need to change in ways that aren't known beforehand. Intuitively, one might want to detune the controller by reducing the gain and increasing the sampling period between control actions for digital control in an attempt to prevent inducing an unstable control response. This is not feasible owing to the reaction scheme's stability properties and dynamics that change vastly depending on the SP and DV behaviours applied.

4.3 Case Study 3: Grinding Circuit

4.3.1 System Description and Model Summary

Semi-autogenous grinding mills (SAG mills) are often-used comminution units that are applied as part of a grinding circuit. The overall goal of such a circuit is the reduction of the particle size of incoming ore in accordance with design specifications of downstream classification and beneficiation equipment. The circuit must therefore allow liberation of the minerals of interest at sufficient capacity (Atkins et al., 1974). Changes in SP are typically of small magnitudes since the design specifications of downstream unit operations do not change frequently.

In practice, grinding circuit operation constitutes a significant portion of overall metal production costs (Rajamani and Herbst, 1991). The costs are driven to a large extent by the consumption of energy and steel grinding media in the mill (Wills and Finch, 2016). Rajamani and Herbst (1991) identify the important DVs for a grinding circuit as ore hardness variation, variations in feed particle size, and changes in the rate of the ore feed. Ore feed rate is typically manipulated and regulating the inlet water flow rate is not a major challenge.

Le Roux et al. (2013) validated the qualitative behaviour of a SAG mill grinding circuit model. A flow diagram for the process is shown in Figure 10. The detailed particle size distribution of the cyclone overflow (or any of the other streams) is not modelled. Rather, a coarse distribution of particle sizes is modelled in terms of five states – rocks (ore too large to leave the SAG mill), solids (particles that can leave the SAG mill), fines (in-specification particles), water, and steel balls (grinding medium). The solids are further categorized as coarse particles which are out of specification and fine particles that are in specification.

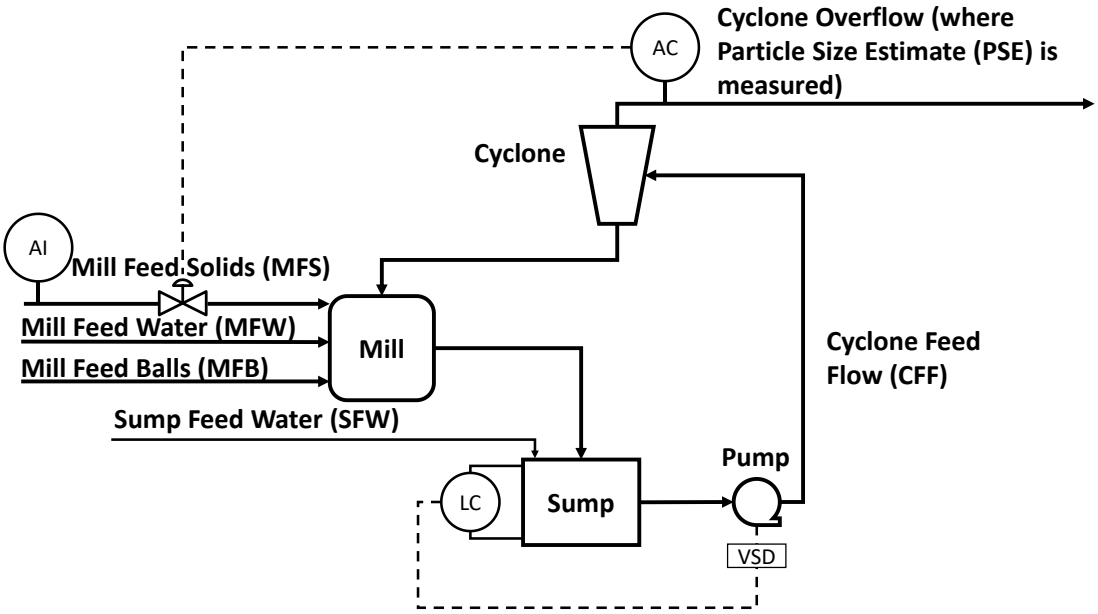


Figure 10: Flow diagram of the grinding circuit model used for Case Study 3, redrawn from Le Roux et al. (2013)

The grinding circuit model cannot be used as an accurate tool for process design or predictive maintenance. One of its main strengths lies in its suitability for the testing of control engineering concepts and the feasibility study of proposed controllers in the mineral processing industries.

The CV indicating whether particle size is reduced sufficiently is the particle size estimate (PSE) which represents the mass fraction of the solid material constituted of fines in the cyclone overflow. Symbols that are of particular interest in this study are the mass fraction of incoming ore that is categorized as rock (υ), the rock abrasion factor that is used to model ore hardness (ϕ_r), and the power consumption per tonne of fines produced (ϕ_f). In this study, particle size analysers are placed on the mill feed solids (MFS) and cyclone overflow streams. These instruments measure υ and PSE, respectively. A PI controller is used to control the level of slurry in the sump by adjusting the cyclone feed flow (CFF) which is the slurry stream pumped from the sump to the cyclone.

Figure 10 shows the control loop pairings selected with the symbol AC denoting a control calculation based on the PSE measurement, and the symbol LC indicating a level control calculation based on a level measurement. The symbol AI is added on the MFS stream to show that υ is available to be used in a control law calculation. The measurement taken in the MFS stream is purposefully assumed to provide υ rather than the fraction of material specifically above the desired particle size (this includes coarse material which is categorised as part of solids). This adds to the modelled uncertainty regarding how measurements of the input to the overall circuit relate to the PSE output. In Figure 10, level control incorporates a signal sent to a variable speed drive (VSD) of a positive displacement pump.

The convention used for the model subscripts is explained in Table 12, and the model equations obtained from Le Roux et al. (2013) are provided in Equations [57] through [102]. Table 13 provides a concise description of the model equations, and how they describe the modules in Figure 10 as denoted by the abbreviations in Table 12.

Table 12: Convention used for subscripts in the grinding circuit model (Le Roux et al., 2013)

Letter position in subscript	**Description**
First	$f = feeder; m = mill; s = sump; c = cyclone$
Second	$w = water; s = solids; c = coarse; f = fines; r = rocks; b = balls$
Third	$i = inflow; o = outflow;\ u = underflow$

Table 13: Grinding circuit model equations obtained from Le Roux et al. (2013)

Equation	Number	Description	Module (Table 12)
Equations for state variables			
$\frac{dX_{mw}}{dt} = V_{mwi} - V_{mwo} = (V_{fwo} + V_{cwu}) - V_{mwo}$	[57]	mill water mass balance	m
$\frac{dX_{ms}}{dt} = V_{msi} - V_{mso} + RC(t) = (V_{fso} + V_{csu}) - V_{mso} + RC$	[58]	mill solids mass balance	m
$\frac{dX_{mf}}{dt} = V_{mfi} - V_{mfo} + FP = (V_{ffo} + V_{cfu}) - V_{mfo} + FP$	[59]	mill fines mass balance	m
$\frac{dX_{mr}}{dt} = V_{mri} - RC = V_{fro} - RC$	[60]	mill rocks mass balance	m
$\frac{dX_{mb}}{dt} = V_{mbi} - BC = V_{fbo} - BC$	[61]	mill balls mass balance	m
$\frac{dX_{sw}}{dt} = V_{swi} - V_{swo} = V_{mwo} - V_{swo} + SFW$	[62]	sump water mass balance	s
$\frac{dX_{ss}}{dt} = V_{ssi} - V_{sso} = V_{mso} - V_{sso}$	[63]	sump solids mass balance	s
$\frac{dX_{sf}}{dt} = V_{sfi} - V_{sfo} = V_{mfo} - V_{sfo}$	[64]	sump fines mass balance	s
Additional equations for combined feeder and SAG mill model			
$V_{fwo} = MIW$	[65]	feeder outlet water flow rate	f
$V_{fso} = \frac{MFS}{D_S}(1 - \upsilon)$	[66]	feeder outlet solids flow rate	f

$V_{ffo} = \frac{MFS}{D_S}\alpha_f$	[67]	feeder outlet fines flow rate	f
$V_{fro} = \frac{MFS}{D_S}\upsilon$	[68]	feeder outlet rocks flow rate	f
$V_{fbo} = \frac{MFB}{D_B}$	[69]	feeder outlet balls flowrate	f
$\varphi = \left\{\max\left[0; 1 - \left(\left(\frac{1}{\varepsilon_{SV}}\right) - 1\right)\left(\frac{X_{ms}}{X_{mw}}\right)\right]\right\}^{0.5}$	[70]	rheology factor calculation	m
$V_{mwo} = V_V \varphi X_{mw}\left(\frac{X_{mw}}{X_{ms}+X_{mw}}\right)$	[71]	mill outlet water flow rate	m
$V_{mso} = V_V \varphi X_{mw}\left(\frac{X_{ms}}{X_{ms}+X_{mw}}\right)$	[72]	mill outlet solids flow rate	m
$V_{mfo} = V_V \varphi X_{mw}\left(\frac{X_{mf}}{X_{ms}+X_{mw}}\right)$	[73]	mill outlet fines flow rate	m
$LOAD = X_{mw} + X_{mr} + X_{ms} + X_{mb}$	[74]	mill volumetric loading	m
$Z_r = \left(\frac{\varphi}{\varphi_{P_{max}}}\right) - 1$	[75]	slurry rheology effect on mill power draw	m
$Z_\chi = \left(\frac{LOAD}{v_{mill} v_{P_{max}}}\right) - 1$	[76]	total charge effect on mill power draw	m
$P_{mill} = P_{max}\{1 - \delta_{PV} Z_\chi^2 - 2\chi_p \delta_{PV} \delta_{PS} Z_\chi Z_r - \delta_{PS} Z_r^2\}\alpha_{speed}^{\alpha_p}$	[77]	mill power draw	m
$RC = \frac{P_{mill}\varphi}{D_S \phi_r}\left(\frac{X_{mr}}{X_{mr}+X_{ms}}\right)$	[78]	rock consumption	m
$FP = \frac{P_{mill}}{D_S\left\{\phi_f\left[1+\alpha_{\phi_f}\left(\frac{LOAD}{v_{mill}} - v_{P_{max}}\right)\right]\right\}}$	[79]	fines production	m

$BC = \frac{P_{mill}\varphi}{\phi_b}\left(\frac{X_{mb}}{D_S(X_{mr}+X_{ms})+D_B X_{mb}}\right)$	[80]	ball consumption	m
Additional equations for sump			
$SVOL = X_{sw} + X_{ss}$	[81]	sump volumetric loading	s
$E(t)_{sump} = V_{SP} - V = (H_{SP} + pump\ inlet\ centre\ line\ position)(l)(b) - SVOL$	[82]	error signal for sump loading control	s
$CFF = CFF_0 + K_c\left(E(t)_{sump} + \frac{1}{T_I}\int E(t)_{sump}\,dt\right)$	[83]	PI control for CFF adjustment	s
$V_{swo} = CFF\left(\frac{X_{sw}}{SVOL}\right)$	[84]	sump water outlet flow rate	s
$V_{sso} = CFF\left(\frac{X_{ss}}{SVOL}\right)$	[85]	sump solids outlet flow rate	s
$V_{sfo} = CFF\left(\frac{X_{sf}}{SVOL}\right)$	[86]	sump fines outlet flow rate	s
Cyclone equations			
$V_{csi} = V_{sso}$	[87]	cyclone solids inlet flow rate	c
$V_{cfi} = V_{sfo}$	[88]	cyclone fines inlet flow rate	c
$V_{cwi} = V_{swo}$	[89]	cyclone water inlet flow rate	c
$V_{cci} = V_{csi} - V_{cfi}$	[90]	cyclone coarse material inlet flow rate	c
$F_i = \frac{V_{csi}}{CFF}$	[91]	cyclone solids fraction in feed	c

$P_i = \frac{V_{cfi}}{V_{csi}}$	[92]	cyclone fraction fines in feed solids	c
$V_{ccu} = V_{cci}\left(1 - C_1 \exp\left(\frac{-CFF}{\varepsilon_C}\right)\right)\left(1 - \left(\frac{F_i}{C_2}\right)^{C_3}\right)\left(1 - P_i^{C_4}\right)$	[93]	cyclone coarse material flow rate in the underflow	c
$F_u = 0.6 - (0.6 - F_i)\exp\left(-\frac{V_{ccu}}{(\alpha_{SU}\varepsilon_C)}\right)$	[94]	cyclone fraction of solids in the underflow volume	c
$V_{cwu} = \frac{V_{cwi}(V_{ccu} - F_u V_{ccu})}{(F_u V_{cwi} + F_u V_{cfi} - V_{cfi})}$	[95]	cyclone underflow water flow rate	c
$V_{cfu} = \frac{V_{cfi}(V_{ccu} - F_u V_{ccu})}{(F_u V_{cwi} + F_u V_{cfi} - V_{cfi})}$	[96]	cyclone underflow fines flow rate	c
$V_{csu} = V_{ccu} + V_{cfu}$	[97]	cyclone underflow solids flow rate	c
$V_{cfo} = V_{cfi} - V_{cfu}$	[98]	cyclone overflow fines flow rate	c
$V_{cco} = V_{cci} - V_{ccu}$	[99]	cyclone overflow coarse material flow rate	c
$V_{cso} = V_{csi} - V_{csu}$	[100]	cyclone overflow solids flow rate	c
$V_{cwo} = V_{cwi} - V_{cwu}$	[101]	cyclone overflow water flow rate	c
$PSE = \frac{V_{cfo}}{V_{cco} + V_{cfo}}$	[102]	particle size estimate	c

Table 14 summarises the key variables of the grinding circuit model with the control problem formulation used in this thesis. Table 15 and Table 16 provide the parameter lists for the combined feeder and mill module, and the cyclone module, respectively.

Table 14: Summary of the key variables in the grinding circuit model

State variables	**Constant external variables ($\boldsymbol{EV}$)**	**Changing external variable ($\boldsymbol{EV(t)}$)**	**Disturbance variable ($\boldsymbol{DV}$)**	**Controlled variables ($\boldsymbol{CV}$)**	**Manipulated variables ($\boldsymbol{MV}$)**	**Measured variables**
X_{mw}	MIW	MFB	υ	H_{sump}	CFF	υ
X_{ms}	SFW	MFS	ϕ_f	PSE	MFS	PSE
X_{mf}			ϕ_r			
X_{mr}						
X_{mb}						
X_{sw}						
X_{ss}						
X_{sf}						
$Degrees\ of\ freedom = 46 - 46 = 0$						

Table 15: Parameter list for combined feeder and mill module, obtained from Le Roux et al. (2013)

Parameter	**Parameter name**	**Numerical value**	**Unit**
α_f	Fines fraction in the ore	0.055	–
υ	Rock fraction in the ore	0.465	–
α_P	Fraction power reduction per fraction reduction from critical mill speed	1	–
α_{speed}	Fraction of the critical mill speed	0.712	–
α_{ϕ_f}	Fractional change in power per tonne of fines produced per change of fractional mill filling	0.01	–
δ_{P_s}	Power-change parameter related to the fraction of solids in the mill	0.5	–
δ_{P_v}	Power-change parameter related to the volume filled in the mill	0.5	–
D_B	Steel ball density	7.85	$\frac{t}{m^3}$
D_S	Feed ore density	3.2	$\frac{t}{m^3}$
ε_{SV}	Maximum volume fraction of slurry comprised of solid material for a stationary slurry	0.6	–

Parameter	Parameter name	Numerical value	Unit
ϕ_b	Abrasion factor of steel	90	$\frac{kWh}{t}$
ϕ_f	Power requirement per tonne of fines produced	29.6	$\frac{kWh}{t}$
ϕ_r	Rock abrasion factor	6.03	$\frac{kWh}{t}$
$\varphi_{P_{max}}$	Maximum mill power draw rheology factor	0.57	–
P_{max}	Mill motor power draw maximum	1 662	kW
v_{mill}	Volume of the mill	59.12	m^3
$v_{P_{max}}$	Fraction of mill volume filled for maximum power draw	0.34	–
V_V	Volumetric flow per driving force for flow (pressure applied to the slurry to discharge from the mill)	84	–
χ_P	Maximum power draw cross-term	0	–

Table 16: Parameter list for cyclone module, obtained from Le Roux et al. (2013)

Parameter	Parameter name	Numerical value	Unit
α_{SU}	Parameter pertaining to underflow solids fraction	0.87	–
C_1	Constant	0.6	–
C_2	Constant	0.7	–
C_3	Constant	4	–
C_4	Constant	4	–
ε_C	Parameter pertaining to the split of coarse material between the overflow and underflow streams	129	$\frac{m^3}{h}$

A noteworthy property of the grinding circuit model as described by Le Roux et al. (2013) is that it targets qualitative modelling accuracy with a concurrent compromise signified markedly by excessively large steel ball consumption rates. In addition, the model parameters fitted to plant survey data by Le Roux et al. (2013) work well near the steady state operating points that the authors considered. The empirical equations used to model steel ball consumption in the mill with their proposed continuous flow rates of mill feed balls (MFB) to the mill limits the numerically feasible modelled operating window significantly. The consequence of this is that cognisance must be shown of the operating window when working with the model.

Green and Perry (2008) do state that grinding circuits can become unstable by overloading the SAG mill, but since the continuous MFB stream behaviour proposed by Le Roux et al. (2013) is not aimed at being practically representative, a more realistic pulse function for the MFB stream is considered appropriate. Potentially increased PSE fluctuations would contribute to a greater challenge being posed to the control laws studied. In this thesis, the MFB stream of the grinding circuit model is modelled using $8\,\frac{t}{h}$ peaks in the intervals $(0;5), (20;25), (40;45), (60;65)$, and $(80;85)$. In between, the MFB stream was equal to $0\,\frac{t}{h}$.

The cyclone in Figure 10 has significantly faster dynamics than the rest of the grinding circuit. As a consequence of this, cyclones in grinding circuits are typically modelled using algebraic equations (Rajamani and Herbst, 1991; Le Roux et al., 2013). This results in a numerically challenging dependence of the mill input on the cyclone underflow stream – the input to the mill at the current time t is dependent on the underflow of the cyclone. The underflow of the cyclone is, in turn, dependent on the output of the SAG mill. To address this numerical problem, the grinding circuit model equations for the SAG mill and the sump are written as delay differential equations with constant lags.

The delay equations rely on the delayed variables being differentiated with respect to time in the model. Since the cyclone equations are algebraic, the delay equations need to be applied to the other units in the grinding circuit. The physical principle leveraged to this end is the incompressibility of liquid water. This gives rise to Equations [103] through [107] for the SAG mill. The same equations are applied to the sump, with the respective concentrations and flow rates changed to those of the sump. The lags used for the state variables and their initial steady state values are given in Table 17. The steady state inlet flow rates are given in Table 18. Calculations for the sump must use the delayed flow rates from the mill, while the mill equations rely on the outputs of the equations describing the cyclone underflow, which must be calculated using the delayed outputs from the sump.

$$C_{ms,delayed} = \frac{V_{mso,delayed}}{(V_{mso,delayed} + V_{mwo,delayed})} \quad [\,103\,]$$

$$C_{mf,delayed} = \frac{V_{mfo,delayed}}{(V_{mso,delayed} + V_{mwo,delayed})} \quad [\,104\,]$$

$$V_{mso,adjusted} = C_{ms,delayed}(V_{mso} + V_{mwo}) \quad [\,105\,]$$

$$V_{mfo,adjusted} = C_{mf,delayed}(V_{mso} + V_{mwo}) \quad [\,106\,]$$

$$V_{mwo,adjusted} = (V_{mso} + V_{mwo})\left(1 - C_{ms,delayed}\right) \quad [\,107\,]$$

Table 17: Initial steady state values and lags used when simulating the grinding circuit model

State variable	Variable name	Steady state (m^3)	Lag (h)
X_{mw}	mill water	4.789	30/3600
X_{ms}	mill solids	4.844	30/3600
X_{mf}	mill fines	1.002	30/3600
X_{mr}	mill rocks	1.797	30/3600
X_{mb}	mill balls	8.488	30/3600
X_{sw}	sump water	4.118	0.1
X_{ss}	sump solids	1.866	0.1
X_{sf}	sump fines	0.3864	0.1

Table 18: Inlet flow rates at the initial steady state of the grinding circuit model

Model input	Variable name	Steady state	Unit
MFS	mill feed solids	65.2	t/h
V_{fwo} (MFW)	mill feed water – the flow rate of water out of mill feeder	4.64	m^3/h
MFB	mill feed balls	5.69	t/h
SFW	sump feed water	140.5	m^3/h

The steady state coordinate of the grinding circuit model is independent of the delays used, as delays do not affect the steady state of a model. The adjusted flow rate calculations given by Equations [105] through [107] have a slight effect on the steady state values of the state variables. The steady state values reported in Table 17 were found by evaluating the model when all time derivatives are set equal to zero. This required the use of a steady state material balance for the sump, Equation [108]. A CFF value of 370.2 $\frac{m^3}{h}$ was used (Le Roux et al., 2013).

$$SFW = CFF - V_{swi} - V_{ssi} \qquad [\ 108\ ]$$

4.3.2 System Inputs

The DVs υ, ϕ_r, the SP for PSE, and the times for all these step changes during each episode were sampled from bounded uniform distributions. The bounds are provided in Section 5.3.5. The changes in ϕ_f were implemented as step changes with magnitudes and times corresponding to the input changes described by Le Roux et al. (2013).

4.3.3 Control Problem Development and Classical Controllers for the System

It is known from literature that sump control is critical, and that a PID controller can exert sufficient control (Atkins et al., 1974; Barker and Hulbert, 1983; Conradie and Aldrich, 2001). The PSE measurement

provided by a particle size analyser was used as a representative characteristic of the entire particle size distribution of the cyclone overflow. Therefore, it was assumed that the fraction of ore in one particle size range is related to the fraction of ore in another particle size range in a constant manner (Atkins et al., 1974).

The relatively faster dynamics of the cyclone prevent high frequency upsets in CFF from being properly attenuated by a control system. A sufficiently consistent CFF ensures smooth flow and prevents spigot overloading. In Case Study 3, flow measurements are assumed to be available immediately – the time constant of the measurement equipment will be sufficiently small to prevent significant change in CFF between two measurements.

The sump is an integrator system and should act as a buffer between the mill and the cyclone (Marlin, 2000). Typically, averaging level control is employed since the most important control objective of the sump is to prevent product quality fluctuations caused by inconsistent CFF.

Based on typical qualitative dynamic characteristics of the grinding circuit shown in Figure 10 and by following similar work in literature, MFS was paired with PSE, and MFW was paired with sump level (Atkins et al., 1974; Barker and Hulbert, 1983; Rajamani and Herbst, 1991; Conradie and Aldrich, 2001). RL-based control was implemented to control the PSE-MFS control loop, while PI control was used for the sump. This allowed applying RL-based control to a SISO control problem while still limiting transmission interaction between the control loops used to control the grinding circuit. A SISO application of RL-based control was studied for the grinding circuit model to maintain a stationary RL environment – the true value associated with each action remains constant (Sutton and Barto, 2018).

Based on the qualitative traits of the system which aided in selecting control loop pairing, it was expected that the specific tuning constants used for sump control do not affect the PSE measurements obtained significantly. This expectation was validated in Figure 11 where the PSE profiles (expressed as percentages) obtained by simulating the grinding circuit model in MATLAB for different sets of PI controller tuning constants for the sump is shown. The CV is the height of slurry in the sump, H_{sump}, and the MV is CFF. Figure 11 also illustrates that the use of sump volume in the sump control calculations, Equations [82] and [83], does not influence the results significantly. The changes in model inputs during the generation of Figure 11 were implemented as step changes and had the same magnitudes as those reported by Le Roux et al. (2013) with constant ϕ_f.

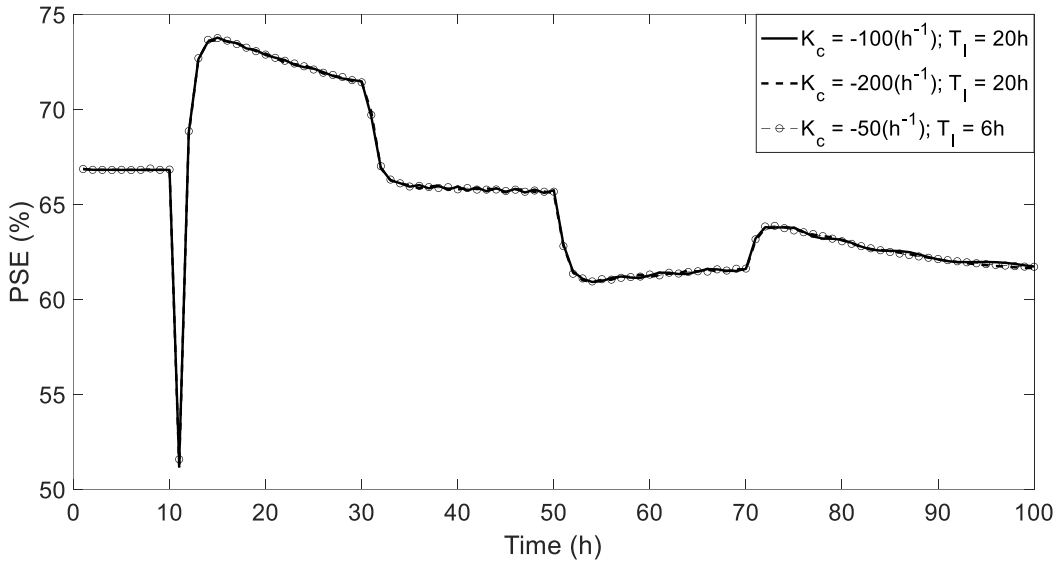


Figure 11: PSE profiles obtained from the grinding circuit model for various PI controller tuning parameters and constant ϕ_f

For comparison to RL-based control, a PID controller was also applied to the PSE-MFS control loop. A step change in the MFS stream from $65.2\frac{t}{h}$ to $46.7\frac{t}{h}$ was applied to the grinding circuit model while all other model inputs and parameters were maintained at their initial steady state values. The Laplace domain approximation to the grinding circuit model's response is shown in Equation [109]. It was observed that τ_1 and τ_2 in the denominator have a significant impact on both the initial slope and the shape of the response, while τ_3 in the numerator mostly has an effect on the fine-tuning of the transfer function model's output. The numerical values selected for τ_1, τ_2, and τ_3 were 3, 10, and 21, respectively. Figure 12 shows the grinding circuit model output (solid line) and the Laplace domain model output (dashed line).

$$G_p(s) = \frac{\left(-0.42\frac{\%\,PSE}{\frac{t}{h}MFS}\right)(21s+1)}{(3s+1)(10s+1)} \qquad [\ 109\ ]$$

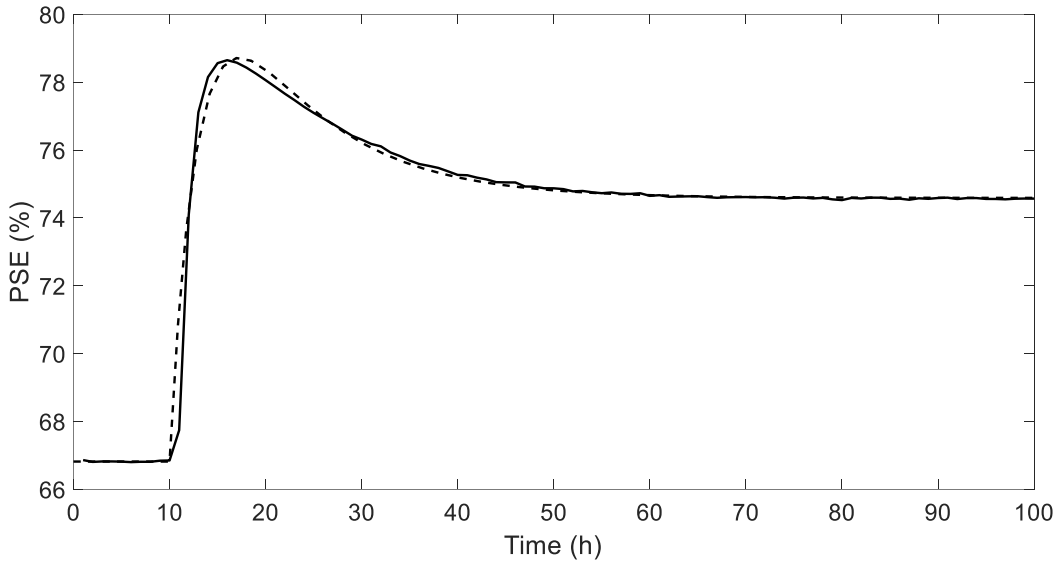


Figure 12: PSE responses to MFS step change from $65.2\frac{t}{h}$ to $46.7\frac{t}{h}$, the solid line is the output from the non-linear grinding circuit model, while the dashed line is the Laplace domain model output

Two key considerations when selecting a PID tuning method were the availability of a handle that allows the author to target robust performance and that suitable assumptions are made in the tuning method with regard to DV and SP changes. The recommended PID control law form of Equation [45] was used. The Direct Synthesis for Disturbances (DS-d) tuning method of Chen and Seborg (2002) was applied. This tuning method provides a closed-loop time constant τ_c that targets SP tracking and DV attenuation at the cost of system robustness as it is decreased (Chen and Seborg, 2002).

In this thesis, $\tau_c = 12.5\ h$ was used as a reasonable closed-loop time constant that ensures that sufficient robust performance is achieved to enable justifiable comparability between control laws without resulting in excessive oscillatory behaviour in the simulations. Fairly large controller gain magnitude and some oscillatory behaviours were allowed – these do not compromise the validity of the comparisons made. Reset windup was prevented by not integrating the error in Equation [45] when the MFS final element was saturated. Saturation was used to ensure that the limits on the MV that ensure that the operational data used for agent training lies within the process operating window (Section 5.3.5) applies to the PID controller as well.

The assumption is made that the disturbance and process transfer functions are equal, $G_d(s) = G_p(s)$, and the desired response to disturbances is used in the analytical design equation that Chen and Seborg (2002) derived for tuning. In the grinding circuit model, υ and ϕ_r are exogenous inputs that enter the grinding circuit as properties of the MFS stream and would therefore have qualitatively similar disturbance transfer functions to the process transfer function $G_p(s)$. Approximate models are sufficient near the point of linearization (Marlin, 2000; Skogestad and Poslethwaite, 2005). The tuning constants determined by

applying the DS-d tuning method were $K_c = -15\ \frac{\left(\frac{t}{h}\right)}{\%\ PSE}$, $\tau_I = 19.6\ h$, and $\tau_d = 5\ h$ (Chen and Seborg, 2002). Subsequently, the gain was detuned to $K_c = -10\ \frac{\left(\frac{t}{h}\right)}{\%\ PSE}$.

Two properties of the RL agents studied ensure that it is reasonable to compare the PID and RL-based controllers. Firstly, from the RL agent's perspective, training is aimed solely at maximising cumulative reward irrespective of whether SP or DV changes occur. Secondly, by discretizing the state-action space (SARSA) and the action space (One-Step Actor-Critic) coarsely in this study, the agent is approaching the control problem from a low-dimensional perspective which contributes to an inherent extent of robustness at the expense of MV adjustments which are typically excessive.

For each execution of the update rule, the agent has $v(T+1) = v(T)$ and $SP(T+1) = SP(T)$, while the only state component for which information regarding $T+1$ is made available during an update at time step T is $E(T+1)$. The PID control law was implemented with discrete time steps of $0.01\ h$ which allows for the use of a continuous control law since the dynamic responses of the grinding circuit take much longer, Figure 12. No instrument lag was incorporated for the PID controller as the goal of simulating instrument lag was to contribute to representing intrinsic uncertainty sensibly when training an RL agent for the control of the PSE-MFS control loop.

Future information would not be available in practice, and therefore the RL agent's current MDP time step T would typically trail behind the time of the process being controlled with one sampling period ΔT in the time domain. The greatest source of partial observability is instrumentation limitation.

Valve stiction modelling was incorporated to evaluate how an RL agent would respond to changing closed – loop dynamics that are usually challenging for PI/PID controllers. The two parameter, data-driven model of Shoukat Choudhury et al. (2005) was implemented. The model has a parameter for deadband plus stickband (S) and a parameter for slip jump (J). Both of these parameters are expressed as percentages of the available MV range. The model uses binary decision making rather than numerically challenging differential equations, has been formulated in a framework where stiction has been clearly defined, and the model's ability to approximate first-principles modelling results has been validated by Shoukat Choudhury et al. (2005). Further, the stiction model does not rely on physical valve characteristics.

Code that may be used to validate the stiction model implementation used in this thesis through simulation of the undershoot case of stiction (S > J) is given in Appendix C. For the stiction model, the MV range for application to the PSE-MFS control loop was $0\frac{t}{h}$ to $100\frac{t}{h}$. This prevented unintended valve saturation. Parameter values (J and S) were maintained constant when training further the initial RL agent policies to test for stiction compensation capability.

The integral of the absolute error (IAE), the integrated product of time and absolute error (ITAE), and the total MV variation until the end of a predefined number of time steps (TV, Equation [110]) were used as quantitative controller performance measures (Marlin, 2000; Chen and Seborg, 2002).

$$TV = \sum_{i=1}^{end} |MV[(i+1)\Delta T] - MV[i\Delta T]| \qquad [\ 110\ ]$$

4.4 Numerical Implementation

All simulations were implemented using MATLAB R2020a. The built-in numerical integrator "ode45" was used to solve the self-regulatory water tank model (Section 4.1), while "ode23s" was used to solve the Van de Vusse reaction scheme model equations (Section 4.2). The grinding circuit model equations (Section 4.3) were solved by using the built-in numerical integrator "dde23" with constant lags and a history function specifying the initial steady state values of the state variables.

CHAPTER 5

RL METHODOLOGY

An RL-based control scheme reminiscent of the work of Syafiie et al. (2008), Brujeni et al. (2010), and Ramanathan et al. (2018) was used to conduct the feasibility study of RL-based process control. This control scheme is described in Section 5.1. Section 5.2 describes the schedule used for the probability of taking a random action, ε, and the allocation of training schemes and parallel computing resources.

Tabular SARSA (Section 2.8.1) was used as the RL agent to study the mechanisms by which the studied control scheme needs to be tuned, zero steady-state offset achieved (Sections 5.3.1 and 5.3.2), and whether controller performance is sensitive to how finely the state-action space is discretized (Section 5.3.3). The effects of the hyperparameters α and γ of SARSA have on learning were investigated using the Van de Vusse reaction scheme model (Section 5.3.4). Q-learning was used to investigate how initial estimates of the parameters of the RL representation used for SARSA (the entries of the tabular approximate action-value function $\tilde{Q}_\pi(\boldsymbol{S}, \boldsymbol{A})$) may be obtained in an off-policy manner (Section 5.4).

The implementation of a SARSA agent and a One-Step Actor-Critic agent (Section 2.9.5) to the grinding circuit model (Section 4.3) are described in Sections 5.3.5 and 5.5, respectively. The RL methodology facilitates critical analysis of the current feasibility of elementary model-free RL-based control in the chemical and mineral processing industries.

5.1 The Control Scheme Used for Study

Figure 13 illustrates the proposed control scheme used for study. The observed state is defined as having components $[E(T)\ DV(T)\ SP(T)]^T$ (Francis and Wonham, 1976; Bertsekas and Tsitsiklis, 1996). Based on the work of Syafiie et al. (2008), the author recommends the use of 10 intervals of equal width for each state component if the RL representation used involves the discretization of the state space, as a first approximation. For both SARSA and Actor-Critic agents, the action available at each time step T, $\boldsymbol{A}$, comprised a discrete set of final element settings. Each action therefore had only one component and may be denoted by a scalar A.

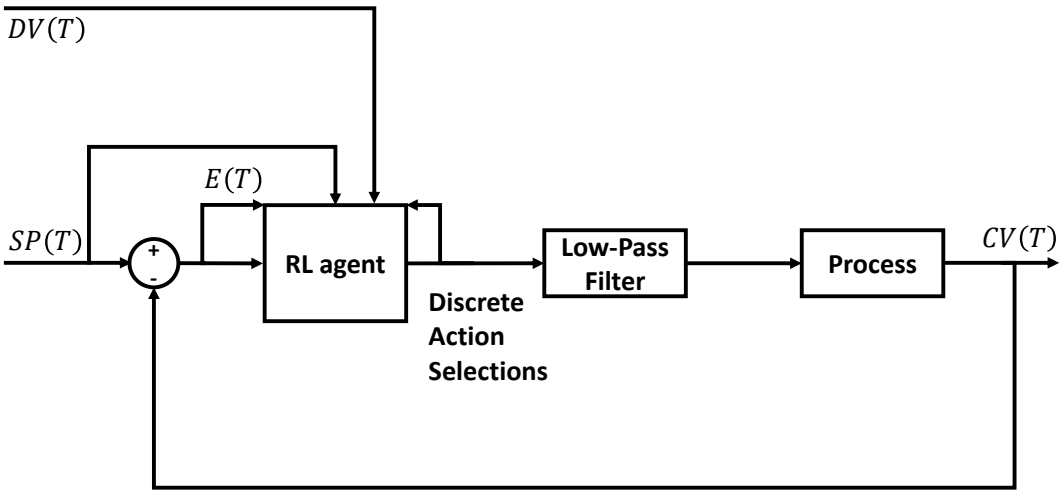


Figure 13: Time domain block diagram of the control scheme studied (RL environment is everything excluding the RL agent)

Feedback must be provided to the agent through the incorporation of the error signal $E(T)$ in the reward function. A binary reward function with a reward width $\pm\beta$ was used. This means that, if $|E(T)| < \beta$, $R = 1$, otherwise $R = 0$. Care was taken to discretize the error signal $E(T)$ symmetrically when defining the state space $\mathcal{S}$ for RL-based control. Half of the reward width, β, may be set equal either to the control tolerance or measurement instrumentation noise, provided that sufficient range is present in the actions available to the agent – this is illustrated in Section 6.2.1.

Intermediate MV adjustments were applied to the process by sending the agent's discrete action selections to a low-pass filter, Equation [111] (Syafiie et al., 2008). This simplifies the action space for the RL agent as it only needs to deal with discrete action selections. The frequencies with which it makes its action selections result in a larger set of filtered actions being applied to the RL environment. This introduces a second tuning constant in addition to β – the filter time constant τ_f. The filter also enables the agent to maintain a constant MV intermediate to the discrete action selections so that the desired value of the CV may be realised.

$$A_{filtered}(T) = \exp\left(\frac{-1}{\tau_f}\right) A_{filtered}(T-1) + \left(1 - \exp\left(\frac{-1}{\tau_f}\right)\right) A(T) \qquad [\ 111\ ]$$

To select a reasonable value for τ_f, final element adjustment must be taken into account – it provides a handle for trading off MV adjustments and CV variations. In addition, it must be considered whether a large number of discrete action selections and a smaller value for τ_f is feasible within the number of operational hours available for training. The type of RL agent is also an important consideration, as will

be clarified in Section 6.7. From an RL perspective, it does not make sense to train an agent off-policy on a different RL environment than the one on which it will be deployed for online training, and therefore the value of τ_f should be fixed before training to prevent possibly degrading agent performance unintentionally. If a control law is required that does not attempt to maintain tightly the CV at SP, increasing β (and possibly τ_f) is suitable. The sampling period ΔT used must enable the agent to easily discern the causal relationship between the MV and the CV in the CV-MV pairing during the process's transient response to a change in input. This aids in simplifying the control problem for the RL agent.

It is important to realise that applying an RL agent to a control problem requires an interface between the RL agent and the RL environment. For a tabular SARSA agent, Section 2.8.1, this means that each state component must be mapped to a coded state component. Therefore, based on predefined ranges, the measurement obtained for each component is mapped to integer values used to represent states in the tabular RL representation. Each action component available to the RL agent is also denoted by an integer number, which is subsequently mapped to the corresponding MV value sent to the low-pass filter. The interface required for the One-Step Actor-Critic agent (Section 2.9.5) was described in Section 2.9.7.

In the RL-based control simulations conducted, all CV measurements were provided to the controllers as noise-free data. Since the state-action space was discretized coarsely when applying SARSA, the signal to noise ratio will be sufficiently large when mapping inputs and outputs relevant to the process to discretized states and actions to prevent significant changes in the coded states and actions corresponding to the plant measurements. In contrast, PID control and Actor-Critic control operate in the continuously modelled state space $\mathcal{S}$, and therefore noise would affect these controllers. Introducing bias during the comparison of different controllers by including instrumentation noise was not desired.

5.2 Exploration Schedule and Allocation of Training Schemes, Serial and Parallel Computation

The simulation work described in Sections 5.3.1 and 5.3.2 involved the application of SARSA agents with an ε-greedy exploration strategy where a large probability of taking a random action, ε, was defined initially. The initial value of ε during each episode was ε_0. At each time step T a coefficient just below unity ($\mathfrak{C}$) is multiplied with the previous value of ε. Therefore, the probability of a random action is expressed as shown in Equation [112] for each episode. This promoted state excitation and therefore facilitated achieving sufficient coverage of the state-action space during the serial computations performed. In Section 5.3.2, ε was only decayed at the end of an episode.

$$\varepsilon(T) = \mathfrak{C}^{T-1}\varepsilon_0 \qquad [\ 112\ ]$$

No terminal state was used since fixed episode length implies that each episode ends with a probability of one. As an example, a training episode of 100 MDP time steps involves 99 transitions from T to $T+1$, since the first training step has $T = 1$.

The first training scheme of Section 4.2.2 applies to Section 5.3.2 and the second training scheme of Section 4.2.2 applies to Sections 5.3.3 through 5.3.5. The other training details were provided in Sections 4.1.2 and 4.3.2. Serial computation is applicable to Sections 5.3.1, 5.3.2, Q-learning applied to Case Study 1 (water tank model), and Section 5.5. The other sections of the RL methodology used parallel computation, Section 5.6, with $\varepsilon = 0.1$. All process models were used in their non-linear forms, as given in Chapter 4.

5.3 SARSA Agent

5.3.1 Value-Based Control Mechanics and Tuning Simplification – Case Study 1 (Water Tank Model)

To study the water tank model, Section 4.1, the observed state S only contained the discretized error signal $E(T)$. This has the benefits of validating that an extent of robust decision making may be obtained even though the state space is incompletely defined, being easily relatable to other elementary RL-based control applications discussed in Chapter 3, and allows readily observing the agent's learning curve. When more than one action is greedy at the state for which an action must be selected at the current time step T, tie-breaking is performed by selecting a random action from the available discrete action selections.

The use of RL-based control inevitably increases the number of parameters that must be set externally by the designer. This only makes tuning challenging for the parameters that need to be tailored manually to the control loop under consideration. These parameters that need to be considered may be categorised as tuning parameters (β and τ_f in the control scheme studied), and RL hyperparameters (described in Chapter 2).

It was therefore necessary to consider the effect that the tuning constants β and τ_f have on qualitative control behaviours obtained for the water tank problem. To do so, the water tank model was studied at different levels of discretization of the state-action space, as is described in Section 6.2.1.

A SARSA agent and a binary reward structure with $\beta = 0.5\,m$ about SP were used. During training, a total of 3 000 episodes were used with 1 200 steps per episode. The agent was applied to learn a satisfactory policy π by training with inputs to the water tank model as described in Section 4.1.2. Because the number of episodes is much smaller than the total number of steps, resetting the starting state at the start of each episode is not practically unreasonable. A summary of the hyperparameters and tuning parameters used are given in Table 19.

Table 19: All hyperparameter and tuning parameter values used to study the water tank control problem

Parameter	Numerical Value	Description
$\mathfrak{C}$	0.99	a constant used to decay the probability of taking a random action
ε_0	0.8	initial probability of taking a random action
ε during testing of policy	10^{-4}	probability of taking a random action used during the testing of the RL agent after initial training
α	0.7	step size hyperparameter
γ	0.7	discount factor
β	0.5 m or 0.2 m	reward function width specification (half of total width)
τ_f	20 min	time constant used for the low-pass filter of Equation [111]
ΔT	1 min	sampling period

5.3.2 RL Benchmarking Simulations

The work of Chen et al. (1995) was consulted so that the Van de Vusse reaction scheme model (Section 4.2) can, to an extent, enable benchmarking in this thesis. The results generated for the Van de Vusse reaction scheme, accompanied by qualitative comparison to Chen et al. (1995), are sufficient. It is important to note that the incorporation of the inlet temperature $\mathcal{T}_0$ in the state of the RL-based controller by coarsely discretizing it as a state component, accompanied by small changes in SP for servo problem simulation, and a different set of DV values from those used by Chen et al. (1995) prevent direct quantitative comparison to literature. "Relaxing" the rules of the benchmark problem in this way to suit the intended application is not detrimental to the project as a whole. The coarse discretization of DV and SP used in this section are revised when investigating discretization coarseness and during Case Study 3 (the grinding circuit model).

The sampling period ΔT was selected to ensure that the agent may only apply an action after approximately 40 seconds of simulated time passed since its previous action during its interaction with the RL environment. This ensured that the agent was exposed to the process dynamics resulting from its discrete action selections and that the same assumption was made as Chen et al. (1995) with regard to time steps between MV adjustments. Presumably, their assumption was aimed at incorporating the influence of measurement delay in their MPC study. In on-policy RL-based control, a time delay in sensor measurement would not only affect the control output, but also the frequency with which operational data may be provided to the agent during training, and therefore operational data requirement. The SARSA update rule, Equation [20], was executed for each transition of T to $T+1$ at the same sampling period with which the actions are applied to the RL environment (also refer to the end of Section 2.8.1).

The hyperparameters, SP sampling vector, and discretization settings used for each component of $\boldsymbol{S}$ are given in Table 20. The unit "steps" in this table refers to the process time associated with each of the MDP time steps, i.e. with each transition from T to $T+1$, since a numerator of one is used in the exponentiated factor of Equation [111]. Two instances of ε_0 were stored in a vector, one for each coded state component interval of the DV discretization. At the end of each episode, Equation [112] was executed for the corresponding entry of DV discretization. This prevented ε from too quickly reaching an unreasonably small value. Instrument lag was simulated as described in Section 4.2.2.

Table 20: Hyperparameters, tuning parameters, SP sampling vector, and discretization settings for SARSA agent training during the benchmarking simulations

Training setting	**Value**	**Description**
Number of episodes	2 000	-
Number of steps per episode	400	-
$timeIfSeconds$	2 500 $seconds$	simulated time per episode (in seconds)
$desiredTrueTime$	40 $seconds$	desired simulated time between adjustments to the action applied to the RL environment (in seconds)
$\mathfrak{C}$	0.99	a constant used to decay the probability of taking a random action
ε_0	$[0.8,0.8]^T$	initial probability of taking a random action
ε during testing of policy	10^{-4}	probability of taking a random action used during the testing of the RL agent after initial training
α	0.5	step size hyperparameter
γ	0.99	discount factor
β	0.02 $\frac{mol}{L}$	reward function width specification (half of total width)
τ_f	20 $steps$	time constant used for the low-pass filter of Equation [111]
ΔT	43.75 seconds	sampling period
$E(T)$ discretization	9 intervals of equal width from $-0.05\frac{mol}{L}$ to $+0.05\ \frac{mol}{L}$, with two coarser state components from $-0.6\ \frac{mol}{L}$ to $-0.05\ \frac{mol}{L}$ and from $+0.05\ \frac{mol}{L}$ to $+0.6\ \frac{mol}{L}$	-

Training setting	Value	Description
$DV(T)$	Two intervals: one from $50°C$ to $107.5°C$, and one from $107.5°C$ to $150°C$	-
A discretization	Discretized from $3\frac{1}{h}$ to $35\frac{1}{h}$ in increments of 10 %	-

5.3.3 Investigating State-Action Space Discretization

To find out how discretization affects performance if hyperparameter and tuning parameter values remain fixed, qualitative factor screening was used. Case Study 2 (the Van de Vusse reaction scheme) was studied and a 2^4 factorial experiment was designed by following the single replicate methodology of Daniel (1959) as described by Montgomery (2013). The response variable was the IAE point estimator generated by calculating the arithmetic mean IAE of 100 generated control problems. In this investigation, each pseudorandom control problem is defined as having 10 SP settings and 10 DV settings which are sampled from uniform distributions on bounded intervals. The study of a SISO control loop pairing ensures that all interactions are a result of how the state-action space is discretized.

The factors used are given in Table 21. Boundary states refer to very coarse intervals of the state component's discretization that prevents failure of RL environment state mapping to the discretized state-action space defined for the RL representation. The two coarsely defined intervals for error signal $E(T)$ in Table 20 were examples of boundary states. Instrument lag was simulated as described in Section 4.2.2.

Table 21: Definitions of design factors used for the investigation of state-action space discretization

Factor name	Description of design factor	Range of design factor (excluding boundary states)	Corresponding component of classical feedback control loop (Marlin, 2000)
A	C_B error signal discretization, $E(T)$	$\left[-0.05\frac{mol}{L}, 0.05\frac{mol}{L}\right]$	Feedback control error $E(t)$
B	Inlet temperature ($\mathcal{T}_0$) discretization	$[100°C, 115°C]$	DV
C	SP discretization for C_B	$\left[0.95\frac{mol}{L}, 1.11\frac{mol}{L}\right]$	SP
D	Discrete action selections available to the agent	$[3\ h^{-1}, 35\ h^{-1}]$	MV

All hyperparameters and other training settings remained constant throughout. These are summarized in Table 22. A single replicate of the factorial experiment was comprised of two batches. Each batch consists of 11 500 episodes with 400 steps per episode. A total of 23 000 training episodes were therefore used for each of the 16 levels of the 2^4 factorial experiment.

Table 22: Training settings used for the investigation of state-action space discretization

Training setting	Value
Number of episodes per batch	11500
Number of steps per episode	400
Number of batches per level	2
α	0.5
β	$0.02\frac{mol}{L}$
ε	0.1
γ	0.99
τ_f	$20\ steps$

During training at each level of the factor screening comparable performance needed to be achieved. Comparable performance does not imply complete coverage of the discretized state-action space, but rather that the main and interaction effects may be observed. The effects of limited coverage are lumped as part of the discretization effect. This is because the net effect of increasing the state-action space discretization is of interest when providing the agent with sufficient data to achieve a reasonable coverage.

To decide on the discretization settings to use, the control study of Syafiie et al. (2008) was consulted. To ensure that the observations do not result purely from training process noise, the high and low factor settings are selected far apart. Specifically, the ability of the agent to effectively use its discretization in the dimension at the low level is removed. This is achieved by defining the low and high levels of a state component's discretization as one state interval with two boundary states and 10 state intervals with two coarse boundary states, respectively. The state intervals are defined within the ranges given in Table 21, and each state component's intervals have uniform width. Importantly, a state component at the low setting (coarse discretization) still contains three coarsely defined intervals. The low and high settings for the number of discrete actions available to the agent were five and 25, respectively.

Since there is significant uncertainty associated with the experimental design, a significance level of 0.10 was chosen for statistical analysis. The uncertainty includes the different and unknown coverages instantiated for the different experimental runs.

In addition to consideration of the memory requirement for the variables generated while executing the implementation code, learning curves were generated using the proposed algorithm given in Appendix D. The parameters of the algorithm are the fixed episode length, Ep, and the window size W. The window size W refers to a predefined number of training episodes for which the rewards obtained are summed independently. The rewards obtained in successive windows gives an indication of how the agent's learning process progresses. These were set to $Ep = 20$ and $W = 500$ for 11 500 episodes. All

hyperparameters and tuning parameters (except $\varepsilon = 0.001$) are kept at the same settings as during agent training (Table 22). The two extremes of discretization are considered again, with curves generated for the $\{L, L, L, L\}$ and $\{H, H, H, H\}$ cases. The learning curves generated are not monotonic owing to the stochastic manner in which inputs to the training process were changed at each time step.

In the $\{L, L, L, L\}$ case, the agent reached its first learning curve peak after approximately 3 windows (1500 episodes). In comparison to this, approximately 11 windows (5500 episodes) were required to reach a similar position in the $\{H, H, H, H\}$ case. It is argued that it is likely that comparable performance is achieved if the number of episodes used in one replicate is more than $\frac{11}{3}$ times the number of episodes required to reach the first learning curve peak in the $\{H, H, H, H\}$ case. This rule of thumb represents an extreme case comparison for the number of coordinates within the action-value hypervolume. To validate that sufficient operational data was provided to the agent to observe the effects of discretization, the results must reflect that a finely discretized state-action space improves the achievable control performance.

5.3.4 Value-Based Hyperparameter Characterization

If a tabular RL representation is used for value-based methods, the convergence properties associated with α are independent of the instance of discretization applied. This may be shown by applying principles of stochastic approximation (Theodoridis, 2020).

Equation [113] for constant α was derived in Section 2.8.1. It illustrates that the SARSA update rule is a weighted average of previous knowledge, $M_{T-1}(\boldsymbol{S}, \boldsymbol{A})$, and the newest information which is represented by the most recent Bellman target t. If $\alpha = 0$, there is no update to the entries of the RL representation, if $\alpha = 1$, previous knowledge is discarded.

$$M_T(\boldsymbol{S}, \boldsymbol{A}) = \alpha t + (1 - \alpha) M_{T-1}(\boldsymbol{S}, \boldsymbol{A}) \qquad [\ 113\ ]$$

After conducting preliminary experiments with the Van de Vusse reaction scheme model (Case Study 2) that involved varying γ for $\alpha = 1$, and subsequently varying the value of α for the selected value of γ, nominal values of $\gamma = 0.99$ and $\alpha = 0.7$ were selected for Case Study 3 (the grinding circuit model). The numerical results for these experiments are presented in Section 6.3.3.

5.3.5 Tabular SARSA Applied to the PSE-MFS Control Loop (Grinding Circuit Model)

The SARSA agents were trained using $\beta = 0.02$, where PSE is expressed as a mass fraction. The components of the observed state provided to the agent, $\boldsymbol{S}$, were $E(T)$, v, and the SP for PSE. The available MFS values were not included as a state component to maintain a clear separation between cause and effect, simulate a constraint in the available instrumentation, and to prevent an excessive increase in the number of entries in the action-value hypervolume. The DVs ϕ_r and ϕ_f were purposefully not included as state components. Rock hardness (modelled using ϕ_r) is not assumed to be practically measurable. The power consumption per tonne of fines produced ϕ_f needs to be changed according to the predefined

schedule given by Le Roux et al. (2013) to maintain the qualitative integrity of the grinding circuit model's behaviour. Including ϕ_f as a state component could therefore bias the RL agent's learned behaviour. Instrument lag was simulated as explained in Section 4.3.3. Initially, a SARSA agent was provided with 20 000 episodes with 100 MDP time steps per episode for training. Each transition from T to $T+1$, (ΔT), was set independently to 1 h. This selection of ΔT is sufficiently small in comparison to the process time constant to expose the agent to the dynamics of the process.

The MFS range was limited so that operational data used for training corresponded to realisable state-action coordinates in the steady state of the grinding circuit model. A representative control problem was studied (after training in the case of the SARSA controller) which required the tracking of the SP of the PSE in the presence of υ, ϕ_r, and ϕ_f changes. During this test υ, ϕ_r, and ϕ_f were changed as shown in Figure 14. After initial SARSA agent training, an additional 176 episodes (approximately 2 years of on-policy training data) were used to further train the SARSA agent in the presence of stiction. All control performance measures include the initial plant-agent mismatch caused by the initial grinding circuit model's steady state value of $MFS = 65.2\frac{t}{h}$ being outside of the MV range provided, and the mismatch between the learned behaviour of the agent with tabular RL representation and the steady-state operating condition of the plant.

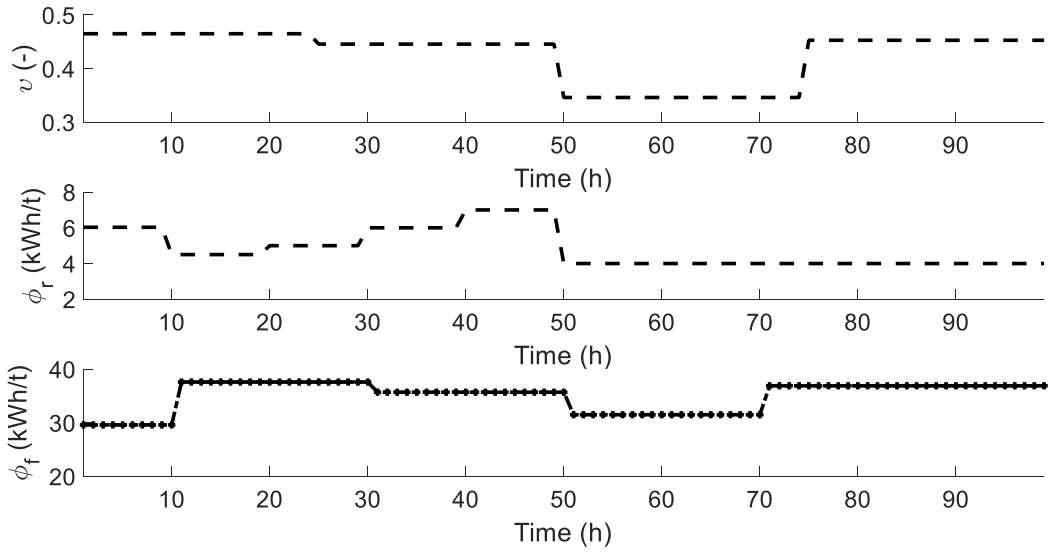


Figure 14: Changes in the DVs υ, ϕ_r, and ϕ_f for the representative control problem

A constant input period of 10 h was recommended by Le Roux et al. (2013) to allow model transients to subside for their proposed changes in ϕ_f. The selection of τ_f is best understood when considering simulation results and detailed discussion is therefore deferred to Section 6.7.

Table 23 and Table 24 provide a summary of the hyperparameters, tuning constants, and state component bounds used. In Table 24, each component of $\boldsymbol{S}$ is discretized with arbitrarily coarse boundary states located at both sides of a more finely discretized region. Symmetric error signal $E(T)$ discretization *must* be used. During training 10 SPs and 10 DVs were sampled for each episode (Section 4.3.2). Table 25 provides the sampling ranges used. Actions were discretized from $33\frac{t}{h}$ to $60\frac{t}{h}$ in increments of 20%.

Table 23: Hyperparameter and tuning settings for comparison of PID and SARSA controllers

Training setting	**Value**
ε during training	0.1
ε during testing of policy	10^{-4}
α	0.7
γ	0.99
PID gain (detuned)	$-10\ (t\ /\ h)/(\%\ PSE)$
PID integral time	$19.6\ h$
PID derivative time	$4.97\ h$
time steps for PID execution	$0.01\ h$

Table 24: Discretization settings of discretized state components

Training setting	**Value**
$E(T)$	4 intervals of equal width from -0.1 to $+0.1$ with two coarser state components from -1 to -0.1 and from $+0.1$ to $+1$
v	20 intervals of equal width from 0.315 to 0.565
$PSE\ SP$	8 intervals from 0.596 to 0.66, and 1 interval from 0.58 to 0.596

Table 25: Ranges for SP and DV sampling during SARSA agent training

SP* or *DV	**Lower bound**	**Upper bound**
PSE SP	$0.58\ (-)$	$0.66\ (-)$
v	$0.315\ (-)$	$0.565\ (-)$
ϕ_r	$5\ (kWh/t)$	$7\ (kWh/t)$

5.4 Q-Learning Agent

Two cases were considered for off-policy training of a tabular Q-learning agent, namely obtaining initial policy estimates for the level control of the water tank model (Case Study 1) and for the PSE-MFS loop of the grinding circuit model (Case Study 3).

In the simulation work conducted, the agent was connected in parallel to the classical controller (the behavioural policy π_b). In reality, historical plant data would need to be pre-processed and used during training. A sampling period ΔT of one minute was applied when training a Q-learning agent on the water tank model, and the integral mode of the PI controller was updated as shown in Equation [114] (after initialization at $T = 1\ min$) (Gilat and Subramaniam, 2014).

$$integral\ mode \leftarrow integral\ mode + \left[E(T-1)(\Delta T) + 0.5\big(E(T) - E(T-1)\big)(\Delta T)\right] \quad [\ 114\ ]$$

In all other cases where integrals had to be approximated, the differential with respect to time was included as a state variable sent to the built-in numerical integrator used.

5.5 One-Step Actor-Critic Agent (Grinding Circuit Model)

Bertsekas and Tsitsiklis (1996) describe the use of a regular grid for basis function placement. Shipman and Coetzee (2019) illustrate the scaling of components comprising the action space to be in the range of $[-1,1]$ to prevent incorporating unintended distorting of the basis functions used. Both of these concepts were incorporated in the One-Step Actor-Critic implementation of this thesis.

The state components obtained from the RL environment at each instance of T were scaled to a range of $[-1,1]$, while the action selections required the scaling of the number of discrete actions available to the same range. The bounds of Table 24 and Table 25 were used. Instrument lag was simulated as explained in Section 4.3.3. The other training specifications (excluding hyperparameter and RBF variance selection) were identical to those used for the training of SARSA agents for the PSE-MFS control loop (Section 5.3.5).

Step sizes of $\alpha_{\boldsymbol{w}} = 0.7$ and $\alpha_{\boldsymbol{\theta}} = 0.05$ were used. The latter was determined through trial and error, while $\alpha_{\boldsymbol{w}}$ was assumed based on the results presented in Section 6.3.3 despite the use of function approximation. While the use of Robbins-Monro convergence conditions for α would guarantee convergence, parameter convergence would likely be slowed down excessively (Robbins and Monro, 1951; Dvoretzky, 1956; Sutton and Barto, 2018; Theodoridis, 2020). In contrast to initialization using zeros for all table entries in SARSA, random numbers sampled from the uniform distribution over $[0, 0.01]$ were used to initialize both $\boldsymbol{w}$ and $\boldsymbol{\theta}$ for the One-Step Actor-Critic algorithm. This was merely done to have some small initial degree of differentiation between the individual actions' selection probabilities, but is a fairly trivial aspect.

The manual selection of RBF variances (σ^2 in Equation [42]) was based on the link between interpolation and the use of parametric models in machine learning. Hastie et al. (2020) highlight that attention has been drawn to interpolators in the machine learning literature. This may mainly be attributed to observations that state-of-the-art neural networks adjust their parameters to optimise a scalar performance objective and may be considered to have attributes similar to interpolators. By studying the work of Bishop (2006) and Sutton and Barto (2018), it may readily be observed that neural networks are non-linear parametric models

that build on concepts developed for the linear parametric basis function modelling principles applied to the actor and critic RL representations in this thesis.

It is therefore assumed that the qualitative effect of RBF shape in a generic interpolation problem are, to an extent, applicable to the One-Step Actor-Critic algorithm. A minimum error is often reached at some point before excessive RBF width is achieved (Mongillo, 2011).

Fasshauer (2007) explains that an excessively large value of σ^2 causes the rows and columns of an interpolation matrix (contains the basis functions that need to be scaled using a weight vector) to become more alike which raises the condition number of the interpolation matrix. This is not a concern in RL. This is because no system of linear equations is solved using an interpolation matrix. Stochastically ascending (Policy Gradient and Actor-Critic) or descending (Value-based) methods can only be incapable of capturing the contours of the optimal policy π^* or the optimal value function $Q_\pi^*(\boldsymbol{S}, A)$ if the RBFs are defined with excessively large perceptive fields.

Different methods have been proposed in literature for the selection of uniform σ^2 for RBF models. Such selection algorithms and correlations often require knowledge of the number and distribution of data points and leverages cross validation to assess the performance of interpolants (Rippa, 1999; Fasshauer, 2007). In RL, the direct use of quantitative selection methods are not feasible because the operational training data cannot be visualized, are non-stationary, and depend on the RL problem formulation and control design (Sutton and Barto, 2018).

The variances σ^2 of both the actor and the critic basis functions were assumed to be uniform and equal to 0.1. This value of σ^2 was considered reasonable on a scale of $[-1,1]$ for each state component and the actions, and Figure 3 of Chapter 2 was specifically created with $\sigma^2 = 0.1$ to aid in conceptualising this selection. Only approximate instances of $\boldsymbol{w}$ and $\boldsymbol{\theta}$ need to be learned. To validate that sensible learning behaviour was achieved, the progression of control performance measures as more training episodes are provided to the RL agent was used as a type of "cross validation" for the numerical experiments conducted. This progression is considered for a fixed control problem after training using pseudorandom inputs to the plant.

To select an action at each time step T, a pseudorandom number between zero and unity was sampled from a uniform distribution. The sampled number was then compared to each element of the probability mass function $\pi(A|\boldsymbol{S}, \boldsymbol{\theta})$. An action was considered likely if the sampled number is lower than the probability mass associated with that action for the current instances of $\boldsymbol{S}$ and $\boldsymbol{\theta}$, with the most likely action selected at T. If none or more than one discrete selection of A is likely at T, tie-breaking is implemented by selecting a random action from the discrete set of available actions. A total of 20 000 episodes with 100 steps per episode were used.

The basis function placement method may be approached through the application of a numerical pattern that ensures that the regular grid starts in each direction at -1 and ends at $+1$. For the actor, five basis

functions were placed in each of the dimensions of the state space, as well as for the one-dimensional action space. This resulted in 625 entries for $\boldsymbol{\theta}$, each entry corresponding to a basis function placed in the region of the state-action space in which operational data is known to be generated. The critic had 4 basis functions in each dimension of the state space, resulting in 64 basis functions for the critic.

5.6 Numerical Implementation

Results that required parallel code execution (see Section 5.2 for allocation) were generated by training the RL agents in parallel using the Stellenbosch University High Performance Computing cluster: http://www.sun.ac.za/hpc. The training of these value-based RL agents was performed using many independent training episodes executed in parallel. The approximation $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ was obtained by summing across the parallel episodes.

The action corresponding to the maximum action-value at the current state is all that must be known to obtain the resulting policy π for an ε-greedy exploration strategy. The pseudorandom DV and SP changes during training ensured that sufficient state excitation occurred to allow an appropriate approximation $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ to be obtained. This should be contrasted to the One-Step Actor-Critic algorithm. The decision making process depends on the relative magnitudes of the actor's parameters which are contained in the vector $\boldsymbol{\theta}$. Each RBF has a perceptive field that causes a great dependence on the history of updates to the parameter vectors. Without modification of the algorithm to accommodate centralization of the parallel agent training instances, the One-Step Actor-Critic algorithm must be executed using serial computation.

Wall time is the actual time that passes during the execution of code. Speed-up is defined as the wall time of a reference number of cores divided by the wall time for running a fixed and representative problem on more than one core Lin and Snyder (2008). The resultant scaling behaviour for execution on a single computational node of the Stellenbosch University High Performance Computing cluster is shown in Figure 15. From Figure 15, 16 MATLAB workers were selected for parallel computation.

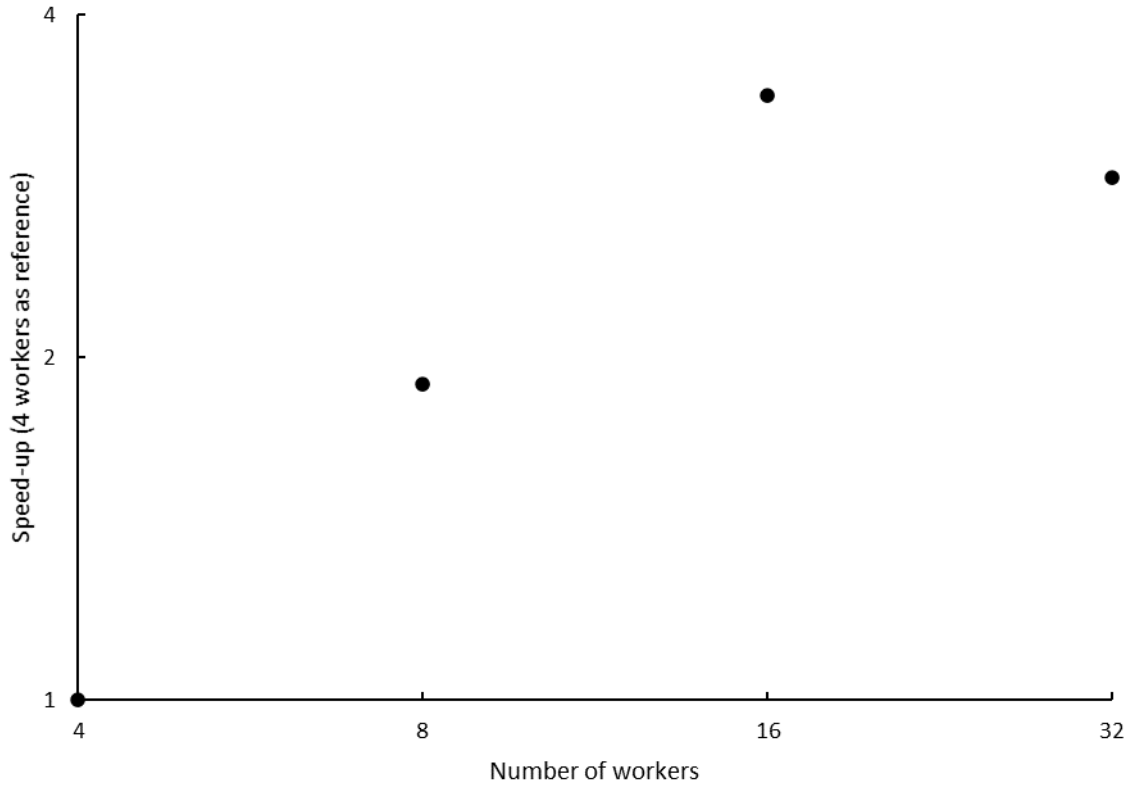


Figure 15: Speed-up versus number of MATLAB workers for the scaling test performed on the Stellenbosch University High Performance Computing cluster: http://www.sun.ac.za/hpc

5.7 Feasibility Criteria Used to Evaluate Elementary RL-Based Process Control

Recall from Section 5.1 that the control problem to be solved by the RL agent is simplified by providing the it with a discrete set of actions and then filtering these discrete action selections through the use of a low-pass filter. From Section 2.11, the optimal policy π^* is unique for a fixed process design and set of process parameters. By filtering the action selections in the control scheme of this study, a different optimal policy is sought by the RL agent. Since the agent does not operate directly in the state-action space of the process being controlled, this policy will be suboptimal to the policy targeted by an RL agent directly operating in the state-action space.

Feasibility was evaluated for this simplified problem since a number of the same strengths and weaknesses of the studied RL-based controllers may be assumed applicable to the more complex RL-based controller problem approached by state-of-the-art methodologies. Hence, the simplified control problem provides a

reference point for the evaluation of feasibility as it portrays an extreme of operation in terms of RL environment simplicity.

The following criteria were used to evaluate the feasibilities of SARSA, Q-learning, and One-Step Actor-Critic RL agents in this study and combines aspects of classical control with criteria from RL (Marlin, 2000; Skogestad and Poslethwaite, 2005; Sutton and Barto, 2018):

1. Memory usage and time for code execution
2. Control performance
3. Ease of RL agent tuning
4. Safety of exploration
5. Whether controller operation is interpretable to a human
6. Process modelling requirement

CHAPTER 6

RESULTS AND DISCUSSION

6.1 General Considerations

In the space of RL-based process control using synthetic data, it is important to conceptually validate the consistency of results with theory from classical control and RL. Secondly, results of the simulation work conducted may be compared qualitatively to results of related work reported in literature. Quantitative comparisons are limited to controllers implemented in this thesis. This is because some benchmark rules for the Van de Vusse reaction scheme (Case Study 2) are relaxed (Section 5.3.2), and the coarse state-action space discretization studied targets small SP changes (Section 6.2.1 and Section 6.3.1). Thirdly, reasonable performance for an elementary control problem (self-regulatory water tank (Case Study 1)) must be obtained since Chapter 2 and Chapter 3 illustrate that this is possible.

6.2 Case Study 1 – Self-Regulatory Water Tank

6.2.1 The Mechanics of Value-Based Process Control and Tuning Simplification

Table 26 provides the discretization specifications for the error signal $E(T)$ which comprised the agent's state, and the discrete actions (values of the fraction valve opening x) available for runs with $\beta = 0.5\ m$. Equation [115] depicts the binary reward function used for training. The second term in Equation [115] is required as no negative rewards were provided to the RL agents and the sign operator comprising the first term provides a value of $+1$ if $|E(T)| < \beta$ and -1 if $|E(T)| > \beta$. The selection of $\beta = 0.5\ m$ is very large in comparison to the SP liquid levels ($1\ m$, $1.4\ m$, and $2\ m$) considered in the simulations.

Table 26: Discretization settings for $\beta = 0.5\ m$ runs

Case	***S* discretization**	***A* discretization**
Low discretization state space, Low discretization action space ($Ld_S Ld_A$)	2 intervals of equal width from $-2.14\ m$ to $+2.14\ m$ with two boundary states	3 actions, namely 3.69×10^{-3}, 0.37, and 0.738
Low discretization state space, Fine discretization action space ($Ld_S Hd_A$)	2 intervals of equal width from $-2.14\ m$ to $+2.14\ m$ with two boundary states	10 evenly spaced actions from 3.69×10^{-3} to 0.738
Fine discretization state space, Fine discretization action space ($Hd_S Hd_A$)	10 intervals of equal width from $-2.14\ m$ to $+2.14\ m$ with two boundary states	10 evenly spaced actions from 3.69×10^{-3} to 0.738
Fine discretization state space, Low discretization action space ($Hd_S Ld_A$)	10 intervals of equal width from $-2.14\ m$ to $+2.14\ m$ with two boundary states	3 actions, namely 3.69×10^{-3}, 0.37, and 0.738

Case	S discretization	A discretization
$Asymmetric$	13 intervals of equal width from $-7\ m$ to $+15\ m$	10 evenly spaced actions from 3.69×10^{-3} to 0.738

$$R(T) = sign(\beta - |E(T)|) + 1(|E(T)| > \beta) \quad [115]$$

Figure 16 displays the learning curves of the agents trained in accordance with Table 26. Instead of counting all the rewards obtained by the agent in successive instances of a fixed window size of episodes, the total rewards obtained at the corresponding episode is plotted against the episode number. The action-value tables generated by the $Ld_S Ld_A$ and $Hd_S Hd_A$ runs of Table 26 are given in Figure 17 and Figure 18, respectively. Figure 19 and Figure 20 relate qualitative height trajectories instantiated in the water tank model by applying a step change in F_{in} from the training value of $190\frac{L}{min}$ to $200\frac{L}{min}$ at the start of the test run of the trained SARSA agent. During the test run, the RL agent's update rule was executed for each time step, as was done for all test runs reported in this chapter. In Figure 20, SP is not reached since the number of discrete states available to the agent are sufficient to allow it to realise that the level of liquid in the tank need only be within $\pm\beta$ of the SP height. Since $\beta = 0.5\ m$ in Figure 20, the height remains within the range $[0.9\ m, 1.9\ m]$ for the SP of $1.4\ m$.

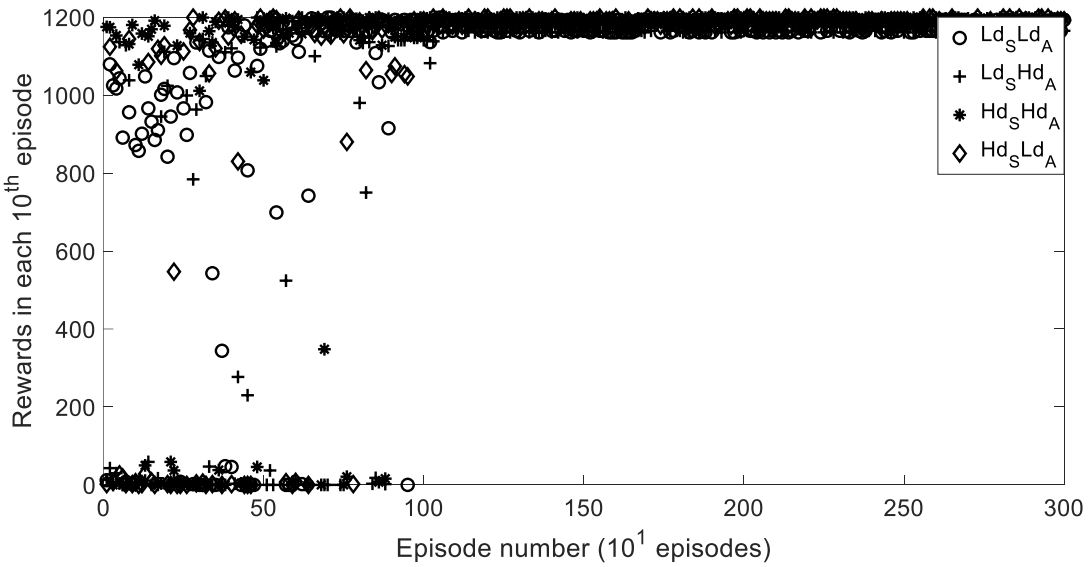


Figure 16: Learning curves for concept validation runs, discretization cases correspond to Table 26

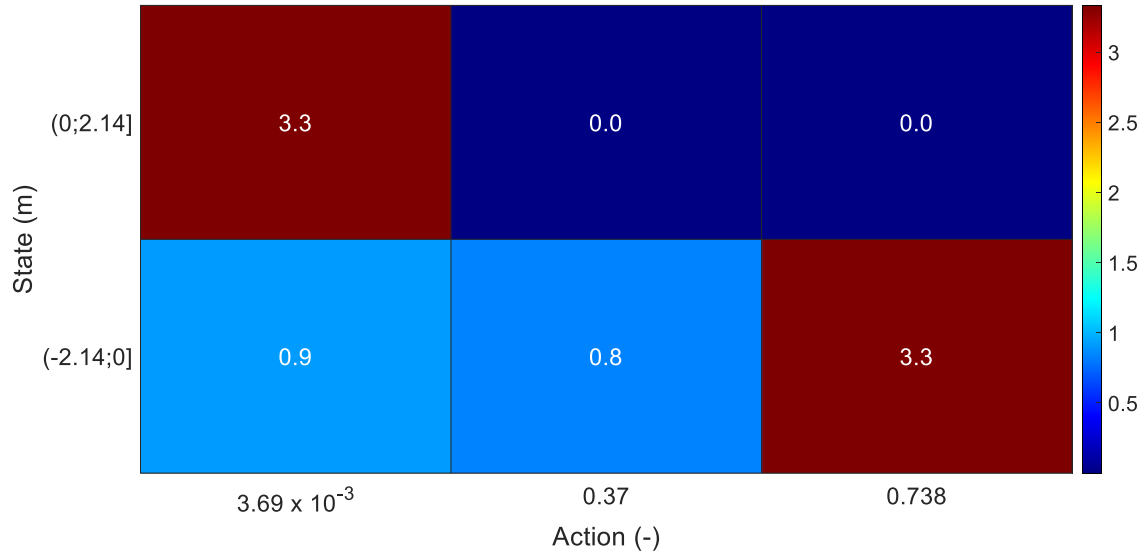


Figure 17: Action-value table after training for $Ld_S Ld_A$ case of Table 26 (units consistent with RL environment)

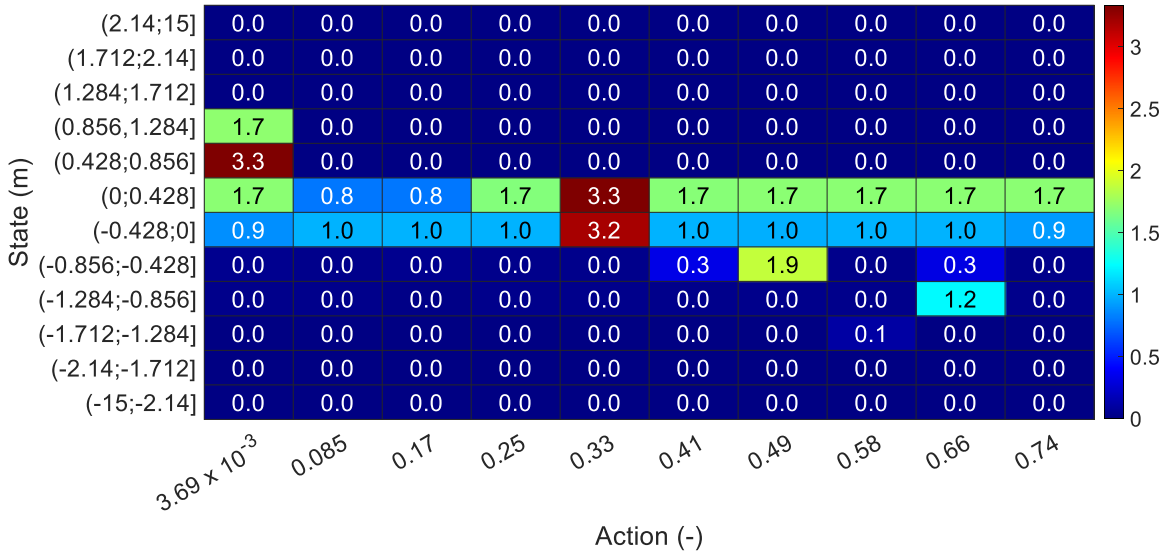


Figure 18: Action-value table after training for $Hd_S Hd_A$ case of Table 26 – the top and bottom rows are boundary states which are specifically chosen to capture outliers during training (units consistent with RL environment)

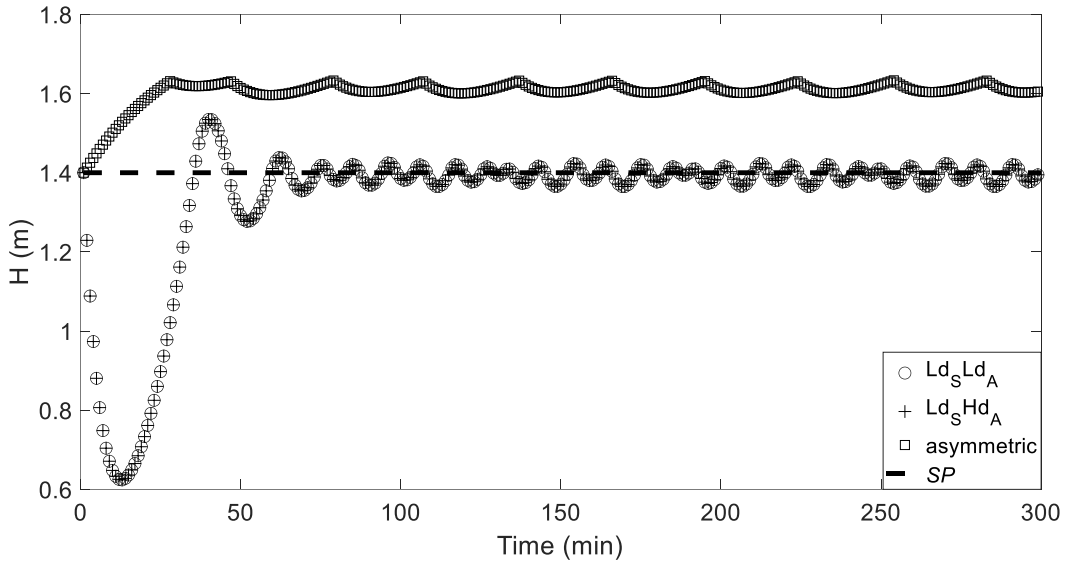


Figure 19: Height trajectories for trained agents with Ld_S state discretization of Table 26 ($\varepsilon = 10^{-4}$; $SP = 1.4\ m$; $DV = 200\,\frac{L}{min}$)

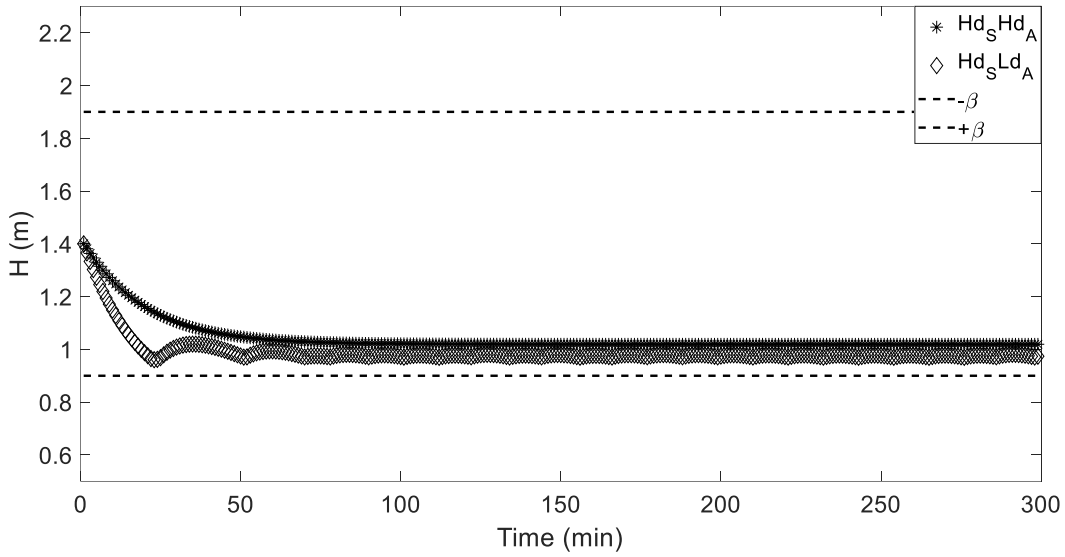


Figure 20: Height trajectories for trained agents with Hd_S state discretization of Table 26 ($\varepsilon = 10^{-4}$; $SP = 1.4\ m$; $DV = 200\,\frac{L}{min}$)

The learning curves of Figure 16 may readily be interpreted. The $Ld_S Ld_A$ and $Ld_S Hd_A$ cases take the longest to achieve a consistently high density of data points near the upper limit of $+1\,200$ rewards per episode (equal to the number of time steps per episode used during training). This is because the excursions in CV resulting from exploration are typically larger for the coarsely discretized error signal $E(T)$ – the agent has to train until the policy reflects that the use of two extremes of action selection are required to prevent the CV from moving outside the desired bounds. The $Hd_S Ld_A$ and $Hd_S Hd_A$ cases achieve high

reward densities close to $+1\ 200$ much quicker as the agent is provided with many state transitions that provide a positive reward for the large value of β.

In the limit of extremely coarse discretization ($Ld_S Ld_A$), the agent's policy π simplifies to bang-bang control logic. This is evidenced in Figure 17 which shows that, of the two discrete states visited during training, the negative state component (lower row with non-zero entries) has a maximum action-value in the rightmost column. This means that the agent opens the valve a lot (large value for x in Equation [49]). Conversely, when a positive error signal $E(T)$ is encountered (meaning that the liquid level is below SP), the agent closes the valve by selecting the lowest available value of x. After applying the action selections filtered through the use of Equation [111] to the self-regulatory water tank model, low-amplitude oscillations in the CV about the SP are caused, as shown in Figure 19.

Such an extremely coarse discretization is not sensible practically. This is because the agent loses its ability to adapt effectively to changes in the RL environment and low-amplitude CV oscillations cannot be avoided. The bang-bang control logic is the optimal policy for this level of discretization irrespective of whether control loop dynamics are altered. This stems from the fact that action selections at the extremes of the action space's range have the largest magnitude effect on the CV in the steady state.

Despite the practical limitation mentioned above, an important theoretical observation may be made by comparing the qualitative results of the $Ld_S Ld_A$, $Ld_S Hd_A$, and asymmetric cases in Figure 19. Even though the agent is provided with the same range of actions to apply to the RL environment, the trained policy fails to ensure that the CV oscillates about the SP in the asymmetric case. The agent's inability to enable zero steady state offset in Figure 19 could only stem from the asymmetric discretization of $E(T)$. This implies that the entries of the action-value function were biased statistically in the sense described by Devore (2017) by the asymmetric discretization of the state space.

In Figure 19, the qualitative results for the $Hd_S Hd_A$ and $Hd_S Ld_A$ cases indicate that the CV is successfully maintained within the $\pm 0.5\ m$ band around the SP of $1.4\ m$. Figure 18 shows that this is achieved without bang-bang control logic – the maximum action-values at the discretized states are distributed across the available columns of the table. Such a policy could potentially be of practical significance for averaging level control for a self-regulatory inventory system.

Table 27 provides the details for runs conducted with $\beta = 0.2\ m$ and involve a much finer discretization of $E(T)$. Since a key idea in the studied control scheme is to leverage the low-pass filter's effect to allow for a coarse discretization of the action space, a maximum of 10 discrete action selections are provided to the agent in Case Study 1 for both sets of runs. Figure 21 displays the learning curves for the runs of Table 27.

Table 27: Discretization settings for $\beta = 0.2\ m$ runs

Case	***S* discretization**	***A* discretization**
Low discretization state space, Fine discretization action space ($Ld_S Hd_A$)	10 intervals of equal width from $-2.14\ m$ to $+2.14\ m$ with two boundary states	10 evenly spaced actions from 3.69×10^{-3} to 0.738
Fine discretization state space, Fine discretization action space ($Hd_S Hd_A$)	100 intervals of equal width from $-2.14\ m$ to $+2.14\ m$ with two boundary states	10 evenly spaced actions from 3.69×10^{-3} to 0.738

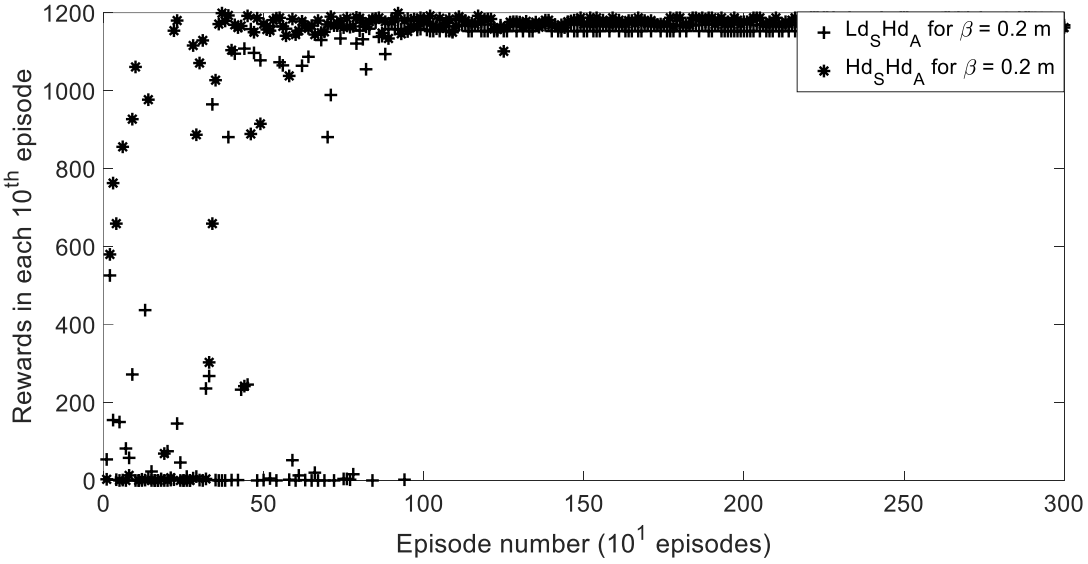


Figure 21: Learning curves for the discretization cases given in Table 27, generated in the same manner as Figure 16

A similar learning curve trend is observed in Figure 21 where the $Hd_S Hd_A$ case of Table 27 displays a much larger initial learning rate (steeper gradient of the learning curve), and a quicker achievement of the plateau of the learning curve. This illustrates that, when allowing for a fairly coarse action discretization, the number of episodes required to achieve CV values within the bounds specified by β is decreased by using a finer discretization of $E(T)$. It should be noted that this observation is made for a value of 2β equal to 40 % of the range of SP values used during agent training. Most often, the value of 2β would constitute a much smaller fraction of the SP range under consideration.

For the smaller reward width associated with $\beta = 0.2\ m$, the action-value table of the $L_S H_A$ case of Table 27 also results in a bang-bang control logic irrespective of more states being experienced by the agent than was the case in Figure 17. This shows that, for $\beta = 0.2\ m$, the discretization of $E(T)$ is too coarse to leverage the effect of the low-pass filter properly. For a large value of β, the threshold of $E(T)$ discretization fineness where the control scheme will start benefiting from the low-pass filter without resorting to bang-bang control will be lower, as evidenced by the results of the runs of Table 26.

Upon increasing the fineness of the discretization as per Table 27 ($Hd_S Hd_A$ case), three regions may be identified in the action-value table. A central region exists where the approximated action-values are relatively high and bands are formed on either side of this central region where there are relatively large action-values which are distributed sparsely in the table. The remainder of the action-value table is the third region and only has zero-valued entries (these entries were also initialized with zeros).

Figure 22 shows four cross sections of the action-value table obtained for the $Hd_S Hd_A$ case of Table 27. The state-action coordinates of the innermost band of the action-value table have the largest number of visits during training once the first positive rewards were obtained within the $\pm\beta$ range. This is shown by the action-values across the available actions being the largest for state 50 and state 53. The agent's more pronounced differences in action-values across the available actions for state 47 and state 56 indicate that the discounted rewards propagate from the centre of the discretized state-action space to peripheral state-action coordinates, thereby informing the agent regarding what corrective action would be beneficial should it encounter a coordinate slightly outside the central region of frequent visits.

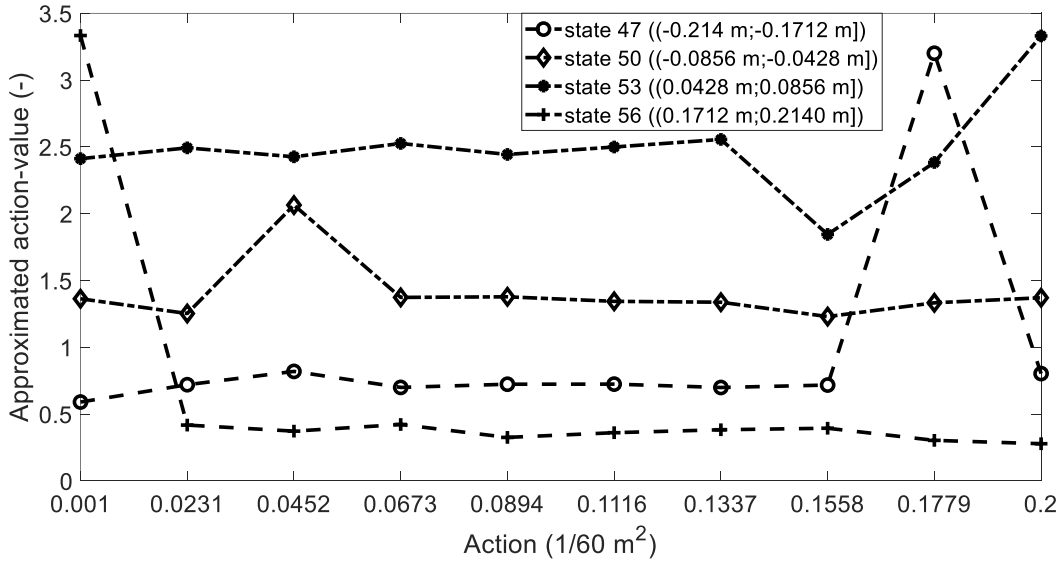


Figure 22: Cross sections of the action-value table for the $Hd_S Hd_A$ case of Table 27 (units consistent with RL environment)

When viewed together with the relatively steep initial gradient of the $Hd_S Hd_A$ entry of Figure 21 and the quick achievement of the maximum positive reward per episode, it may be inferred that $E(T)$ values in the remainder of the action-value table were not encountered often. This illustrates that the RL agent's control behaviour for a discretized state-action space will be most satisfactory for small magnitude SP changes if the state-action space is discretized coarsely – the central state-action coordinates will be the quickest to converge to their optimal values. Therefore, the shape of the optimal action-value function $\tilde{Q}_\pi(\boldsymbol{S}, A)$ will be learned the quickest for states within the region where $|E(T)| < \beta$.

The results of this section illustrate that the proposed control scheme studied (Section 5.1) enables the achievement of sensible simulated non-linear control for the feasibility study of an RL-based controller applied to a self-regulatory SISO control problem. This is consistent with the findings of Syafiie et al. (2008), Brujeni et al. (2010), and Ramanathan et al. (2018) in their studies of value-based RL controllers for SISO control systems (Chapter 3).

Since the same maximum rewards per episode were achieved in all the runs discussed in this section (Figure 16 and Figure 21), and positive rewards were only provided to the agent if CV was within the desired region about SP (Equation [115]), it may be inferred that maintaining CV sufficiently close to SP is not very sensitive to the selection of β. The value of β may therefore be selected according to a fixed rule, and may be set equal to measurement noise or control tolerance. In contrast, τ_f will have a very significant influence on controller performance, as it influences the frequencies with which final element adjustments may be applied to the plant.

6.2.2 Q-Learning

To investigate the behaviour of Q-learning as a potential off-policy method for developing a first approximation policy for a SARSA agent, the self-regulatory water tank model was controlled using the PI controller described in Section 4.1.3 as behavioural policy π_b. The controller's integral time was de-tuned in an attempt to promote increased coverage of the state-action space. Decreasing the magnitude of the controller gain would have had a larger impact in this regard, but it was assumed that this typically cannot be justified for industrial application.

During training, it was assumed that the water tank's inlet flowrate varies according to two scenarios, as shown in Equations [116] and [117]. Equation [116] represents an almost symmetric DV range about the initial steady-state conditions of $H = 1.5\,m$, $F_{in} = 190\,\frac{L}{min}$ and $x = 0.259$, while Equation [117] places significantly more weight on large DV values. At the start of each episode, a DV value from the relevant range was sampled for use in the training scheme of Section 4.1.2. The Q-learning agent has $\beta = 0.2\,m$ and was trained for a total of 3 000 episodes with 1 200 steps per episode, as was the case in Section 6.2.1. A SP of $1.5\,m$ was used throughout training.

$$inlet\ flowrates\ \left(\frac{L}{min}\right) = [50:10:350] \qquad [\ 116\ ]$$

$$inlet\ flowrates\ \left(\frac{L}{min}\right) = [50:10:500] \qquad [\ 117\ ]$$

The resulting action-value tables corresponding to the Ld_SHd_A and the Hd_SHd_A discretization cases denoted in Table 27 and DVs sampled according to Equation [116] are shown in Figure 23 and Figure 24. Figure 25 displays the action-value table resulting from application of the DV range shown in Equation [117] with the H_SH_A discretization case of Table 27. Only the distribution of the non-zero entries across the discretized state-action space is of interest in this section.

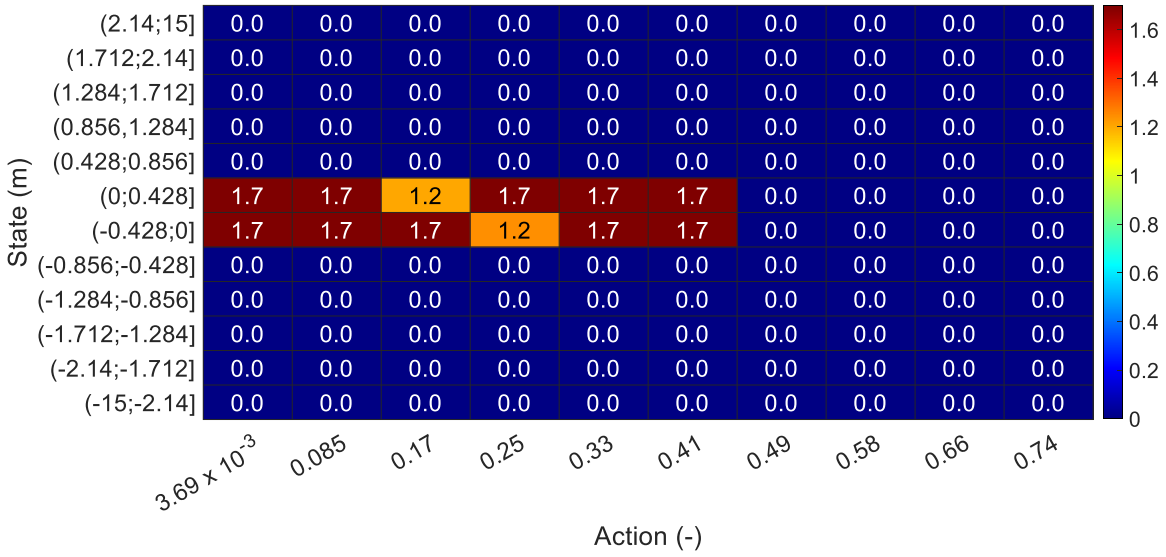


Figure 23: Action-value table of the Q-learning agent trained using the $Ld_S Hd_A$ case of Table 27 and inlet flowrates sampled from Equation [116]

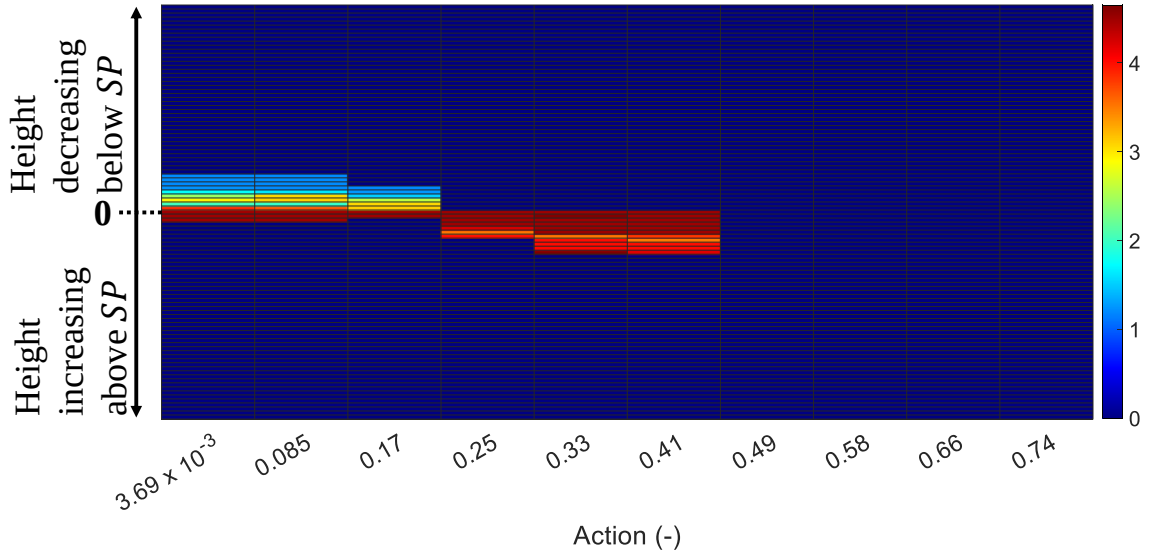


Figure 24: Action-value table of the Q-learning agent trained using the $Hd_S Hd_A$ case of Table 27 and inlet flowrates sampled from Equation [116]

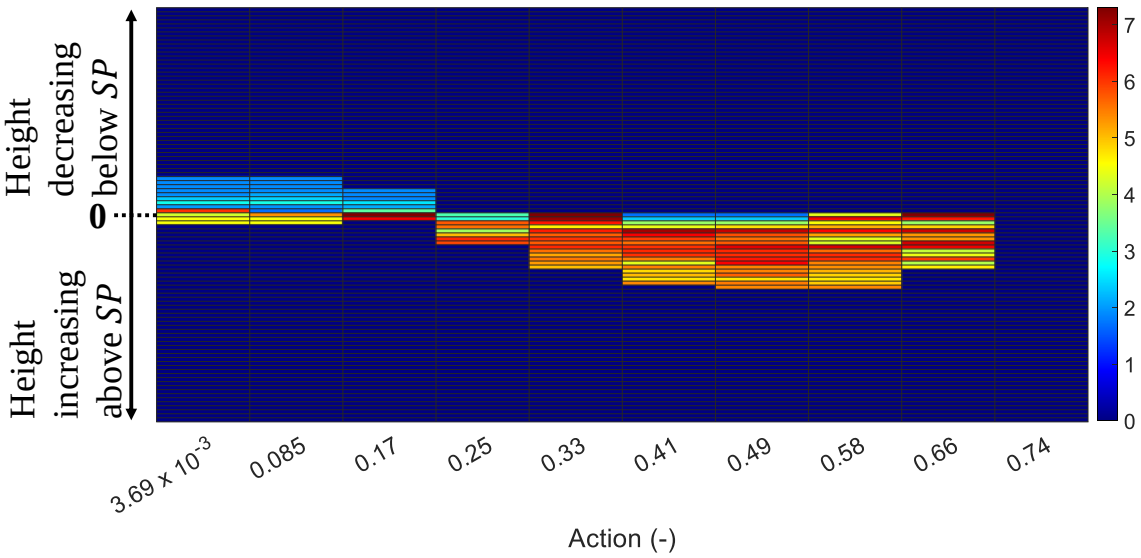


Figure 25: Action-value table of the Q-learning agent trained using the $Hd_S Hd_A$ case of Table 27 and inlet flowrates sampled from Equation [117]

Figure 23 shows non-zero action-values that are arranged symmetrically within the discretized state-action space. Five out of six entries for both the relevant states (the rows in the table) have numerical values that are equal to one decimal place and are larger than the remaining entry at that state. This shows that the state discretization is too coarse for the agent to discern meaningful control behaviour from the PI controller which is capable of making adjustments of small magnitude to the final element.

Figure 24 conveys that the largest action-values are situated towards the right below the discretized state containing $E(T) = 0\ m$ (indicated by **0** at the relevant state), while the opposite is true above this state. This indicates that the agent successfully discerned the logic of opening the valve more if the current level is above SP and closing the valve more if the level is below SP.

The SARSA agents of Section 6.2.1 were also able to learn this logic. The way in which this is achieved with online training is different to off-policy training using Q-learning. Specifically, the SARSA agent visits the discretized states, and starts by filling in positive action-values in the region of $\pm\beta$. This is possible only because the agent explores the RL environment while learning during trial and error. In contrast, the Q-learning agent only experiences the state-action coordinates brought about by the behavioural policy π_b and is directly taught that a level above SP requires opening the valve more, while a level below SP requires closing the valve more.

The process gain K_p is insensitive to the coordinate considered within the process operating window. Many processes do not display this property and therefore necessitate careful consideration when defining the state space $\mathcal{S}$ even when defining coarsely discretized states. In Figure 24, the central row in the region of

non-zero action-value entries has a numerical value of 4.5 for all columns. This indicates that the achievement of zero steady state offset purely by training on the PI control data poses a challenge resulting from the nuances in the PI controller's MV adjustments near SP not being captured in the RL representation.

The behavioural policy π_b (the PI controller) is different from the target policy π which is learnt by the Q-learning agent. Therefore, an asymmetric action-value table may be obtained even though the PI controller does not show preference to certain final element adjustments purely because an error signal $E(T)$ is positive or negative. The symmetry or asymmetry in the Q-learning agent's action-value table is purely driven by the examples provided by the PI controller which, in turn, are a result of the DV ranges encountered. This is illustrated well by the shapes of Figure 24 and Figure 25. The action-value table of Figure 25 contains many more states associated with negative $E(T)$ values for which the agent learnt that opening the valve more is beneficial. This is aligned with the use of larger inlet flow rates in Equation [117].

A potential practical concern is the coverage of the action-value table depicted in Figure 24 as evidenced by the non-zero entries only being present for six valve openings and a limited number of rows. It is fair to expect that a significant number of states won't be visited if the existing control is good. In industrial processes, it is often the case that process operation is, as far as possible, maintained within an optimal region and deviation from this region or the exploration of other regions need to be motivated. If an on-policy agent such as SARSA encounters insufficiently explored state-action coordinates, erratic behaviour will result. Further, while limited instrumentation leads to partial observability and changes in or replacement of process equipment cause changes in the RL environment.

Heuristically Accelerated Reinforcement Learning (HARL) (see, for example, Bianchi et al. (2012)) provides a promising approach to deal with this limitation in certain cases, where PI/PID control may provide guidelines regarding suitable heuristics. In such approaches, the learned policy π is changed through mathematical combination with heuristics obtained from domain knowledge. The Q-learning algorithm therefore shows promise for estimating action-values for simple, self-regulatory SISO control problems.

Importantly, the RL environment studied for Q-learning was not the same as was the case with the SARSA agent in Section 6.2.1. This is because actions selected by the PI controller were not filtered before recording the actions as the behavioural policy π_b actions for training. As will be clarified in Section 6.7, a significantly large filter time constant τ_f is required if a relatively low number of discrete action selections are made available to the RL agent. Filtering the PI controller's actions for prolonged periods of time will unnecessarily degrade control performance. In this thesis, the filter was omitted for Q-learning as it was of interest to see whether a reasonable first guess policy π may be obtained when mapping the continuous adjustments of an analogue controller to a discretized state-action space.

The action-value table will not differ significantly between the two RL environments for a state-action space that is coarsely discretized in such a way that the low-pass filter does not change to which coded states the states of the RL environment map, or if the filter time constant τ_f is small. A small τ_f does not align with the use of coarse discretization, as mentioned above. This is a practical limitation of the methodology used, but is appropriate for the feasibility study conducted.

6.3 Case Study 2 – Van de Vusse Reaction Scheme

6.3.1 Benchmarking Simulations Using SARSA

The steady state operating window for the Van de Vusse reaction scheme model is shown in the contour plot of Figure 26. The bounds for the inlet temperature $\mathcal{T}_0$ and $\frac{\dot{V}}{V_R}$ (the inverse of the residence time) correspond to the bounds prescribed for the SISO control problem described by Chen et al. (1995).

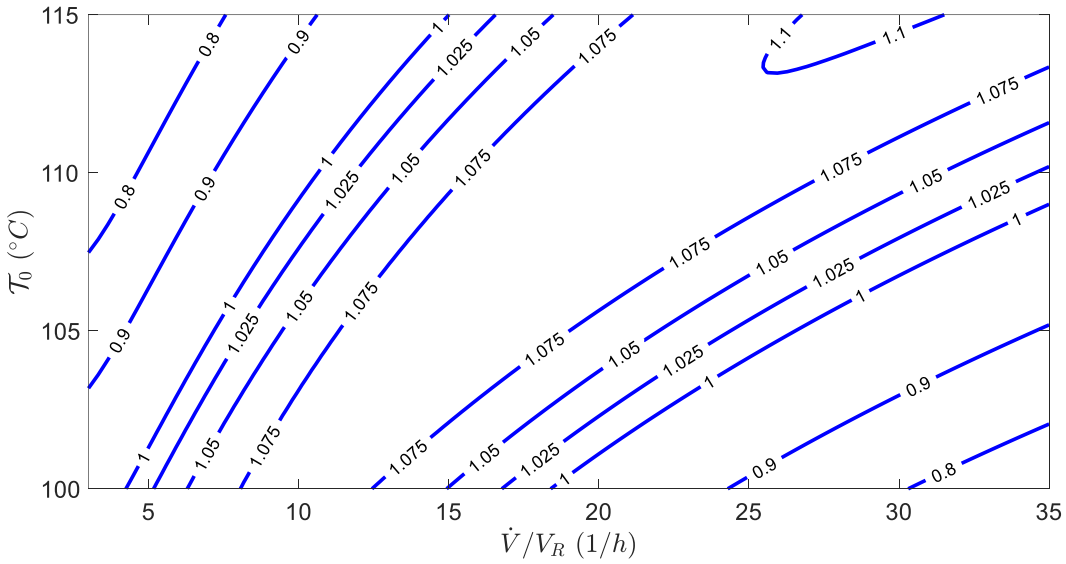


Figure 26: Operating window for the Van der Vusse reaction scheme model displaying lines of constant C_B $\left(\frac{mol}{L}\right)$ for $\dot{Q}_C = -1113.5\frac{kJ}{h}$

Figure 26 shows that larger values for C_B may be achieved at lower values of $\mathcal{T}_0$ for lower inverse residence time values and vice versa. The production of compound B is exothermic. Therefore, at a lower value of $\mathcal{T}_0$, consumption of the reagent is promoted and the yield is larger for reduced inverse residence time. Excessive inlet flow rate would result in insufficient residence time. At large values of $\mathcal{T}_0$, larger inlet flow rate is required since larger reagent concentration is needed in the reactor to achieve the same C_B.

An important observation is made when considering an isotherm drawn at a level of $\mathcal{T}_0 = 105\,°C$ across all the values of inverse residence time within the operating window. The shape of this exemplar isotherm is asymmetric and it contains two instances of every C_B value within the range of flow rates available. This characteristic is prevalent for a wide range of isotherms within the operating window.

This poses a significant challenge to a PI/PID controller, since the classical control law will not be able to select appropriate MV adjustments if different actions are required for the same error signal $E(T)$. From Section 6.2.1, it is known that a tabular SARSA agent's discounted return propagates through the state-action coordinates, originating within $E(T) = \pm\beta$. For a finely discretized state-action space, a SARSA agent will be able to learn that, even though the state $\boldsymbol{S}$ is the same, a different action A is required to obtain more cumulative rewards.

In the simulations conducted, only coarse discretization of the state components is applied. As a result, small changes in SP are required for the agent to select appropriate actions. This is not a limitation of RL-based control, but rather a characteristic of how the feasibility study was approached. In practice, SP values are set externally and typically do not need to change with large magnitudes in short periods of time.

The specifications in Table 20 (Section 5.3.2) show that the training conducted to establish a benchmark for nonlinear control involved 0.16 steps per second and 7 MDP time steps between successive action selections (desired sampling period ΔT converted to the rounded number of MDP time steps). The latter corresponds to a sampling period ΔT of 43.75 seconds.

After training, SP tracking and DV attenuation behaviours were tested by setting the simulated time of the model to $4\,000$ seconds with 200 MDP steps per episode, and thereby allowing the agent to select an action at a sampling period ΔT of 20 seconds. More frequent actions were taken by the agent for the same value of τ_f and this does not compromise the appropriateness of the policy π generated during training. This is because the logic learned by the agent remains unchanged and the merit of tailoring the sampling period ΔT to a particular process is illustrated. For a large sampling period ΔT, the agent would possibly achieve $|E(T)| \leq \beta$ for considerable time periods for certain SP and DV changes. The coarse discretization of the state-action space, large sampling period ΔT, and the dynamic properties of the Van de Vusse reaction scheme model (Section 4.1.1) do, however, not reliably guarantee satisfactory SP tracking and DV attenuation.

The SP tracking results for $\Delta T = 20$ seconds are shown in Figure 27, while the unfiltered action selections and the output of the low-pass filter (filtered actions) are given in Figure 28. Figure 29 and Figure 30 show the qualitative results generated for DV attenuation where changes in $\mathcal{T}_0$ were introduced using step changes ($\Delta T = 20$ seconds). From Figure 27 and Figure 29 it may be observed that there is an initial mismatch between the agent's action selections and the initial input corresponding to the starting value of C_B at steady state. The initial steady state coordinate of the model requires a specific output from the agent. The studied control scheme incorporates agent training without modification to reduce this effect.

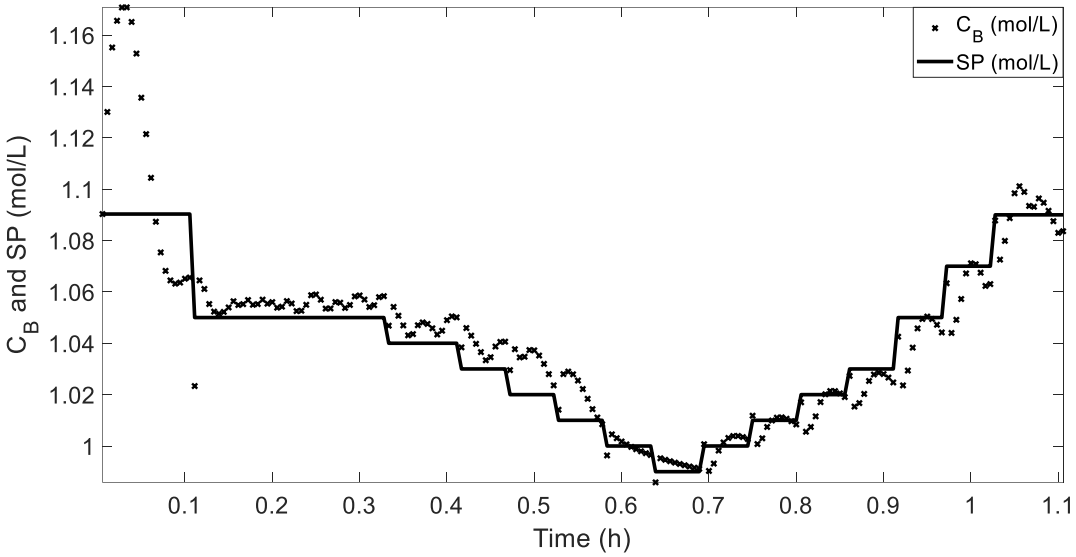


Figure 27: *SP* tracking behaviour for baselining simulations

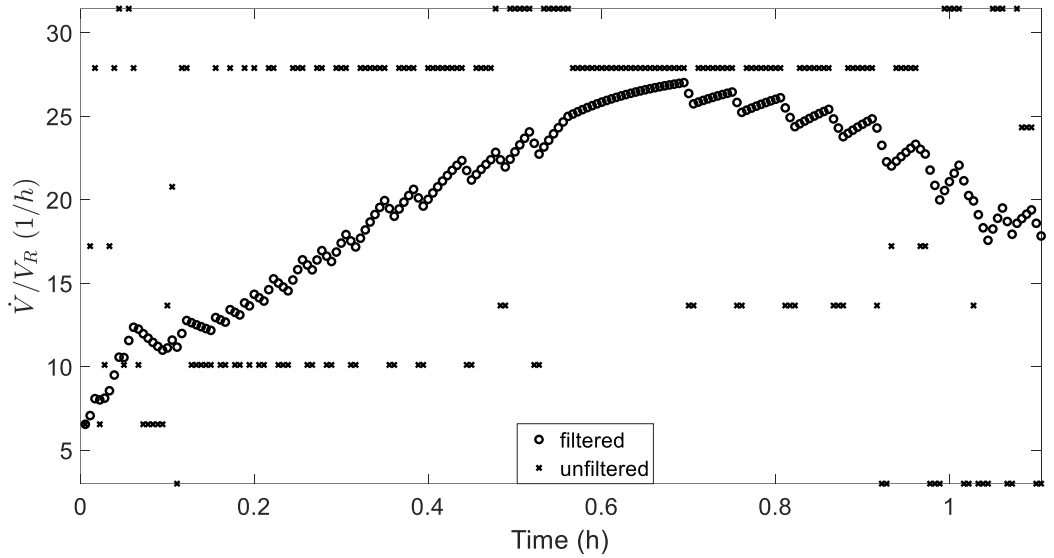


Figure 28: *SP* tracking action selections for baselining simulations

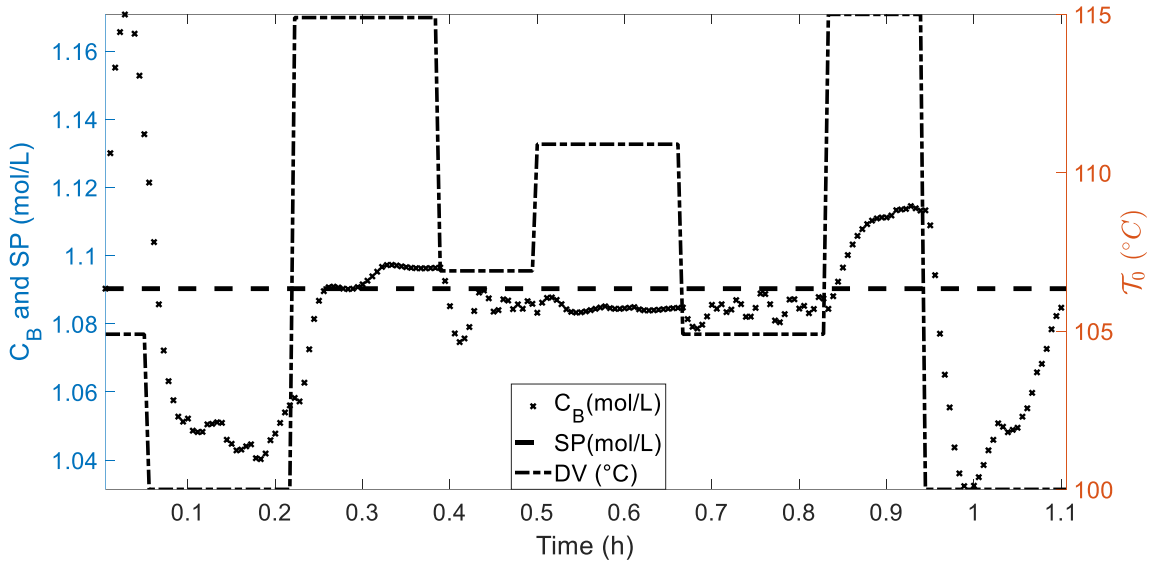


Figure 29: DV attenuation behaviour for baselining simulations, C_B and SP on left scale

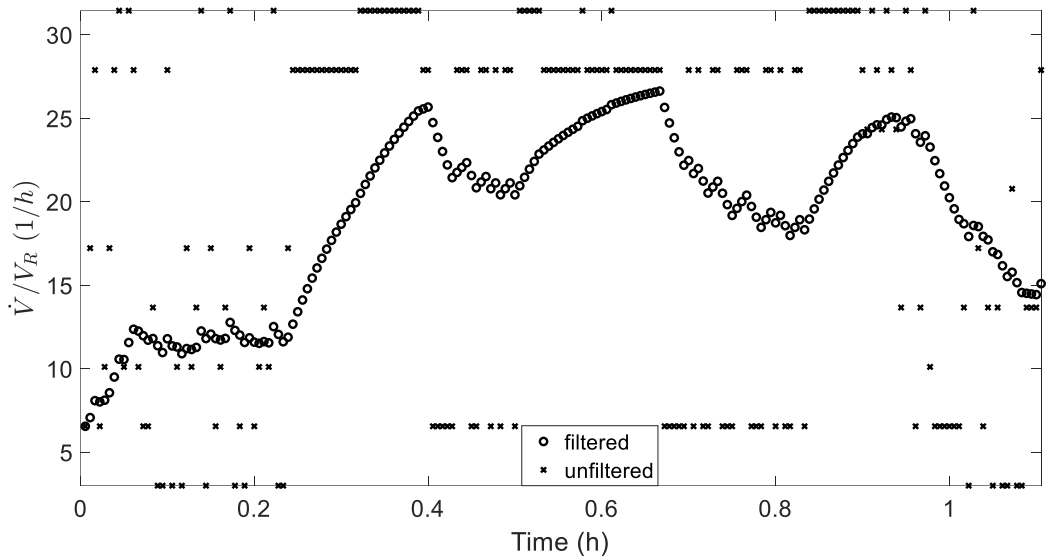


Figure 30: DV attenuation action selections for baselining simulations

Possible approaches to reducing this effect could be to model both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ as continuums, to increase the rate of action selection significantly (subject to available instrumentation), or to decrease τ_f before training provided that sufficient actions are available without a large τ_f. These approaches assume that the initial steady state of the process may be achieved given the bounds in states and actions given to the RL agent – an exception is shown in Section 6.4.

The various action selections that are maintained constant for significant periods of time in Figure 28 and Figure 30 show that that the agent's policy did not merely simplify to a bang-bang control logic which would involve only the application of two different action selections. It may therefore be inferred that, for the value of $\beta = 0.02\frac{mol}{L}$ used, the discretization of the components of $\boldsymbol{S}$ were overall sufficiently fine to

prevent the generation of a bang-bang control law. Further, not all state-action coordinates lie within the area associated with adequate coverage despite on-policy training being used. This is evidenced by the dispersed, low-density selection of actions within the first $0.2\ h$ in Figure 30.

The DV attenuation behaviour shown in Figure 29 illustrates that the trained agent was able to maintain the values of C_B close to SP to a noteworthy degree despite changes in $\mathcal{T}_0$ across the range relevant to the operating window of Figure 26. The ability of the agent to return the value of C_B to SP after the sudden decrease from $\mathcal{T}_0$ from $115\,°C$ to $100\,°C$ is worth pointing out. Chen et al. (1995) highlight that maintaining the $C_B = 1.09\,\frac{mol}{L}$ operating point with a control tolerance of $0.02\,\frac{mol}{L}$ is very difficult during such a decrease in temperature. This effect is dynamic and therefore cannot be inferred by studying the steady-state operating window of Figure 26. The last C_B data point in Figure 29 has a numerical value of $1.086\,\frac{mol}{L}$ which represents an offset smaller in magnitude than the final steady state value achieved for an analogous step decrease in $\mathcal{T}_0$ employed by Chen et al. (1995). It should be noted, however, that their control law was not allowed to use information about $\mathcal{T}_0$ at all.

Qualitative assessment of the results presented in this section with reference to the results reported by Chen et al. (1995) do show that a reasonable controller was successfully synthesized and the studied control scheme is concluded to be suitable to the achievement of the aim and objectives of this thesis.

6.3.2 Investigating the Influence of Finer Discretization of the State-Action Space

The experimental data and the intermediate results of data processing are provided in Appendix E, where the levels of the different factors defined in Section 5.3.3 were coded using -1 and $+1$ for low and high factor levels and used in subsequent calculations. This is necessary to ensure that unintended effects of using unscaled variables are not incorporated. A sampling period ΔT of 40 seconds was used throughout.

The effects of the significant factors are shown in Figure 31. The boxplots were generated by considering all the IAE response data relevant to each of the factor levels reported on the x-axis. The number of data points for the A- and A+ boxes was therefore 800, while the rest of the boxes describe 400 data points each. Figure 31 shows a statistically significant effect of increasing IAE as the error signal $E(T)$ is discretized more finely. This is consistent with the expectation created by the results of Section 6.2.1, as a coarse discretization for $E(T)$ promotes more aggressive action by the agent. As $E(T)$ is discretized more finely, the agent is more equipped to maintain C_B within the $\pm\beta$ tolerance band of $\pm 0.02\,\frac{mol}{L}$. If low IAE is targeted, it may be worth considering discretizing $E(T)$ less finely than the other components of $\boldsymbol{S}$, as is done in this thesis when studying the grinding circuit model (Case Study 3).

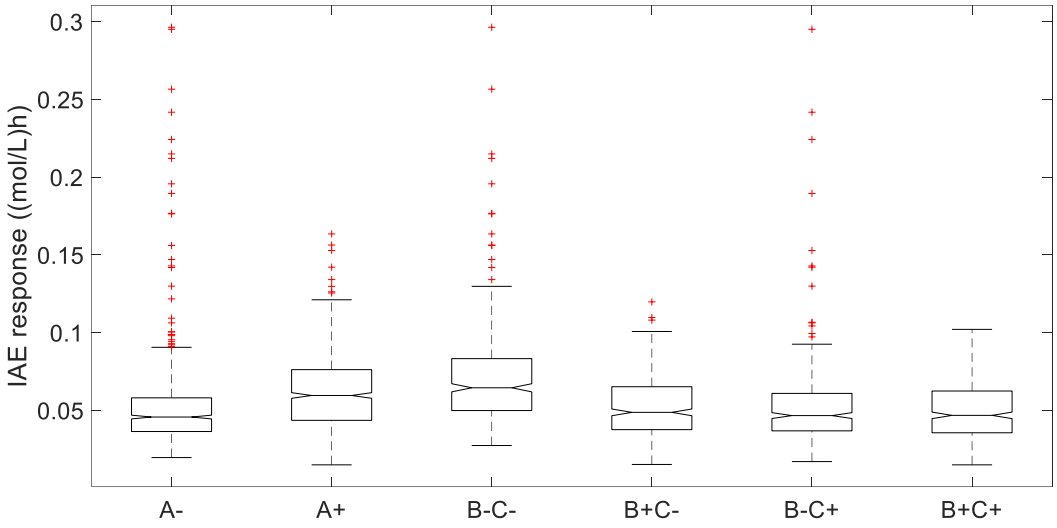


Figure 31: Boxplots summarising the main effect of factor A ($E(T)$ discretization) and the interaction effects of factors B (DV) and C (SP), a '+' sign indicates the fine discretization of a factor, a '-' sign indicates the coarse discretization of a factor, and a red '+' in the graph indicates an outlier

Since DV and SP are involved in significant interactions, it is necessary to consider the interaction effects shown in Figure 31 as the representative description of the experimental effects regarding these two factors. Despite the smaller interquartile range for the A- box in comparison to the A+ box, the number of outliers that show significantly larger IAE responses are much larger for the former. This is evidenced by the number of red crosses above the maxima of the respective box plots. Hence, the aggressive control action displayed when discretizing $E(T)$ coarsely contributes to a greater number of data points having unexpectedly large IAE values. For the interaction of DV and SP, B-C- and B-C+ have the largest numbers of outliers. The different relative numbers of outliers, the positions of the minima and maxima, as well as the interquartile ranges when comparing the B+C- and B-C+ boxes show that the effects of DV and SP discretization are different. In this case study, the B+C- case is associated with a wider distribution of IAEs, but fewer outliers, while the opposite is true for the B-C+ case.

It is observed that increasing the discretization of either DV or SP at a constant discretization level for the other factor results in a decrease in IAE relative to the B-C- case. This is consistent with our intuition. At a fine discretization of SP, the effect of DV discretization on the IAE response is very small – the interquartile ranges and notches (calculated at a 0.05 significance level) of the B-C+ and the B+C+ boxes overlap significantly. Conversely, when the SP is discretized coarsely, the fineness of the DV discretization has a very large effect on the IAE response – B+C- box lies lower than the B-C- box.

Therefore, discretizing all components of $\boldsymbol{S}$ finely does not necessarily contribute to the decrease of IAE. There are two possible reasons for this experimental result. Firstly, coverage could be limiting, where a large number of entries in the action-value hypervolume are not experienced sufficiently during training. Secondly, either discretizing SP or DV finely is sufficient to produce a SARSA controller that achieves the

$\pm\beta$ band which is satisfactory from the agent's perspective for the binary reward function used and the RL environment studied. Interestingly, no outliers were obtained for the B+C+ case, which indicates the greatest consistency in the IAE responses obtained.

The results of this section illustrate that modelling each component of the observed state $\boldsymbol{S}$ as a continuum (which motivates the use of a vast number of parameters and/or function approximation), would not necessarily constitute the optimal approach in terms of controller performance for a specific reward function and control problem. This is not intuitive, as conceptually one would likely expect that the modelling of the state-action space as a continuum would always yield the best trained policy π for any control problem. This section stresses that finely discretized state components need to enable effective leveraging of the reward function to prevent incorporating an excessive number of parameters in the RL representation during the design of an RL-based controller. Further, the number of parameters to learn, the controller performance obtained, and the frequency with which outliers are encountered need to be considered when designing an RL-based controller.

The discretization of the available actions (factor D) does not contribute significantly to the reduction of IAE, as evidenced by a P-value of $0.27 > 0.10$ for this factor. This illustrates that it is the range of action selections available to the agent that is important if one only includes deviation from SP in the reward function.

6.3.3 RL Hyperparameter Properties for SARSA

Figure 32 shows average IAE values and their corresponding 95% confidence intervals, calculated as described by Devore (2017), for γ and α, respectively. A sampling period ΔT of 40 seconds was used throughout.

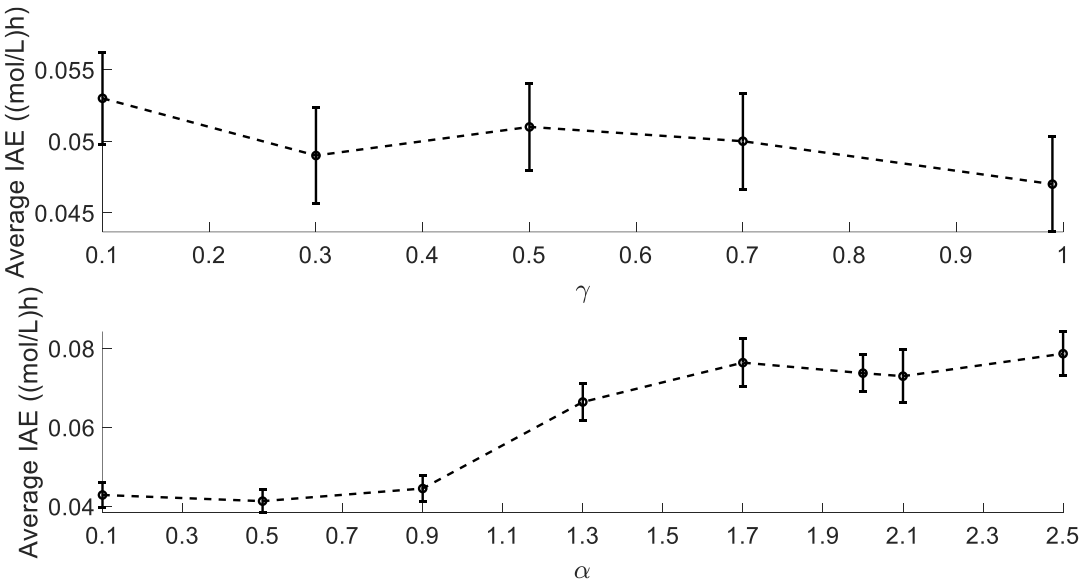


Figure 32: IAE responses with 95 % confidence intervals obtained at different values of γ with $\alpha = 1$ (top panel), and at different values of α at $\gamma = 0.99$ (bottom panel)

From Figure 32, it is observed that the average IAE decreases slightly as γ is increased towards unity. This is aligned with the author's expectation – maintaining CV in the vicinity of SP for long time periods is required in a control problem.

The IAE's are not very sensitive to the value of γ chosen as indicated by the range of IAE points generated for the values of γ considered. The confidence intervals also show that the bounds within which the true population means of the IAE values will lie with 95% confidence for a very large number of replicates are approximately equal in size. This is a reflection of the fact that SARSA's convergence properties are not affected by the specific value of γ, provided that $0 < \gamma \leq 1$.

From Figure 32, for values of $\alpha > 1$, the IAE increases rapidly in comparison to $0 < \alpha < 1$. The uncertainty in the resulting policy π increases significantly then, as shown by the noteworthy increase in the confidence intervals' sizes for $\alpha > 1$. This confirms that there is no justification for using $\alpha > 1$ when applying tabular SARSA, and that the learned behaviour is insensitive to the value of $0 < \alpha < 1$ selected.

6.4 Case Study 3 – Grinding Circuit

6.4.1 Placing the Results in Context

In this study, tabular RL methods were tested in simulations with the ideal result being that such methods would result in an interpretable controller with moderate operational data requirements in comparison to deep learning methods applied to large control problems (Section 6.5). For Case Study 1 (water tank model, Section 6.2) and Case Study 2 (Van de Vusse reaction scheme, Section 6.3) this worked well (subject to the practical limitations of the RL methodology used), but not for Case Study 3 (grinding circuit model). For the results of Case Study 3, the PID controller outperformed all elementary RL-based control methods applied. The value of the results presented in this section lies in highlighting nuances regarding how elementary RL agents learn in process control systems requiring significantly robust control and certain challenges that may prevent successful industrial application of RL-based control at present.

6.4.2 PID Controller MFS Adjustments

First, the performance of the PID controller, both with and without stiction, was evaluated. This aided in the selection of parameters for the stiction model (S and J in Section 4.3.3) as a sensible degree of degrading controller performance had to be simulated. Figure 33 shows two superimposed outputs of the PID controller without stiction and with stiction. These outputs correspond to the grinding circuit PID control output data presented in Section 6.4.4.

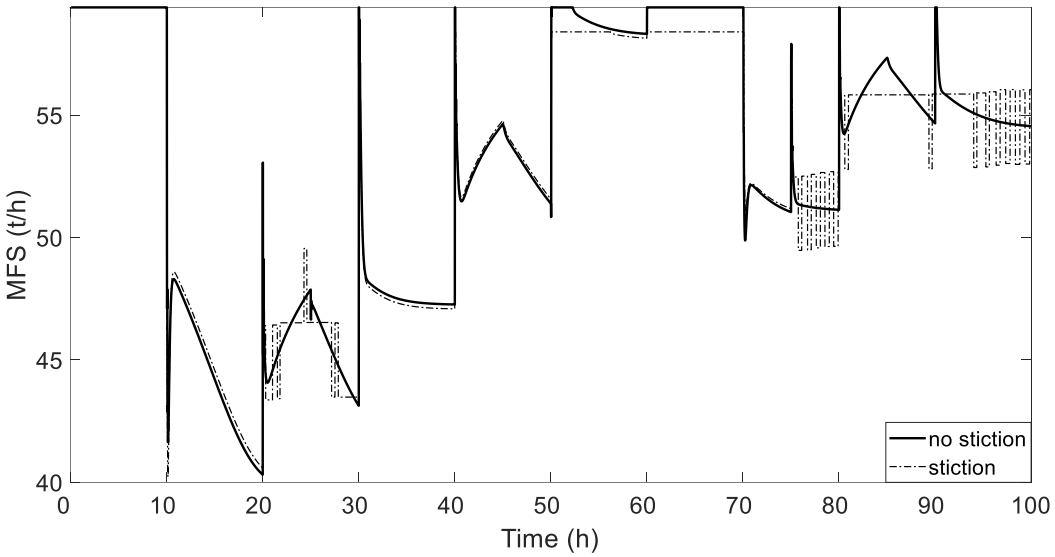


Figure 33: The MV adjustments of the PID controller without and with stiction (J = 2 % and S = 4 % of the MFS range $0\,\frac{t}{h}$ to $100\,\frac{t}{h}$, respectively)

Firstly, the oscillations in PID controller output (from approximately 95 h to 100 h, for example) are known as limit cycles and have the effects of causing excessive wear in the final element and low-amplitude variations in the CV. In reality, stiction would increase over significant time periods and the classical PID controller with predefined tuning parameters cannot automatically adapt to these parameters.

6.4.3 Process Operating Window

Figure 34 illustrates the operating window created by numerically solving the grinding circuit model using a simulated time period of 100 h at each level of $(MFS; \phi_r; \upsilon)$ with changing ϕ_f implemented as step changes with magnitudes and times corresponding to the work of Le Roux et al. (2013).

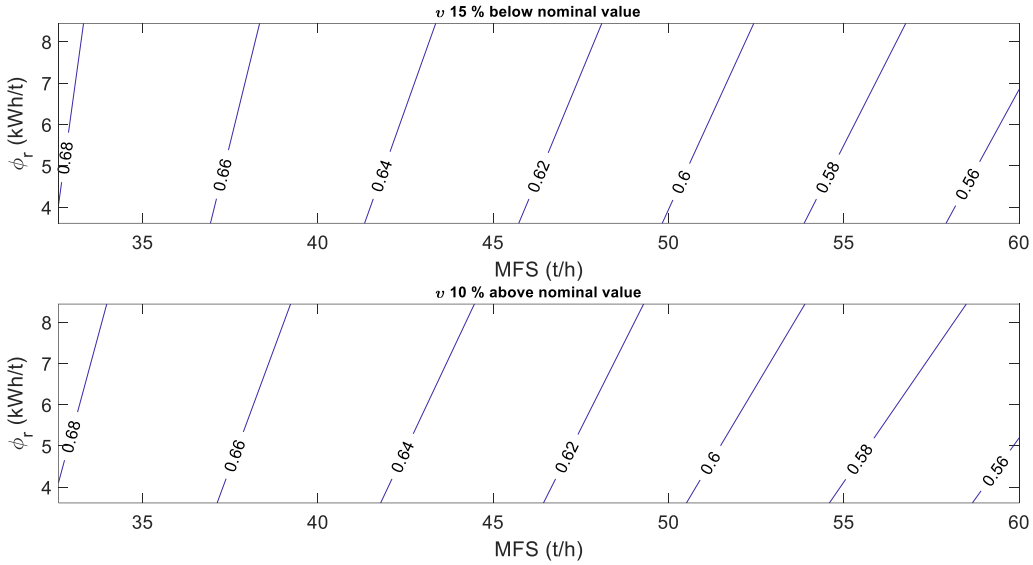


Figure 34: Operating window under consideration for grinding circuit study – the contours represent lines of constant PSE (-)

The PSE contours shift in the direction of decreasing MFS as the ore rock fraction υ is increased. This makes sense as an increased rock fraction υ causes an increase in the solids loading of the SAG mill and a coarser classification in the cyclone – MFS needs to be reduced to achieve a similar PSE under these conditions. Each contour of constant PSE has a positive gradient. The reduced values of ϕ_r result in an increase in rock consumption (RC, Equation [78]). This increase results in a larger conversion of material from the "rock" state to the "solids" state in the grinding circuit model. Subsequently, the effect of reducing ϕ_r is similar to increasing MFS, resulting in the requirement that the MFS input should be reduced to compensate for this effect.

6.4.4 Application of SARSA-Based Controller to the PSE-MFS Control Loop

Before discussing the characteristics of the SARSA-based controller studied, it must be stressed that qualitative characteristics of the control responses generated do not represent satisfactory control performance. For the lower value of τ_f used, MV adjustments were excessive, resulting in underdamped PSE responses that aided in achieving smaller IAE values. For the larger setting of τ_f, the successive MV adjustments applied to the process were not sufficient to maintain the resulting PSE values within the $\pm\beta$ tolerance band around SP.

Recall from Section 6.4.1 that the aims of the results were to understand the learning processes of elementary RL agents and to highlight challenges that may prevent successful industrial RL-based control at present. A summary of the control performance measures is shown in Table 28, while PSE behaviours generated without and with stiction are presented in Figure 35 and Figure 36, respectively. Figure 37 displays the unfiltered and filtered actions corresponding to the PSE outputs in Figure 35 for $\tau_f = 5\ h$.

Table 28: Control problem cases and measures for comparison of PID to SARSA-based control (corresponds to input perturbations of Figure 14 (Section 5.3.5) and SP changes of Figure 35 and Figure 36)

Controller	**S** (%)	**J** (%)	$\boldsymbol{\tau_f}$ $(\boldsymbol{h})$	$\boldsymbol{IAE}$ $(-)$	$\boldsymbol{ITAE}$ $(\boldsymbol{h})$	$\boldsymbol{TV}$ $\left(\frac{\boldsymbol{t}}{\boldsymbol{h}}\right)$
SARSA	0	0	5	2.80	102.7	103.1
SARSA	0	0	0.5	2.1	80.5	210.6
PID	0	0	-	1.7	63.8	223
PID	4	2	-	1.62	64.7	252.8
SARSA	4	2	5	3.8	135.7	16.7
SARSA (adjusted)	4	2	5	2.9	104.1	11.7

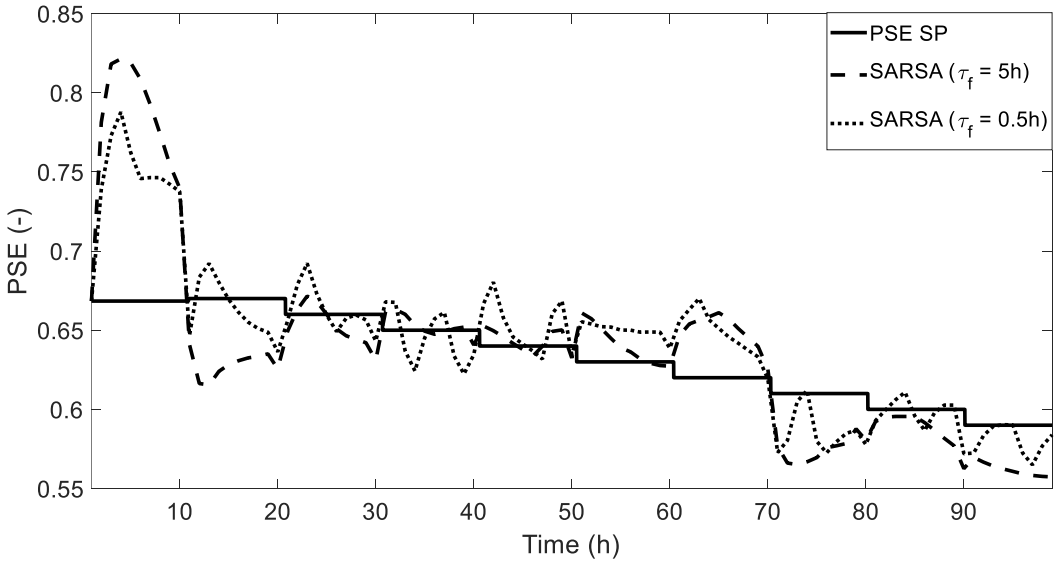


Figure 35: SARSA ($\tau_f = 5\ h$) and SARSA ($\tau_f = 0.5\ h$) control behaviour in accordance with Table 28

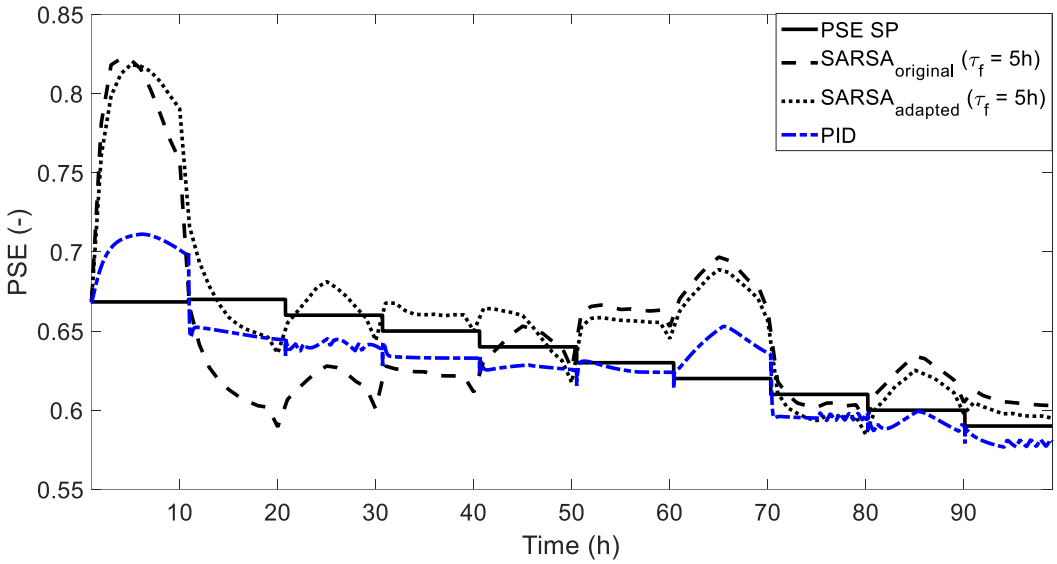


Figure 36: PID, SARSA and SARSA trained for 176 additional episodes in the presence of stiction in accordance with Table 28

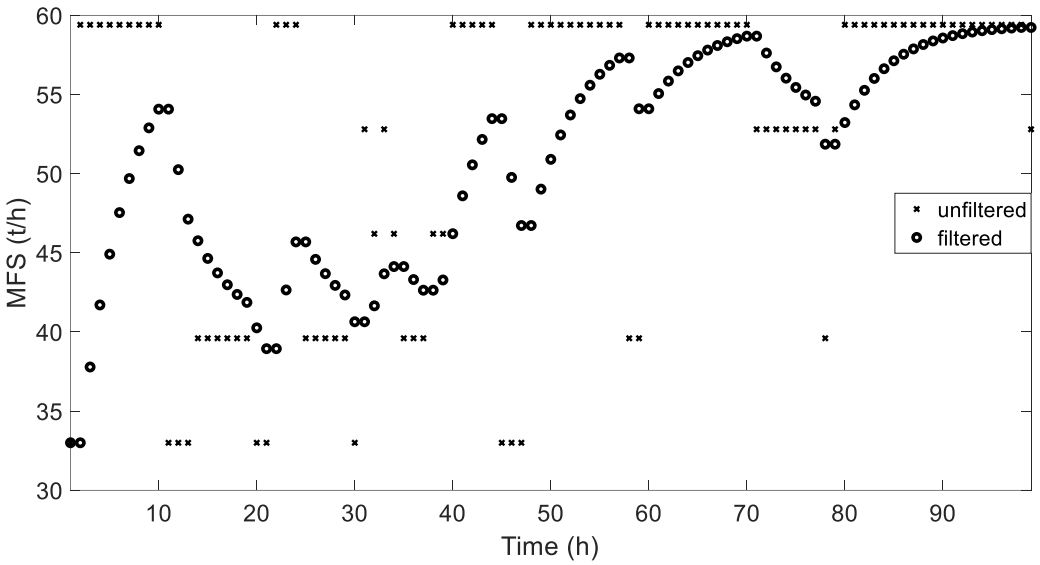


Figure 37: MV behaviour for SARSA with $\tau_f = 5\ h$ in accordance with Figure 35

The data presented qualitatively in Figure 35 through Figure 37 illustrate SARSA-based control's capability of establishing a policy that exhibits some degree of robustness and was observed in Sections 6.2 and 6.3 as well as in literature (Syafiie et al., 2008; Brujeni et al., 2010; Ramanathan et al., 2018). Specifically, this is shown by the ability of the SARSA policies to keep PSE close to SP after 10 h for the test control problem of Figure 14 (Section 5.3.5). This is partly a result of the filtering of the discrete action selections. The selection of τ_f for study is discussed in Section 6.7 as it relies on an understanding of the results for the different RL agents in this chapter.

Significant robustness limitations are observed between $50\,h$ and $70\,h$ on the x-axis of Figure 35 and Figure 36 and are largely a result of the concurrent changes in ϕ_r and ϕ_f in a relatively short time period. This shows that a robustness limit is applicable when applying SARSA as the RL-agent.

A large initial change is observed in all PSE response figures – the initial steady state MFS value of $65.2\,\frac{t}{h}$ does not lie within the steady state operating window given in Figure 34 and is therefore excluded in the MV range available to the agent. This is a result of the changes to the parameter ϕ_f. The control performance measures presented in Table 28 consistently include the influences of these initial mismatches.

Decreases in IAE and ITAE with concurrent TV increases are observed in the first two rows of Table 28 as τ_f is decreased. The agent is consistently outperformed by the PID controller in terms of the overall reduction in the error signal as evidenced by the lower IAE and ITAE values in row three of Table 28.

Figure 38 was generated by introducing the υ, ϕ_r, and ϕ_f changes of Figure 14 (Section 5.3.5). Each integer value on the x-axis of Figure 38 is associated with a recorded data point on the y-axis. It should be emphasized that the first data point in Figure 38 corresponds to 100 episodes of training. When initialising $\tilde{Q}_\pi(\boldsymbol{S}, A) = \mathbf{0}$, the values of the control performance measures were 3.4, $180.4\,h$, and $72.2\,\frac{t}{h}$ for IAE, ITAE and TV, respectively. Therefore, Figure 38 shows that the steepest ITAE decrease with $\tau_f = 5\,h$ occurs in the first 100 episodes (1.1 years). A more stable policy π for the fixed control problem is achieved after 3 500 episodes (40 years) and after 14 000 episodes (160 years), respectively. Similar observations are supported in terms of operational data requirement when considering the variances of all three control performance measures. The recording of the measures also involved running the SARSA update rule in between the addition of the experience of each 100 episodes of training.

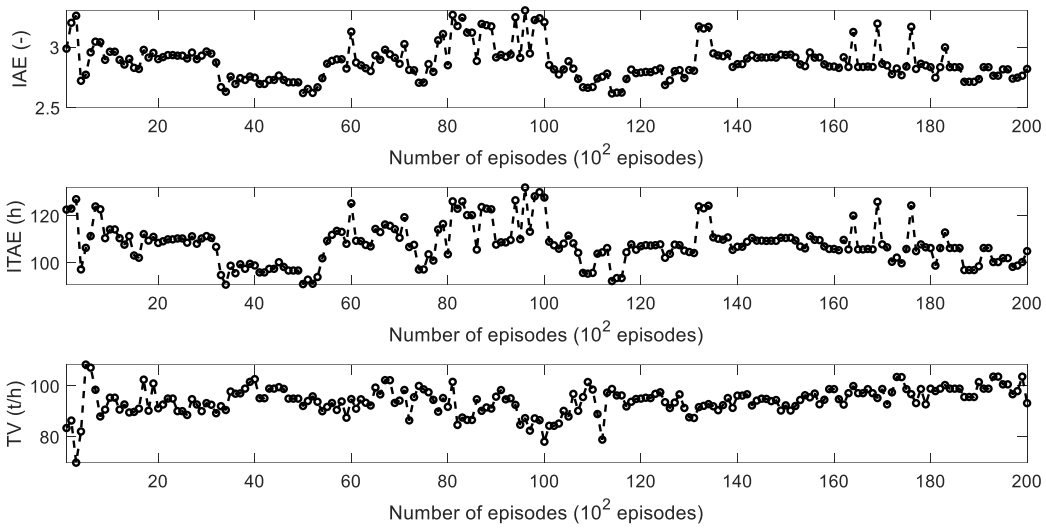


Figure 38: Progression of control performance measures for SARSA ($\tau_f = 5\ h$) (corresponds to input perturbations of Figure 14 (Section 5.3.5) and SP changes of Figure 35 and Figure 36)

Despite how quickly the initial improvement of SARSA's policy occurs when considering the ITAE response, the rest of the control performance progression indicate that it is only a certain portion of $\tilde{Q}_\pi(\boldsymbol{S}, A)$ that has achieved a well-established policy. This is evidenced by variations in IAE, ITAE, and TV along the entire x-axis of Figure 38 without clear stabilisation of the controller performance measures. It is important to realise that Figure 38 was generated for a single exemplar control problem, while the Policy Improvement theorem (Section 2.6) is applicable to the RL problem as a whole. Therefore, improvements to the overall policy do not imply that the control performance measure progression for the exemplar control problem will be monotonic.

While the TV values are increasing with concurrent decreases in IAE and ITAE during the final portion of the response, the trend will likely not be maintained until $\tilde{Q}_\pi(\boldsymbol{S}, A)$ converges. One can also not predict when $\tilde{Q}_\pi(\boldsymbol{S}, A)$ will be fully converged, but the training is still limited to 20 000 episodes since the entire training period is already representative of an unrealistically large requirement for the number of operational hours used during training. The results show that the SARSA agent has an unrealistically large operational data requirement.

On a practical level, the cost of efficient use of plant operating hours is to allow more time to pass between the generation of operational data points. This also simplifies the control problem for the RL agent. Decreasing the sampling period ΔT will result in the availability of more operational data points within the same time period of operation, likely accompanied by a greater operational data requirement since a smaller fraction of the process dynamic response occurs between MDP time steps.

Table 28 shows that the TV values of SARSA are much lower with stiction and this is a result of the filtered action selections often not having sufficient amplitudes to overcome stiction. Adjusted SARSA's policy

comprises the originally generated SARSA agent policy after training with a further 176 episodes (2 years) in the presence of stiction. Adjusted SARSA's lower values for IAE, ITAE, and TV in comparison to the initially trained SARSA agent's measures shows that the RL-based controller is, in a sense, capable of leveraging the stiction to maximise the cumulative rewards it receives. The ability of SARSA to compensate for stiction is considered promising as it typically becomes more severe over long periods of time.

In the presence of stiction, a significant increase in excessive final element adjustment is observed for the classical PID controller (TV value in the fourth row of Table 28). Stiction causes low-amplitude oscillations in the corresponding PSE output of Figure 36 as a result of the limit cycling shown in Figure 33.

Coverage is also a problem, as is evidenced by the non-zero entries of $\tilde{Q}_\pi(\boldsymbol{S}, A)$ at the end of the 20 000 episodes only numbering 2 319 (59 %) for the $\tau_f = 5\ h$ run, 2 397 (61 %) for the $\tau_f = 0.5\ h$ run, and 2 444 (62 %) for the adapted SARSA run. The total number of discrete entries in $\tilde{Q}_\pi(\boldsymbol{S}, A)$ (including the boundary states) was 3 960. The coverage limitations arising in discretized state-action spaces is a key driving force in the study of function approximating agents.

While coverage could be enhanced by the use of function approximation, the large variance in the control performance measures are a result of the large variances in the parameter estimates of stochastic approximation algorithms which are enhanced by the action selections being very sensitive to the position of the maximum action-value for each state observed (Sutton and Barto, 2018; Theodoridis, 2020).

6.4.5 Application of Q-learning to the PSE-MFS Control Loop

The progression of the control performance measures is shown in Figure 39. Stiction was not included during the training of the Q-learning agent. Recall from Section 6.2.2 that the filter time constant was not incorporated when studying Q-learning and the implications thereof.

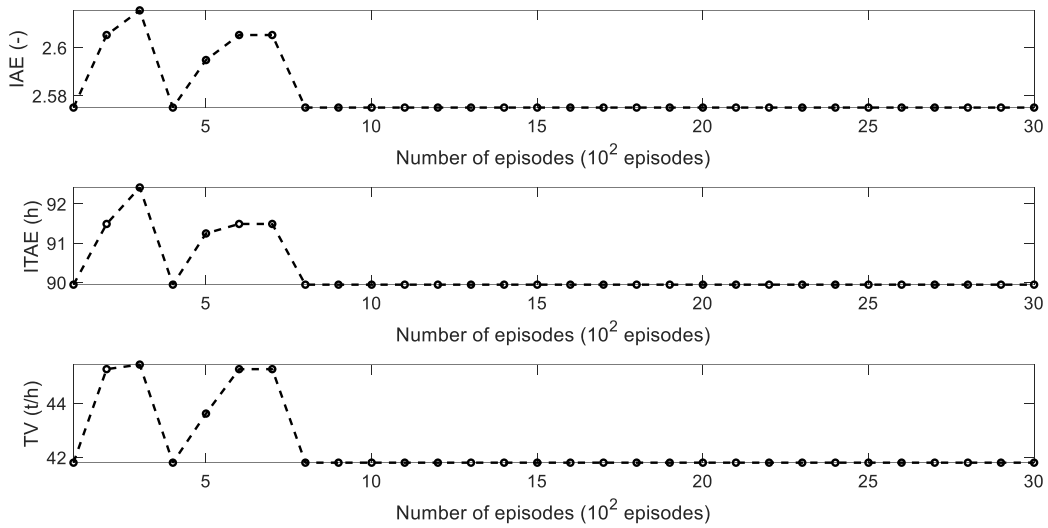


Figure 39: Progression of control performance measures for Q-learning ($\tau_f = 5\ h$ when providing the policies to a SARSA agent) tested on the example control problem

The control performance measures reported were generated by connecting the policy generated using Q-learning online as the first estimate of a SARSA policy and running the example control problem used in Section 6.4.4 after each multiple of 10^2 training episodes are added to $\tilde{Q}_\pi(\boldsymbol{S}, A)$. The training consisted of 3 000 episodes, where each episode consisted of 10 000 steps with a sampling period of $\Delta T = 0.01\ h$ since this is the sampling period at which the PID control calculations were performed. The PID controller was not saturated when used as π_b, but the data pertaining to the action selections were saturated to the range of the available final element adjustments when mapping the training data to the appropriate state-action coordinate of the Q-learning agent.

The statistics reported in Figure 39 show a significant improvement in comparison to the statistics reported for SARSA in Table 28. A 0.225 decrease in IAE, a 12.76 h reduction in ITAE, and a 61.29 $\frac{t}{h}$ reduction in TV were achieved in comparison to the SARSA results with $\tau_f = 5\ h$. The performance obtained for the RL representation design used was still worse than the PID controller results – IAE was 0.875 larger, ITAE was 26.14 h larger. The final element adjustments were much lower – TV was 181.19 $\frac{t}{h}$ lower for Q-learning.

As the PID controller responds deterministically and immediately to the error signal sent to it, the agent is capable of benefiting from the steps provided to it at a sampling period of $\Delta T = 0.01\ h$. Figure 39 illustrates well the potential problem arising from coverage limitations, since the agent's policy converged within the first 800 episodes (9.13 years). This is only possible if a certain fraction of the action-value hypervolume was experienced far more often during training. Out of the 3 960 entries in $\tilde{Q}_\pi(\boldsymbol{S}, A)$, only 965 (24 %) had non-zero entries at the end of training which shows quantitatively that coverage is constrained by the examples provided by the PID controller.

6.4.6 Application of Actor-Critic-Based Controller to the PSE-MFS Control Loop

Actions were discretized from $33\frac{t}{h}$ to $60\frac{t}{h}$ in increments of 10%. The control performance measure progression results are shown in Figure 40, where this figure was generated in the same way as Figure 38. The vertical dashed lines in Figure 40 correspond to five phases of the RL agent's training – an initial improvement in behaviour, reaching best controller performance with large variance in TV values, and three phases associated with reducing variance in the control performance measures and the stabilisation of the actor's parameter vector $\boldsymbol{\theta}$.

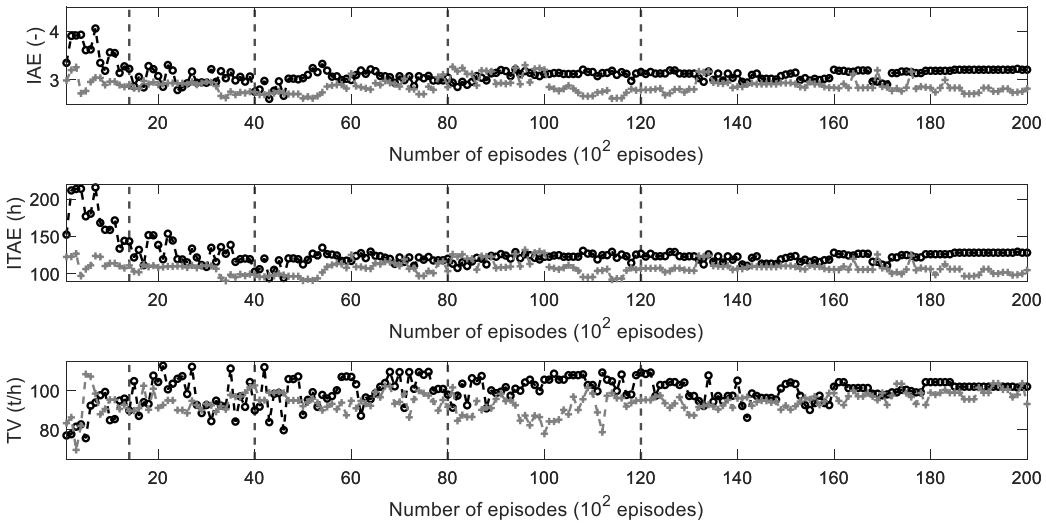


Figure 40: Progression of control performance measures for One-Step Actor-Critic ($\tau_f = 5\ h$) (black circles at data points) and SARSA (grey '+' markers corresponding to data points of Figure 38) – the boundaries of five regions of policy determinism are indicated using vertical dashed lines

Since linear regression problems were set up for the One-Step Actor-Critic and SARSA agents, the number (and arrangement) of parameters used constrains the ability of each agent to deal with the control problem. Therefore, the fact that the SARSA agent's control performance measures are "better" merely indicates that insufficient parameters were provided to the One-Step Actor-Critic agent to capture all the contours of the optimal policy π^* during training despite its increased coverage arising from the use of function approximation (Section 2.9.6).

As such, the monotonic decrease in both ITAE and IAE may be contrasted to the SARSA agent that shows a significant decrease in ITAE within the first 100 episodes, followed by unpredictable changes in the control performance measures (Section 6.4.4). The initial IAE and ITAE values in Figure 40 differ considerably since the Actor-Critic agent's learning is much more gradual. This is a result of modelling the state space as continuous, the differences between the learning rates of the two agents, and the different types of policy parameterization.

Figure 40 shows that IAE and ITAE both show monotonic response shapes with a significantly smaller extent of variance in the control performance measures as training progresses than is the case with SARSA. The TV values show a significantly more gradual increase than is the case with SARSA. After approximately 2 000 training episodes (22.8 years), a pronounced variance is observed in the TV values. This is followed by a reduction in the variance of TV values between just over 5 000 episodes (57 years) and 16 000 episodes (182 years). The TV values stabilise by 18 500 episodes (211 years).

There are two properties of the One-Step Actor-Critic agent that allows for a control performance measure trajectory that may more easily be modelled. Firstly, $\pi(A|\boldsymbol{S},\boldsymbol{\theta})$ follows a soft-max distribution and therefore is not prone to abrupt changes in policy that temporarily decrease performance in regions of the state-action space as is the case with SARSA which relies on the discrete action-values. Secondly, the critic aids in the reduction of the variance of the parameter estimates about the values corresponding to the optimal parameter vector $\boldsymbol{\theta}^*$ (Sutton and Barto, 2018; Theodoridis, 2020).

During training, the inputs to the grinding circuit model and the SP were changed in a pseudorandom fashion as described in Section 4.3.2. Therefore, a learning curve summarising the rewards obtained by the agent during its training will not necessarily have a clearly discernible monotonic shape. It is assumed that a fixed starting state $\boldsymbol{S}_0$ was present in the vast majority of the training episodes, as is required for the One-Step Actor-Critic agent (Section 2.9).

Figure 41 shows the rewards summed for each test on the exemplar control problem as training progressed, with the One-Step Actor-Critic agent's graph shown in the top panel – these curves are also not monotonic. The trajectories of the rewards obtained by the agent do not correspond qualitatively to the control performance measure behaviours reported in Figure 38 and Figure 40.

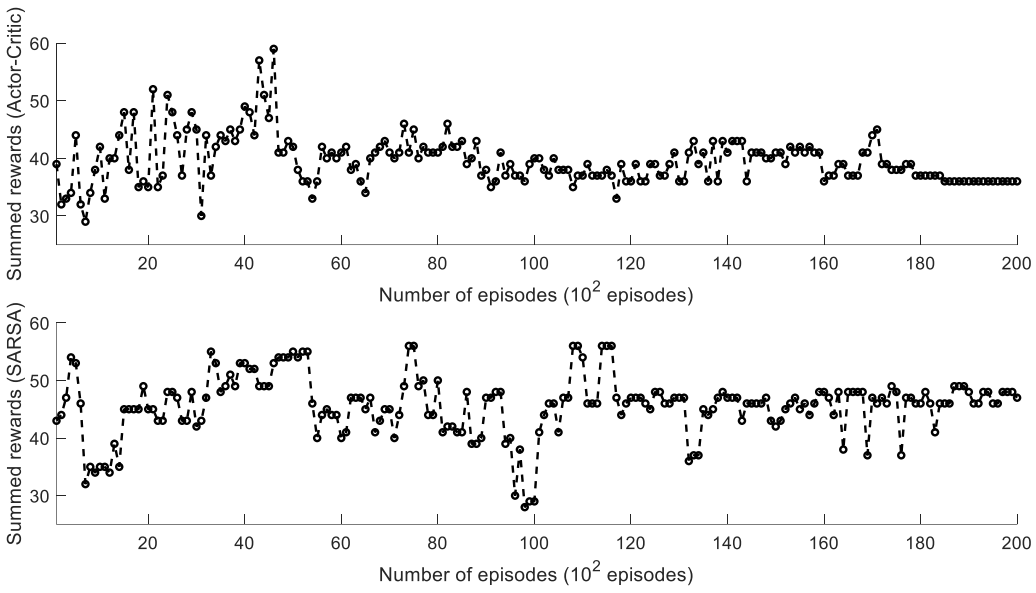


Figure 41: Progression of summed rewards for the exemplar control problem – each circle corresponds to a data point

For the One-Step Actor-Critic agent, there is an initial increase of summed rewards obtained for the test control problem as training progresses. This continues until training episode 4 600, before a sudden drop occurs in the summed rewards. After this sudden drop, the trajectory of summed rewards shows a variance about some unknown mean value without a discernible trend during the progression of the control performance measures. The sudden drop in rewards is a result of the preference function $h(\boldsymbol{S}, A, \boldsymbol{\theta})$ being driven towards much larger values in certain regions of the state-action space.

Interestingly, the control performance measures do not show the same behaviour as the rewards – the IAE and ITAE essentially maintain their minimum values obtained during the period of stochastic action selections (second phase in Figure 40). This illustrates that classical control performance measures are not unique to a particular instance of the parameter vector $\boldsymbol{\theta}$ as the optimal parameter vector $\boldsymbol{\theta}^*$ for the reward function used did not need to be learned before achieving the best control performance in terms of CV deviation.

The communication of insufficient control objectives for continuous state space modelling can, however, lead to unintended control behaviour. The stabilisation of the data points in Figure 40 and the drop in rewards correspond to a bang-bang control logic that was learnt by the RL agent. The One-Step Actor-Critic agent leant that, for a binary reward function, using the two action selections at the extremes of the available range is the most effective for maintaining the CV in the vicinity of the SP for the binary reward function used. Improving the policy across the entire state-action space may result in a decrease in the rate at which rewards are received when considering a specific control problem, as is evident in Figure 41.

In contrast, the reward progression in the bottom panel of Figure 41 shows that a qualitatively unpredictable pattern is displayed by the tabular SARSA agent. The total number of parameters used for the One-Step Actor-Critic agent $(625 + 64 = 689)$ is much less than the 3 960 (including the boundary states) available to the SARSA agent. Therefore, the quicker convergence of the One-Step Actor-Critic agent cannot solely be attributed to the use of the critic and is likely largely influenced by the number of parameters in the RL representation.

The One-Step Actor-Critic agent was not able to overcome stiction within the same constraints used for the SARSA agent in Section 6.4.4. Stiction compensation for the One-Step Actor-Critic agent is therefore limited to slowly changing closed-loop dynamics.

6.5 Placing Operational/Plant Hours in Perspective

Table 29 summarises the numbers of parameters used in various RL-based process control studies. The number of MDP steps are not indicated for evolutionary RL since the number of parameters is adjusted by the RL agent. Appendix F provides sample calculations for determining the numbers of parameters used in RL representations of the works cited.

Table 29: Summary of numbers of parameters used in various works

Work	Chemical Engineering Problem	RL Algorithmic Principle	Total Number of Parameters	Parameter Ratio (Case Study 3 /cited work)	Actor	Critic	Total number of MDP steps used
(Shipman and Coetzee, 2019)	Tuning a PI controller for a stochastic instance of a FOPTD model.	Actor-Critic (A2C)	42 851	0.092 (SARSA) 0.016 (Actor-Critic)	21 410	21 441	1×10^8
(Ma et al., 2019)	Control of a batch polymerization reaction's product quality.	Actor-Critic (DDPG)	352 204	0.011 (SARSA) 0.002 (Actor-Critic)	218 402	133 802	95 200 (Increase in MW)
(Conradie, 2000)	Control of the Agrawal bioreactor.	Evolutionary RL (SANE)	97	40.82 (SARSA) 7.10 (Actor-Critic)	N/A	N/A	N/A
	Van de Vusse reactor control (MIMO).	Evolutionary RL (SANE)	106	37.36 (SARSA) 6.5 (Actor-Critic)	N/A	N/A	N/A
	Multi Effect Batch Distillation Column control.	Evolutionary RL (SANE)	106	37.36 (SARSA) 6.5 (Actor-Critic)	N/A	N/A	N/A
(Conradie and Aldrich, 2001)	Control of PSE in a pilot plant scale grinding circuit model developed in the first part of the study by Rajamani and Herbst (1991).	Evolutionary RL (SANE)	149	26.6 (SARSA) 4.62 (Actor-Critic)	N/A	N/A	N/A

Work	Chemical Engineering Problem	RL Algorithmic Principle	Total Number of Parameters	Parameter Ratio (Case Study 3 /cited work)	Actor	Critic	Total number of MDP steps used
This work	Control of PSE in the grinding circuit model described by Le Roux et al. (2013) (Sections 6.4.4 and 6.4.6).	Value-based (SARSA)	3 960	1 (SARSA) 5.75 (Actor-Critic)	N/A	N/A	2×10^6
		Actor-Critic (One-Step Actor-Critic)	689	0.17 (SARSA) 1 (Actor-Critic)	625	64	2×10^6
	Van de Vusse reactor control (SISO) (Section 6.3.1).	Value-based (SARSA)	220	18 (SARSA) 3.13 (Actor-Critic)	N/A	N/A	8×10^5

The number of episodes in the work of Ma et al. (2019) is counter-intuitive considering that the number of parameters in their work is by far the most. This may be attributed to their judicious use of domain knowledge through simplification of the control problem by including historical information and target MW values in their definition of the observed state $\boldsymbol{S}$, and training the RL agent on target trajectories for increasing and decreasing SP for MW.

Contrasting this to the larger number of MDP steps used by Shipman and Coetzee (2019), the size of the control problem to be solved significantly influences operational data requirement. The size is influenced by whether a single set of physical properties describe the plant and the number of components defining the state $\boldsymbol{S}$ observed by the agent. The sampling period ΔT selection needs to be viewed together with the typical time duration of a process's dynamic response. In general, it is known that deep neural networks require large amounts of data despite their improved efficiency over linear parametric function approximators. Table 29 illustrates that each RL design much be viewed together with the process to which it is applied, as these influence operational data requirement significantly.

6.6 Comparison of Studied RL Agents

Table 30 compares the RL agents studied to summarise the findings in this thesis and to aid in the understanding of the strengths and weaknesses of the RL agents.

Table 30: Comparison of studied tabular RL agents (SARSA and Q-learning) and One-Step Actor-Critic agent

	Tabular SARSA (and Q-learning)	**One-Step Actor-Critic**
Less than two parameters to tune for the specific process to be controlled?	Yes	No
Agent decides degree of determinism?	No	Yes
Can first guess of parameters be obtained off-policy?	Yes (Q-learning)	No (need algorithmic adjustments, see literature review – Chapter 3)
Progression of control performance measures for a fixed control problem	Large variance in progression, slow learning, sensitive to greedy action locations.	Monotonic progression of measures, variances in parameters reduce as agent becomes more confident in its action selections.
Type of process (criteria for application)	Sufficient instrumentation with challenging dynamics; self-regulatory SISO problems with slowly changing closed-loop dynamics; processes with quantitatively accurate models	Processes with greater uncertainty in measurements; processes with quantitatively accurate models; processes where SARSA is appropriate

	Tabular SARSA (and Q-learning)	**One-Step Actor-Critic**
Key limitations	Limit to robustness afforded by state-action space discretization; large operational data requirement; potential coverage limitations; stability challenges when using Q-learning with function approximation; potential for excessive final element adjustment; limited handles to enable safe exploration; noisy updates to parameters; degree of determinism in the policy π must be selected manually	Large operational data requirement; lower interpretability; challenging tuning; limited handles to enable safe exploration; needs algorithmic adjustments for offline parameter estimation and training using parallel computing

6.7 Selection of Filter Time Constant for Case Study 3 (The Grinding Circuit Model)

The MFS adjustments shown for the SARSA agent with $\tau_f = 5\ h$ in Figure 37 illustrated that the agent can access final element settings intermediate to the discrete actions made available to it since the selections are filtered using Equation [111] before being applied to the plant. The control problem is simplified as the agent does not have to learn the individual intermediate actions which are in reality infinite in number. This results in the control scheme studied (Section 5.1) being sensible for a feasibility study.

Figure 42 was generated for the SARSA agent with $\tau_f = 0.5\ h$ from Section 6.4.4. The blue squares are the unfiltered actions, while the black dots are filtered. Some intermediate actions are realised through filtering, but drastic and rapid changes in MV are caused, there are many intermediate actions that are not accessed at all, and the use of a filter is now questionable for the studied control scheme of Section 5.1. The filter is not used in the way that is described by Marlin (2000) and with different objectives – the best value for τ_f would depend on the number of actions available. Since only a few discrete action selections were made available in each of the simulations conducted, large values of τ_f were appropriate.

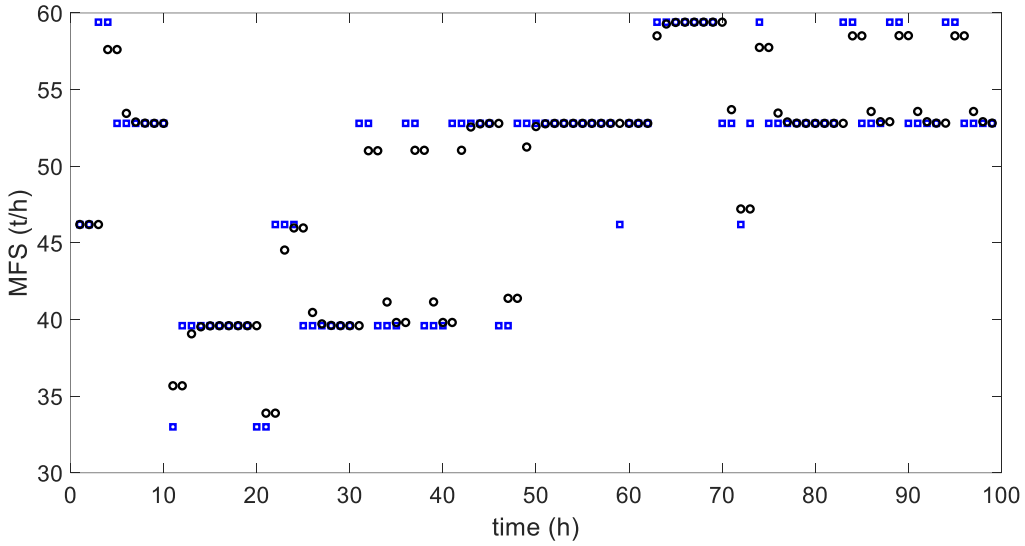


Figure 42: MV behaviour for SARSA with $\tau_f = 0.5\ h$ corresponding to results reported in Section 6.4.4 (blue squares are the unfiltered actions)

The MV behaviours for $\tau_f = 5\ h$ (Figure 37) and $\tau_f = 0.5\ h$ (Figure 42) also illustrate that the two SARSA agents approximated the optimal policy differently – the agent therefore adjusts its action selections based on the filter time constant τ_f. If we make τ_f larger, the agent reduces the frequencies with which it selects different actions, and increases the magnitudes of the changes between successive action selections.

RL agents start learning with a random policy. For SARSA, a deterministic or stochastic policy is selected by the designer. In contrast, an Actor-Critic agent will also start off randomly, but it then adjusts the degree of determinism in its decision making as appropriate by driving the preference function $h(\boldsymbol{S}, A, \boldsymbol{\theta})$ towards larger values in appropriate regions of the state-action space.

Let us first consider what happens when we filter the action selections of a SARSA agent that is only starting to learn. Figure 43 for $\tau_f = 5$ hours was generated by training a SARSA agent for one episode, and plotting the unfiltered (blue squares) and filtered (black circles) action selections. For coarse action space discretization (only a few actions available), the filtered actions do not jump around as much as the blue squares. Between 20 and 40 hours, the action is maintained constant since certain actions are considered greedy during the first episode of training. These learning properties will not be achieved for $\tau_f = 0.5\ h$.

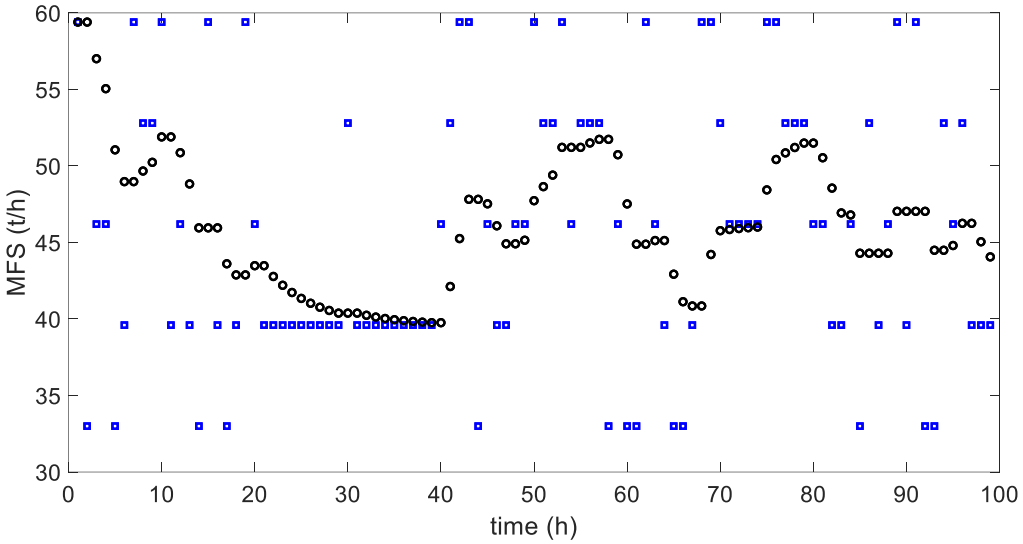


Figure 43: *MV* behaviour for SARSA with $\tau_f = 5\ h$ after training for only one episode on the PSE-MFS control loop (blue squares are the unfiltered actions)

Now we can consider what happens if we train an Actor-Critic agent with the two filter time constants based on Section 6.4.6. The agent will start off randomly, and through the preference function $h(\boldsymbol{S}, A, \boldsymbol{\theta})$ will *gradually* approach more deterministic decision making as it continues training.

If $\tau_f = 5$ hours, the agent can access the intermediate actions after filtering, and the actions applied to the plant do not jump around as much. This allows the Actor-Critic agent to "evolve" its policy during training. If we were to use $\tau_f = 0.5$ hours *with only a few discrete actions available*, the Actor-Critic agent would realise that the "safest" way of behaving with limited information is maintaining seemingly random decisions rather than becoming more confident in certain selections.

In literature, there are two main ways to apply an Actor-Critic agent to address this problem (Sutton and Barto, 2018). Firstly, we could provide *many* actions to the agent. Secondly, we could express the parameters of a pdf in terms of $\boldsymbol{\theta}$. State-of-the-art RL approaches to process control can perform much better than the results presented in Case Study 3. For this feasibility study it was of interest to see what happens if one trains a One-Step Actor-Critic agent with a binary reward function, and *allow it to access many actions through the low pass filter*. This inherently requires a large τ_f.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

Recall from Chapter 1 that this thesis aims to provide an account of the feasibility, nuances, and challenges of elementary RL-based process control through simulation-based study. An RL-based control scheme that allows one to study interpretable and simple RL control designs was used to conduct the study. The controllers studied did not outperform PI/PID controllers where these may successfully be applied.

Process control is a field where a notable contrast exists between theoretical controllers and those that may be applied safely and economically in industry. RL-based process control is still a fairly nascent technology that is a good example of this. The SARSA, Q-learning, and One-Step Actor-Critic agents were studied. These algorithms form the basis for more advanced, state-of-the-art approaches.

7.1 Answering the Key Questions

What are the limits of tabular Q-learning and tabular SARSA in RL-based process control?

The results of training a tabular SARSA agent with a binary reward function on the self-regulatory water tank model (Case Study 1 – Section 6.2) and the mixing tank model (example problem in Appendix A) showed that it is *theoretically* possible to train a tabular agent so that the resulting policy is capable of achieving $|E(T)| < \beta$ for a considerable region of the state-action space.

Section 6.2 indicated that, when synthetic data provides sufficient quantitative accuracy to allow the use of a SARSA agent during training, tabular SARSA-based control may be reasonably applied to a self-regulatory process for which the process gain does not vary depending on the location within the operating window. While self-regulatory processes pose a sufficiently common control problem, the availability of a mathematical model suitable for the adequate pretraining of a SARSA agent for industrial application is not considered likely. Further, regions of the state-action space will be encountered where exploration is not yet sufficient, resulting in erratic control behaviour in those regions.

This necessitated investigation of Q-learning and the presence of potential coverage limitations in the resulting pre-trained policy were found (Section 6.2.2 and Section 6.4.5). The example control behaviour provided by the behavioural policy π_b may be used effectively in the specific region of the state-action space where examples are provided often. Heuristics are expected to aid in overcoming this problem (Section 6.2.2).

In contrast to the results for Case Study 1 (water tank, Section 6.2) and Case Study 2 (Van de Vusse reaction scheme, Section 6.3), Case Study 3 (grinding circuit, Section 6.4) delivered the most unsatisfactory control performance. This indicates that a simplistic RL-based control approach may be severely limited in its ability to deal with significant partial observability (Section 2.14.1 and 2.14.4) in a robust control problem of a process with non-linear dynamics. The approach to simulating such control

challenges was described in Section 4.3.3. This illustrates that limitation in the available instrumentation is a significant challenge.

Therefore, a tabular RL-based controller may be reasonably applied to *simulated* self-regulatory process control problems where partial observability is not a severe problem, but the application in industry is met with considerable practical impediments resulting from coverage limitations, robustness requirements, the need for exploration before sensible decisions can be made across the state-action space, and large operational data requirement.

What are the limits of One-Step Actor-Critic in RL-based process control?

In Chapter 2 and Section 5.6, it was seen that the One-Step Actor-Critic algorithm requires on-policy training accompanied with the application of serial computation. From Section 5.6 and the results of Section 6.4.6, the One-Step Actor-Critic algorithm does not have heuristics or rules that may be applied to tune $\alpha_{\boldsymbol{w}}$ for the critic or $\alpha_{\boldsymbol{\theta}}$ for the actor. An excessively small or excessively large value of $\alpha_{\boldsymbol{w}}$ may result in the critic being unable to adjust the magnitudes of the adjustments to the parameter vector $\boldsymbol{\theta}$ properly since the temporal difference in terms of state-value is multiplied with $\alpha_{\boldsymbol{\theta}}$ in the actor's update rule.

An excessively large value of $\alpha_{\boldsymbol{\theta}}$ may cause the preference function, $h(\boldsymbol{S}, A, \boldsymbol{\theta})$, to be driven quickly to excessively large values across certain regions of the state-action space during training. This may cause the policy π of the RL agent to behave deterministically when it is not yet appropriate. If $\alpha_{\boldsymbol{\theta}}$ is too small, the agent's ability to adapt to changes in its RL environment within a reasonable number of training steps is compromised. Recall from Section 5.5 that ensuring convergence through the application of Robbins-Monro convergence criteria will likely result in excessively slow learning rates.

While the progressions of control performance measures were found to be monotonic in Section 6.4.6, this is of little practical use. The speed of this progression for a fixed control problem cannot be predicted for an arbitrary RL environment, function approximator, and value of $\alpha_{\boldsymbol{\theta}}$. Therefore, the number of training steps required before the RL agent has reached its maximum state-action space coverage for the RL representation design used and hyperparameter selection for a specific application cannot be predicted beforehand. The operational data requirement for the training of the One-Step Actor Critic agent in Section 6.4.6 is too large to be feasible.

Can control problem complexity be practically reduced for an RL agent by simplifying the RL environment, e.g. by discretizing the state-action space?

Discretizing the state-action space for SARSA and Q-learning and the filtering of the RL agent's discrete action selections in the control scheme studied simplifies the control problem for the agent. Since all case studies considered have continuous operating windows, the best achievable control behaviour for a selected reward function requires decision making directly in the continuous state-action space. As illustrated in Section 6.7, this control scheme requires the use of a large filter time constant τ_f for the RL agent. The agent is capable of adjusting the frequencies and magnitudes of its changes in the selected discrete actions

to accommodate the filter, but is still restricted to a predefined set of available actions. As a result, the dynamics of the low-pass filter and the discretized state-action space likely result in constraining the RL agent to solving a problem for which the optimal policy π^* approximated is suboptimal to the optimal policy π^* that could have been learned in the continuous state-action space. Further, from Section 6.2.2, different RL environments emerge for off-policy and on-policy training owing to the absence of filtered historical data in practice.

As a result, it may be concluded that the simplification of the problem to be learned by the RL agent by simplifying the RL environment needs to be done within the context of other control theory. Chapter 3 showed that a common choice is utilising RL as a method of automatically tuning a PI or PID controller.

7.2 Addressing the Aim of the Project

Recall from Chapter 1 the aim of this thesis:

To evaluate the feasibility of applying elementary RL techniques to automatically determine the optimal control actions for process control systems.

The feasibilities of the SARSA, Q-learning, and One-Step Actor-Critic agents were evaluated using the criteria from Section 5.7. The control scheme studied, Section 5.1, simplifies the RL-based control problem targeted through the filtering of discrete action selections, sampling period ΔT selection, and the use of a binary reward function. Hence, the operational data requirement for the stabilisation of RL representation parameters is most likely an underestimation of the requirement when omitting the low-pass filter and using an elaborate reward function for the same sampling period ΔT. Recall from Section 6.5 that operational data requirement is not only influenced by the number of parameters to be learned, but also by the agent applied, the RL representation's design, and the specific RL environment to which it is applied.

7.2.1 Memory Usage and Time for Code Execution

Representative approximate wall times for the three case studies are provided in Table 31 since this is dependent on which other tasks are run if a regular computer were used for serial computation. In cases where large computational tasks have been parallelised and executed as described in Section 5.6, the corresponding entry using one worker was approximated.

Table 31: Approximate wall times for the training of RL agents as described in Chapter 5 and Chapter 6 (wall times projected to equivalent wall times for serial execution are indicated in italics)

			Approximate wall time[1]	
Case Study	**Algorithm**	**Sections**	**Serial (one worker)**	**Parallel (16 workers)**
1 (water tank)	SARSA	5.3.1 and 6.2.1	20 to 40 min	N/A
	Q-learning	5.4 and 6.2.2	20 to 40 min	N/A
2 (Van de Vusse reaction scheme)	SARSA (factorial experiment)	5.3.3 and 6.3.2	*5.7 hours/batch*	45 min/batch
3 (grinding circuit)	SARSA	5.3.5 and 6.4.4	*22.8 hours*	3 hours
	Q-learning	5.4 and 6.4.5	*7.6 days*	24 hours
	One-Step Actor-Critic	5.5 and 6.4.6	4 days	N/A

Simulations performed for SARSA and Q-learning agents using serial computation were not mapped to approximate wall times for parallel execution since scaling behaviour needs to be assessed for computational problems of similar magnitudes. For One-Step Actor-Critic, serial computation is required (Section 5.6).

For Case Study 1 (water tank), serial computations for SARSA and Q-learning do not produce wall times that would prevent the study of these methods on a regular computer – approximate wall times of 20 minutes to 40 minutes recorded in Table 31. Similarly, for Case Study 2 (the Van de Vusse reaction scheme), wall times would not prevent the training of a SARSA agent on a regular computer. For the factorial experiment used to investigate discretization fineness (Sections 5.3.3 and 6.3.2), the large number of time steps allowed for training per batch (defined in Section 5.3.3) requires the use of high-performance computing. From Table 31, each parallel training batch maps to a wall time for serial execution in the range of approximately 5.7 hours. The 2^4 factorial experiment required 32 batches in total, clearly showing that execution on a regular computer would not be reasonable.

The projected wall times for serial training of SARSA for the PSE-MFS control loop (Section 5.3.5 and Section 6.4.4) show that the use of a regular computer is in theory possible. This result should be viewed together with how the memory usage scales for SARSA. For Q-learning, the PID execution time of $0.01\ h$

[1] For wall times in italics: These projected values were calculated using a speed-up conversion factor derived from the linear portion of Figure 15 in Section 5.6.

for the behavioural policy π_b resulted a large number of function evaluations during training, thereby causing wall times that may not be reasonable for a regular computer. Again, scaling of memory usage must be kept in mind. The wall time of four days for the One-Step Actor-Critic algorithm are much larger than the projected wall times for serial execution obtained for SARSA. This shows that it is not only the number of episodes and the numerical complexity of the RL environment that contribute to the wall time, but also the scaling behaviour of the agent in terms of time.

For a SARSA or Q-learning agent with N discrete intervals in each dimension of a state-action space where $\boldsymbol{S} \in \mathbb{R}^3$ and $A \in \mathbb{R}^1$, each update of the approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S}, A)$ requires working with one scalar entry at each instance of T. This $\mathcal{O}(1)$ complexity in time is applicable irrespective of the number of discretized states and actions as well as the number of components comprising each. In terms of memory usage, however, $\mathcal{O}(N^4)$ complexity is obtained for these agents (Sutton and Barto, 2018). This places a limit on the number of episodes and the fineness of discretization that can be studied on a regular computer.

Consider the case where there are N elements in both the critic and actor parameter vectors of a One-Step Actor-Critic agent. At each instance of T, $2N$ elements need to be evaluated during the updates to the critic and actor parameters. Therefore, the complexity of the calculation scales according to $\mathcal{O}(2N)$ (Sutton and Barto, 2018). This contributes to the relatively large wall time observed for the One-Step Actor-Critic algorithm in Table 31, while storage of the parameter vectors in memory is typically not a problem.

The simulated operational data requirements for the training of the RL agents in the different case studies are presented in Figure 44 and Figure 45. In these figures, the number of steps reported for SARSA and Q-learning in Case Study 1 (water tank) and Case Study 3 (grinding circuit), respectively, were truncated to the point where the approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S}, A)$ stabilised. To do so, the learning curves given for SARSA applied to the water tank model in Figure 16 and Figure 21 of Section 6.2.1 as well as the control performance measure progression for Q-learning applied to the grinding circuit model, Figure 39 of Section 6.4.5, were consulted. In Case Study 3, $\tilde{Q}_\pi(\boldsymbol{S}, A)$ did not stabilise after initial improvement during the number of training episodes allowed for SARSA. The small sampling period of the behavioural policy π_b resulted in many time steps for the Q-learning agent in Case Study 3, but is associated with an operational data requirement similar to that of SARSA in Case Study 1. For the One-Step Actor-Critic agent, "Improvement end" refers to the end of the initial improvement in control performance measures observed in Figure 40 of Section 6.4.6.

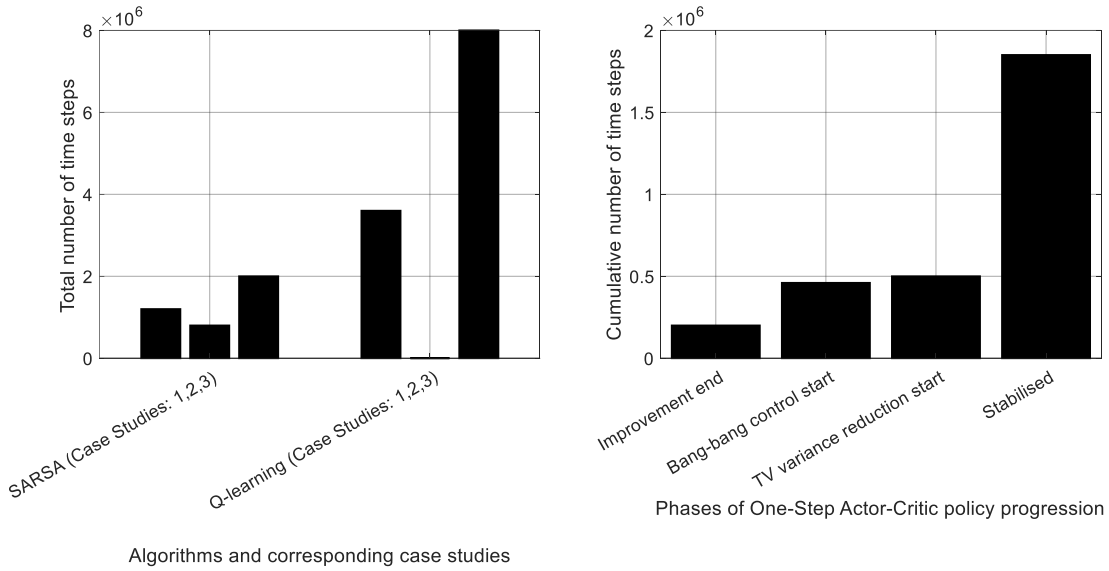


Figure 44: Total numbers of discrete time steps for training SARSA, Q-learning (not applied to Case Study 2), and One-Step Actor-Critic RL (only applied to Case Study 3): sampling period ΔT was $1\ min$, $43.75\ s$, and $1\ h$ for Case Studies 1 (water tank with $\beta = 0.2\ m$), 2 (Van de Vusse reaction scheme), and 3 (grinding circuit) with the exception of $\Delta T = 0.01\ h$ for Q-learning applied to Case Study 3

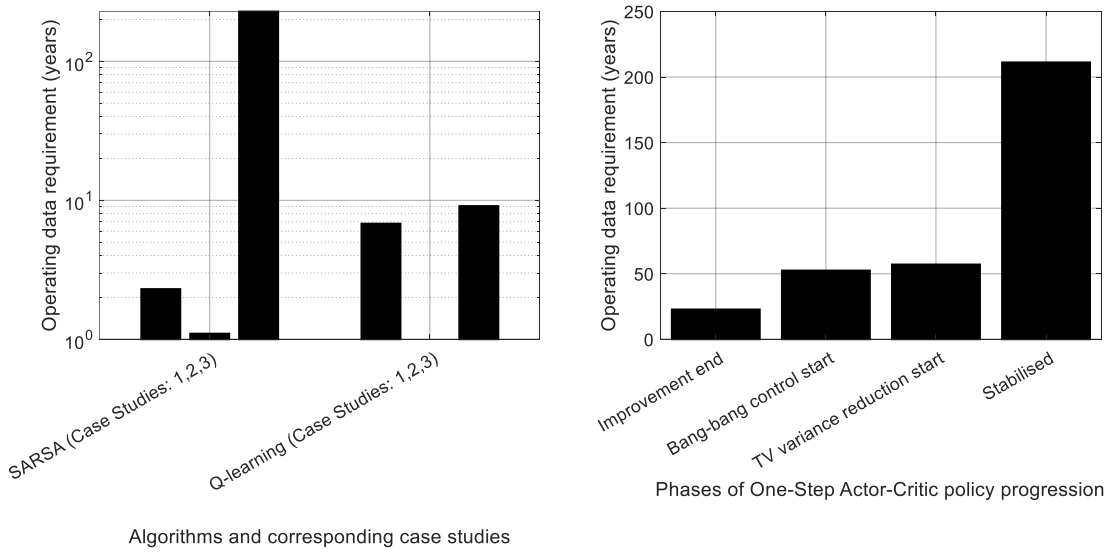


Figure 45: Operational data requirements for training SARSA, Q-learning (not applied to Case Study 2), and One-Step Actor-Critic (only applied to Case Study 3): sampling period ΔT was $1\ min$, $43.75\ s$, and $1\ h$ for Case Studies 1 (water tank with $\beta = 0.2\ m$), 2 (Van de Vusse reaction scheme), and 3 (grinding circuit) with the exception of $\Delta T = 0.01\ h$ for Q-learning applied to Case Study 3

At first glance, the operational data requirements for the SARSA agents trained for the first two case studies may seem reasonable. The requirements are, however, still deemed too large when considering that the

quality of a first guess policy obtained through Q-learning is dependent on the likely limited state-action space coverage allowed for by the behavioural policy π_b (Section 6.2.2).

All the other entries for on-policy RL agents (SARSA and One-Step Actor-Critic) in Figure 44 and Figure 45 represent physically unrealisable operational data requirements. Case study 3 (grinding circuit) represents the most practically representative simulated control problem for the studied RL agents. The significant increase in the operational data requirements accompanying this problem indicate that the increase in problem complexity owing to the presence of partial observability without the incorporation of historical information in the observed state $\boldsymbol{S}$ has a marked impact on how quickly an RL agent's parameters stabilise.

It may be concluded that the operational data requirements for the elementary RL algorithms studied are too great to be feasible for industrial use. The algorithms may be feasible in the case of certain laboratory scale RL experiments, especially if a quantitatively accurate model for the process exists (Syafiie et al., 2008; Brujeni et al., 2010; Ramanathan et al., 2018) – Chapter 3.

7.2.2 Control Performance

Independent of the RL agent studied, if state-action coordinates are encountered that are not associated with sufficient coverage, erratic control behaviour will result. The mixing tank example in Appendix A provides a clear graphical illustration of such a case where random action selections result. Different degrees of coverage can achieve any behaviour between the extremes of random actions and the best decision making that can be learned by the RL agent. The occurrence of failure modes and the degree of failure cannot readily be predicted beforehand irrespective of whether the most commonly encountered DV values are known.

The study of the grinding circuit model showed that 2 years' worth of synthetic data was sufficient to adapt the policy of the SARSA agent to compensate for a "worst-case" stiction scenario (Section 6.4.4). The One-Step Actor-Critic agent was not able to compensate for stiction within the same number of operational hours. This emphasizes that there exist practical challenges when applying One-Step Actor-Critic agents (Section 6.4.6). The typically low values of $\alpha_{\boldsymbol{\theta}}$, and the benefits of operating in the region of a stochastic policy with optimal control statistic measures when using a binary reward function prevent quickly compensating for significant stiction.

An Actor-Critic agent requires a large number of actions to enable the agent to become more confident in the appropriate action selections. More thought also needs to put into the design of a suitable reward function as the state space $\mathcal{S}$ is inherently modelled as continuous. Feedback loop dynamics will typically change very slowly which could potentially allow sensible application of One-Step Actor-Critic agents to compensate for such effects.

In classical control, the final value theorem provides a method that may be applied to illustrate analytically that the integral mode of PI/PID control ensures that zero steady-state offset is eventually achieved in

response to SP or DV changes. In contrast, closed-loop tracking in RL is dependent on the designer's knowledge of the RL environment.

Specifically, the reward function and defined observed state must enable closed-loop tracking by fully constraining the RL environment. Since measurements are often unavailable, decision making must allow for the presence of a stochastic optimal policy π^* that needs to be approximated and is of learned independent of the action-value function. If an RL agent were allowed to directly adjust the final element, there would be no limit to the speed with which the RL agent responds to a CV measurement once it is available.

Assuming that a sufficient range in action selections are available to the RL agent, it would be able to adapt automatically to any noise present in the CV measurements provided to it. In the case of a One-Step Actor-Critic agent, this is subject to the limitations discussed earlier in this section regarding adaptation to altered closed-loop dynamics. For SARSA, if the signal to noise ratio is sufficiently large, the mapping of plant measurements to observed state $\boldsymbol{S}$ will not be affected.

7.2.3 Ease of RL Agent Tuning

For the specific cases of tabular SARSA and tabular Q-learning agents, the step size hyperparameter α and discount factor γ can be selected independently of the instance of discretization and the RL environment studied – Sections 5.3.4 and 6.3.3. For One-Step Actor-Critic, the step-size hyperparameters $\alpha_{\boldsymbol{\theta}}$ and $\alpha_{\boldsymbol{w}}$ need to be selected using a trial-and-error procedure.

Irrespective of the RL agent studied, design of the reward function may hold significant complexity, especially when modelling the state space $\mathcal{S}$ as a continuum, as is the case with One-Step Actor-Critic. This is because the reward function is the only available handle for the communication of control objectives in the context of model-free RL.

In contrast to elementary RL-based controllers, classical control has well-established tuning methodologies applicable to different processes (Marlin, 2000; Skogestad and Poslethwaite, 2005).

7.2.4 Safety of Exploration

If an RL agent is trained on-policy without sufficient coverage obtained beforehand using historical data, any of the available states may be encountered. This was shown by testing for the overflow condition in the mixing tank example of Appendix A, but is also evident from the theory presented in Chapter 2 and the necessity of boundary states when applying state-action space discretization for SARSA and Q-learning (Chapter 5). A model of the RL environment would be required to promote safe exploration in certain regions of the process operating window.

7.2.5 Whether Controller Operation is Interpretable to a Human

For a tabular RL representation, irrespective of the number of components comprising $\boldsymbol{S}$ and A, RL agent behaviour may easily be conceptualised and predicted to a large degree of accuracy when presented with $\tilde{Q}_{\pi}(\boldsymbol{S}, A)$ and the current observed state $\boldsymbol{S}$. In contrast, a linear basis function RL representation, as was used for the One-Step Actor-Critic agent, is difficult to conceptualize since each learned parameter is associated with a basis function that has a perceptive field. The output for a particular measurement in the infinite number of possible instances of $\boldsymbol{S}$ may potentially be calculated manually to a reasonable accuracy, but is not considered readily interpretable since the contours of the optimal policy π^* cannot be conceptualised more than one state or action component is applicable to the problem under consideration.

7.2.6 Process Modelling Requirement

Model-free RL does not require a model of the RL environment to approximate the optimal policy π^* and such techniques learn purely through experience. This allows a better approximation to the optimal policy π^* to be obtained since the approximation is not dependent on the accuracy of a model of the RL environment (Moerland et al., 2021). No linear model of the process is required either.

7.2.7 Conclusion Regarding Feasibility

From the evaluation of the feasibility criteria in Sections 7.2.1 through 7.2.6, it is concluded that elementary RL is not feasible for determining the optimal actions in industrial process control. This confirms the appropriateness of the prevalent RL-based control research directions observed in state-of-the-art methods – Chapter 3.

7.3 Addressing the Objectives

Recall from Section 1.2 that the objectives included synthesis of an RL-based SISO control methodology for feasibility study, validation of the control methodology, investigation of discretization coarseness and hyperparameter tuning, and evaluation of the feasibility of tabular SARSA, tabular Q-learning, and One-Step Actor-Critic RL agents in terms of feasibility criteria applied in the context of a ball mill grinding circuit simulation.

A non-linear control synthesis with a binary reward function was proposed for feasibility study and understanding the learning behaviours of RL agents irrespective of their relative complexities (Section 5.1). The RL methodology was validated qualitatively by applying a tabular SARSA agent to a self-regulatory water tank control problem (Case Study 1) – Section 6.2.1. Further, control of the Van de Vusse reaction scheme (Case Study 2) was studied to this end (Section 6.3.1).

The IAE responses in a 2^4 factorial experiment applied to Case Study 2 (the Van de Vusse reaction scheme) indicated that the error signal $E(T)$ may be discretized coarsely to reduce IAE at the expense of more pronounced final element adjustments. The responses indicated little benefit of discretizing both SP and DV finely. Section 6.3.2 therefore showed that a finely discretized state-action space needs to enable

effective leveraging of the reward function used. If this is not done, an excessive number of parameters may be defined in the RL representation, thereby likely increasing the operational data requirement associated with RL agent training.

RL hyperparameters were investigated (Sections 5.3.4 and 6.3.3). RL-based control was applied to the PSE-MFS control loop of a qualitatively accurate grinding circuit model (Sections 5.3.5, 5.4, 5.5, and 6.4). Based on the lessons learnt by studying the grinding circuit model, practical limitations could be incorporated in the comparison of the RL agents (Section 6.6). The set of feasibility criteria identified in Section 5.7 were used to assess the feasibilities of the studied RL agents (Section 7.2).

7.4 Contributions to the Field

This thesis contributes to the field of RL-based process control by critically assessing the feasibility of elementary model-free RL-based control and identifying impediments to the industrial adoption of RL that originate at the foundations of the field. A simplified RL environment was used to understand the behaviour of the elementary RL agents applied to simulated control problems (Section 5.1). Further, the thesis illustrates nuances of designing and implementing RL-based control solutions, some of its practical limitations (Sections 6.2, 6.3, and 6.4), and provides stakeholders with a reference to the field of RL-based process control that focuses extensively on the elementary principles of model-free RL-based control.

7.5 Code Used for the Feasibility Study

MATLAB code used during the feasibility study is available at:

https://github.com/Stellenbosch-University-Process-Eng/Feasibility-of-elementary-RL-for-process-control

Code that may be used to simulate the Van de Vusse reaction scheme model (Section 4.2) and the grinding circuit model (Section 4.3) are provided. Further, the code used to apply SARSA and Q-learning to the water tank model (Section 6.2), to investigate SARSA application to the Van de Vusse reaction scheme (Section 6.3), and to apply PID control, SARSA, and One-Step Actor-Critic agents to the grinding circuit model (Section 6.4) are also provided.

7.6 Practical Limitations of the RL Methodology

While the control scheme studied (Section 5.1) was suitable for a feasibility study, a few practical limitations need to be highlighted.

Safe exploration is most naturally investigated in the context of model-based RL (see, for example, (Berkenkamp et al., 2017)). Model-free RL only provides the reward function and parameterization of the learned policy π as handles to maintain CV within a specified region (omitting from consideration non-RL interventions in the control approach). Any practical approach to safe exploration requires the designer to

be very confident that the uncertainty arising from limited instrumentation could be accounted for reliably, sustainably, and within a reasonable operational data requirement.

For a fixed value of τ_f, excessive final element adjustment arises as a result of leveraging only a few actions and not penalizing the agent for excessive MV adjustments through the reward function design. Increasing τ_f in the studied control scheme does reduce the extent of excessive final element adjustment, but is limited depending on how coarsely the action space is discretized. In addition to difficulties with coverage in tabular SARSA and Q-learning, the studied method requires the use of different RL environments for Q-learning and SARSA (Section 6.2.2). Recall from Section 5.7 that including the low-pass filter also degrades the optimal policy π^* targeted in the applications considered.

RL-based control is focused on performance optimization by the RL agent's search for greater cumulative rewards during training. Process control is, however, oriented towards ensuring the stability of a controller, which would require consideration of the learning process rather than the achievable performance alone in the case of an RL-based control approach. No clear stability criterion currently exists for RL-based control (Buşoniu et al., 2018).

While an RL agent would likely adapt its action selections to account for dead time in a process, the presence of large dead times may have a significant influence on operational data requirements before a policy π is reached that would ensure stability when closing the loop using the RL agent. When insufficient state components can be measured to fully describe the RL environment, historical information may also need to be incorporated to ensure that the problem is adequately approximated by the assumption of a Markovian RL environment (Moerland et al., 2021).

7.7 Recommendations

In the author's opinion, RL should not be approached as a completely separate paradigm to process control, but rather as a way to establish a complementary relationship between computational thinking and well-established control approaches. Based on the RL methodology of Chapter 5 and the results presented in Chapter 6, this section describes shortly recommendations for future work that aims to further investigate elementary RL principles with the ideal result being that such work forms a point of departure to better understand aspects of state-of-the-art approaches.

7.7.1 Quantification of Action-Value Function Coverage in Tabular RL Methods

When applying any RL-based control approach, there is a certain fraction of the state-action space for which the agent has learned optimal control behaviour provided that sufficient training has occurred. This fraction is unknown when working with RL agents which greatly increases practical difficulties in process control systems where erratic control behaviour must always be avoided.

From the results presented in Section 6.2 it is known that the state-action coordinates closest to $E(T) = 0$ have the largest number of visits during the training of an RL agent. Further, from Section 6.3.2 it is known

that increasing the discretization of the components comprising the observed state $\boldsymbol{S}$ increases the consistency with which controller performance lies within the interquartile range associated with the population distribution of IAEs, but may not necessarily significantly improve controller performance for a given reward function. Learning curve shapes may be difficult to observe if stochastic process inputs are used during RL agent training. The coverages required to achieve stable control is naturally also unknown.

This accentuates the need for an increased focus on quantifying the progression of training for process control applications in addition to the stability of the resulting controllers, as has been noted in literature (Buşoniu et al., 2018). The observations from Chapter 6 mentioned above may potentially aid in providing a point of departure to quantify coverage for tabular methods.

7.7.2 Investigating the Influence of Including Historical Information in the Observed State

From Section 2.14.1, the sequential decision making modelled using an MDP in RL relies on the assumption that the presently observed state $\boldsymbol{S}$ and the action taken $\boldsymbol{A}$ fully constrain the state transition that occurs between time steps T and $T+1$. Owing to the prevalence partial observability in process control systems, this assumption is unlikely to hold true unless historical information is included in the observed state $\boldsymbol{S}$ (Conradie, 2000; Ma et al., 2019).

Chapter 3 and Chapter 6 have shown that this does not prevent successful application of RL-based control to simulated process control systems. To potentially enhance the training processes of RL-based controllers, the effects of including historical data points must be investigated. The MDP assumption (Section 2.14.1) will likely hold when sufficient historical information is included in $\boldsymbol{S}$. Two approaches are to modify the definition of $\boldsymbol{S}$ and to incorporate eligibility traces that scale the different adjustments to the RL representation's parameters based on visitation frequencies within the state-action space (Sutton and Barto, 2018). The simulation results generated by Ma et al. (2019), the operational data requirement observed for their work in Section 6.5, and the infeasible practical control behaviours observed in this study motivate this recommendation.

7.7.3 Elementary RL Agent Learning Behaviours for Simulated MIMO Control

In this thesis, each case study involved the application of an RL agent as the controller in one control loop. If more than one agent-based control loop were implemented, each RL agent would see the others as part of its RL environment. The RL environment for each agent would not be stationary (Sutton and Barto, 2018)

In RL, a curriculum refers to non-stationarity in the RL environment purposefully introduced by the investigator to allow behaviours to be learned that would not have been learned otherwise. While plantwide RL-based control is much further from successful application, the hypothesis of curricula being instantiated automatically through the interacting learning processes of more than one agent, as described by Leibo et al. (2019), may be worth investigating. Competition and exploration in the changing RL environments are likely to arise naturally between the different RL agents as each agent only wants to

maximize the cumulative rewards it receives. This may potentially contribute to the learning of different types of control actions for simple reward functions.

7.7.4 Development of a 'Mixture RL' Controller for Process Control Systems

The integration of system identification, classical process control and RL is a developing field of research. The application of model-free and model-based RL in a complementary fashion in this context may be referred to as "Mixture RL" (Moerland et al., 2021).

Future research could be aimed at developing an RL-based controller that combines global, model-free RL (provides better asymptotic performance) with forward-model-based RL principles for local decision making (planning) and state-action space coverage specifically for application to chemical and mineral processing plants. Such a controller should likely incorporate principles of classical control to enhance safety and feasibility. The global RL agent could aim to obtain a higher-level overview of the required policy π across the state-action space, while the planning method corrects local mistakes arising from coverage limitations.

A balance must be found between learning, planning, and data collection using an increased number of hyperparameters. To do so, data-efficient mixture approaches with local planning and global model-free function approximation to achieve reasonable control performance with significant interpretability, transfer, stability, and safety within operational data constraints may potentially be applied (Moerland et al., 2021). Integration of process control knowledge and principles from the model-based RL paradigm may form an ideal candidate for achieving more accessible and industrially feasible learning control without necessarily resorting to deep learning methods.

LITERATURE CITED

Atkins, A. R. *et al.* (1974) 'The control of milling circuits.', *Journal of The South African Institute of Mining and Metallurgy*, 74(11), pp. 388–395.

Baird, L. C. (1995) 'Residual algorithms: Reinforcement Learning with function approximation', in *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*. Morgan Kaufmann Publishers, Inc., pp. 30–37.

Barker, I. J. and Hulbert, D. G. (1983) 'Dynamic Behaviour in the Control of Milling Circuits', *IFAC Proceedings Volumes*, 16(15), pp. 139–152. doi: 10.1016/s1474-6670(17)64264-2.

Bellman, R. (1972) *Dynamic Programming*. New Jersey: Princeton University Press.

Berkenkamp, F. *et al.* (2017) 'Safe model-based reinforcement learning with stability guarantees', in *Proceedings of Neural Information Processing Systems (NIPS)*, pp. 908–918.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996) *Neuro-Dynamic Programming*. Belmont, M.A.: Athena Scientific.

Bianchi, R. A. C., Ribeiro, C. H. C. and Costa, A. H. R. (2012) 'Heuristically accelerated reinforcement learning: Theoretical and experimental results', *Frontiers in Artificial Intelligence and Applications*, 242(January), pp. 169–174. doi: 10.3233/978-1-61499-098-7-169.

Bishop, C. M. (2006) *Pattern Recognition and Machine Learning*. 1st edn. Edited by M. Jordan, J. Kleinberg, and B. Schölkopf. New York: Springer.

Brujeni, L. A., Lee, J. M. and Shah, S. L. (2010) 'Dynamic tuning of PI-controllers based on model-free reinforcement learning methods', *ICCAS 2010 - International Conference on Control, Automation and Systems*, pp. 453–458. doi: 10.1109/iccas.2010.5669655.

Buşoniu, L. *et al.* (2018) 'Reinforcement learning for control: Performance, stability, and deep approximators', *Annual Reviews in Control*, 46, pp. 8–28. doi: 10.1016/j.arcontrol.2018.09.005.

Chen, D. and Seborg, D. E. (2002) 'PI/PID controller design based on direct synthesis and disturbance rejection', *Industrial and Engineering Chemistry Research*, 41(19), pp. 4807–4822. doi: 10.1021/ie010756m.

Chen, H., Kremling, A. and Allgöwer, F. (1995) 'Nonlinear Predictive Control of a Benchmark CSTR', in *Proceedings of the 3rd European Control Conference ECC'95*. Rome, Italy, pp. 3247–3252.

Conradie, A. V. E. (2000) *Neurocontroller Development for Nonlinear Processes Using Evolutionary Reinforcement Learning*. Stellenbosch University.

Conradie, A. V. E. and Aldrich, C. (2001) 'Neurocontrol of a Ball Mill Grinding Circuit Using Evolutionary Reinforcement Learning', 14(10), pp. 1277–1294. doi: 10.1145/3205651.3207865.

Daniel, C. (1959) 'Use of Half-Normal Plots in Interpreting Factorial Two-Level Experiments', *Technometrics*, 1, pp. 311–342.

Deisenroth, M. P., Faisal, A. A. and Ong, C. S. (2020) *Mathematics for Machine Learning*. 1st edn. Edited by L. Cowles. Cambridge: Cambridge University Press. Available at: http://www.maa.org/external_archive/QL/pgs75_89.pdf.

Deisenroth, M. P. and Rasmussen, C. E. (2011) 'PILCO: A model-based and data-efficient approach to policy search', in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*. Bellevue, W.A., pp. 465–472.

Devore, J. L. (2017) *Probability and Statistics for Engineering and the Sciences*. 9th edn. Cengage Learning.

Dvoretzky, A. (1956) 'On Stochastic Approximation', in Neyman, J. (ed.) *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, pp. 39–55. doi: 10.2307/2980937.

Engel, Y. (2005) *Algorithms and Representations for Reinforcement Learning*. Senate of the Hebrew University. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.6809.

Engel, Y. (2011) 'Gaussian Process Reinforcement Learning', in Sammut, C. and Geoffrey, I. W. (eds) *Encyclopedia of Machine Learning*. Springer, pp. 439–445.

Fan, Y., Chen, L. and Wang, Y. (2018) 'Efficient Model-Free Reinforcement Learning Using Gaussian Process'. Available at: http://arxiv.org/abs/1812.04359.

Fasshauer, G. E. (2007) *Meshfree Approximation Methods with MATLAB*. Edited by J. Duan. New Jersey: World Scientific. Available at: http://bookzz.org/book/2472384/30c5e2%5Cnpapers3://publication/uuid/06E018CF-3ABE-48BF-B43A-0EAF18B299ED.

Felder, R. M., Rousseau, R. W. and Bullard, L. G. (2017) *Felder's Elementary Principles of Chemical Processes*. 4th edn. Hoboken: John Wiley & Sons, Inc.

Francis, B. A. and Wonham, W. M. (1976) 'The internal model principle of control theory', *Automatica*, 12(5), pp. 457–465. doi: 10.1016/0005-1098(76)90006-6.

Gilat, A. and Subramaniam, V. (2014) *Numerical Methods for Engineers and Scientists*. 3rd edn. Hoboken: Wiley.

Green, D. W. and Perry, R. H. (2008) *Perry's Chemical Engineers' Handbook*. 8th edn. New York: McGraw-Hill.

Hafner, R. and Riedmiller, M. (2011) 'Reinforcement learning in feedback control : Challenges and benchmarks from technical process control', *Machine Learning*, 84(1–2), pp. 137–169. doi: 10.1007/s10994-011-5235-x.

Hastie, T. *et al.* (2020) 'Surprises in High-Dimensional Ridgeless Least Squares Interpolation'. Available at: http://arxiv.org/abs/1903.08560.

Hermansson, A. W. and Syafiie, S. (2015) 'Model predictive control of pH neutralization processes: A review', *Control Engineering Practice*, 45, pp. 98–109. doi: 10.1016/j.conengprac.2015.09.005.

Hessel, M. *et al.* (2018) 'Rainbow: Combining improvements in deep reinforcement learning', in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3215–3222.

Hoskins, J. C. and Himmelblau, D. M. (1992) 'Process control via artificial neural networks and reinforcement learning', *Computers and Chemical Engineering*, 16(4), pp. 241–251. doi: 10.1016/0098-1354(92)80045-B.

Hunter, J. K. and Nachtergaele, B. (2005) *Applied Analysis*. New Jersey: World Scientific.

Jaakkola, T., Jordan, M. I. and Singh, S. P. (1994) 'On the Convergence of Stochastic Iterative Dynamic Programming Algorithms', *Neural Computation*, 6, pp. 1185–1201. doi: 10.1007/978-1-4842-3721-2_2.

James, G. *et al.* (2013) *Springer Texts in Statistics: An Introduction to Statistical Learning - with Applications in R*. Edited by G. Casella, S. Fienberg, and I. Olkin. Springer.

Lee, J. M. (2004) *A Study on Architecture, Algorithms, and Applications of Approximate Dynamic Programming Based Approach to Optimal Control*. Georgia Institute of Technology.

Lee, J. M. and Lee, J. H. (2005) 'Approximate dynamic programming-based approaches for input-output data-driven control of nonlinear processes', *Automatica*, 41(7), pp. 1281–1288. doi: 10.1016/j.automatica.2005.02.006.

Leibo, J. Z. *et al.* (2019) 'Autocurricula and the Emergence of Innovation from Social Interaction: A Manifesto for Multi-Agent Intelligence Research', (2016). Available at: http://arxiv.org/abs/1903.00742.

Lillicrap, T. P. *et al.* (2016) 'Continuous control with deep reinforcement learning', in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Lin, C. and Snyder, L. (2008) *Principles of Parallel Programming*. 1st edn. Edited by M. Hirsch et al. Boston: Addison-Wesley. doi: 10.1201/9780429439780-9.

Ma, Y. *et al.* (2019) 'Continuous control of a polymerization system with deep reinforcement learning', *Journal of Process Control*, 75, pp. 40–47. doi: 10.1016/j.jprocont.2018.11.004.

Marlin, T. E. (2000) *Process Control: Designing Processes and Control Systems for Dynamic Performance*. 2nd edn. McGraw-Hill.

McAvoy, T. J., Hsu, E. and Lowenthal, S. (1972) 'Dynamics of pH in Controlled Stirred Tank Reactor', *Industrial and Engineering Chemistry Process Design and Development*, 11(1), pp. 68–70. doi: 10.1021/i260041a013.

Mnih, V. *et al.* (2015) 'Human-level control through deep reinforcement learning', *Nature*, 518(7540), pp. 529–533. doi: 10.1038/nature14236.

Mnih, V. *et al.* (2016) 'Asynchronous Methods for Deep Reinforcement Learning', in *Proceedings of the 33rd International Conference on Machine Learning*. New York: JMLR W&CP.

Moerland, T. M., Broekens, J. and Jonker, C. M. (2021) 'Model-based reinforcement learning: A survey', *arXiv:2006.167v3[cs.LG]*.

Mongillo, M. (2011) 'Choosing Basis Functions and Shape Parameters for Radial Basis Function Methods', *SIAM Undergraduate Research Online*, 4, pp. 190–209. doi: 10.1137/11s010840.

Montgomery, D. C. (2013) *Design and Analysis of Experiments*. 8th edn. New Jersey: John Wiley & Sons, Inc.

Nian, R., Liu, J. and Huang, B. (2020) 'A review On reinforcement learning: Introduction and applications in industrial process control', *Computers and Chemical Engineering*, 139, p. 106886. doi: 10.1016/j.compchemeng.2020.106886.

Olsson, H. and Åström, K. J. (2001) 'Friction generated limit cycles', *IEEE Transactions on Control Systems Technology*, 9(4), pp. 629–636. doi: 10.1109/87.930974.

Poole, D. L. and Mackworth, A. K. (2010) *Artificial Intelligence: Foundations of Computational Agents*. 1st edn. Cambridge: Cambridge University Press.

Potapov, A. and Ali, M. K. (2003) 'Convergence of reinforcement learning algorithms and acceleration of learning', *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 67(2). doi: 10.1103/PhysRevE.67.026706.

Precup, D., Sutton, R. S. and Dasgupta, S. (2001) 'Off-policy temporal-difference learning with function approximation', in *Proceedings of the 18th International Conference on Machine Learning*. Available at: http://www.incompleteideas.net/sutton/papers/PSD-01-retypeset.pdf.

Rajamani, R.K. and Herbst, J. A. (1991) 'Optimal control of a ball mill grinding circuit-II. Feedback and optimal control', *Chemical Engineering Science*, 46(3), pp. 871–879. doi: 10.1016/0009-2509(91)80194-4.

Rajamani, R. K. and Herbst, J. A. (1991) 'Optimal Control of A Ball Mill Grinding Circuit - I. Grinding Circuit Modeling and Dynamic Simulation', *Chemical Engineering Science*, 46(3), pp. 861–870.

Ramanathan, P., Mangla, K. K. and Satpathy, S. (2018) 'Smart controller for conical tank system using reinforcement learning algorithm', *Measurement: Journal of the International Measurement Confederation*, 116, pp. 422–428. doi: 10.1016/j.measurement.2017.11.007.

Rippa, S. (1999) 'An alternative procedure for selecting a good value for the parameter c in RBF-interpolation', *Advances in Computational Mathematics*, 11, pp. 193–210. doi: 10.1007/s10444-010-9146-3.

Robbins, H. and Monro, S. (1951) 'A Stochastic Approximation Method', *Annals of Mathematical Statistics*, 22, pp. 400–407.

Le Roux, J. D. *et al.* (2013) 'Analysis and validation of a run-of-mine ore grinding mill circuit model for process control', *Minerals Engineering*, 43–44, pp. 121–134. doi: 10.1016/j.mineng.2012.10.009.

Shin, J. *et al.* (2019) 'Reinforcement Learning – Overview of recent progress and implications for process control', *Computers and Chemical Engineering*, 127, pp. 282–294. doi:

10.1016/j.compchemeng.2019.05.029.

Shipman, W. J. and Coetzee, L. C. (2019) 'Reinforcement Learning and Deep Neural Networks for PI Controller Tuning', *IFAC-PapersOnLine*, 52(14), pp. 111–116. doi: 10.1016/j.ifacol.2019.09.173.

Shoukat Choudhury, M. A. A., Thornhill, N. F. and Shah, S. L. (2005) 'Modelling valve stiction', *Control Engineering Practice*, 13(5), pp. 641–658. doi: 10.1016/j.conengprac.2004.05.005.

Simon, J. L. (1997) *Resampling: The new Statistics*. Resampling Stats.

Skogestad, S. and Poslethwaite, I. (2005) *Multivariable Feedback Control: Analysis and Design*. 2nd edn. Chichester: John Wiley & Sons, Inc.

Sutton, R. S. and Barto, A. G. (2018) *Reinforcement Learning: An Introduction*. 2nd edn. Cambridge, M.A.: The MIT Press.

Syafiie, S., Tadeo, F. and Martinez, E. (2008) 'Model-Free Learning Control of Chemical Processes', *Reinforcement Learning*. doi: 10.5772/5287.

Szepesvári, C. (2010) 'Algorithms for reinforcement learning', in *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, pp. 1–89. doi: 10.2200/S00268ED1V01Y201005AIM009.

Tan, M. (1993) 'Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents', in *Machine Learning: Proceedings of the Tenth International Conference*. San Mateo, C.A.: Morgan Kaufmann Publishers, Inc., pp. 330–337.

Theodoridis, S. (2020) *Machine Learning: A Bayesian and Optimization Perspective*. 2nd edn. London: Elsevier.

Wakefield, B. J. *et al.* (2018) 'Monitoring of a simulated milling circuit: Fault diagnosis and economic impact', *Minerals Engineering*, 120, pp. 132–151. doi: 10.1016/j.mineng.2018.02.007.

Watkins, C. J. C. H. (1989) *Learning from delayed rewards*. King's College. doi: 10.1016/0921-8890(95)00026-C.

Whitehead, S. D. and Lin, L. (1995) 'Reinforcement learning of non-Markov decision processes', *Artificial Intelligence*, 73, pp. 271–306.

Wills, B. A. and Finch, J. A. (2016) *Wills' Mineral Processing Technology - An Introduction to the Practical Aspects of Ore Treatment and Mineral Recovery*. 8th edn. Amsterdam: Elsevier.

Wright, R. A. and Kravaris, C. (2001) 'On-line identification and nonlinear control of an industrial pH process', *Journal of Process Control*, 11(4), pp. 361–374. doi: 10.1016/S0959-1524(00)00003-2.

Zhang, Y. *et al.* (2021) 'A Survey on Neural Network Interpretability', *IEEE Transactions on Emerging Topics in Computational Intelligence*. Available at: https://www.acm.org/media-center/2019/march/turing-award-2018 (Accessed: 7 September 2021).

APPENDIX A – WORKED EXAMPLE: LEVEL CONTROL OF A SELF-REGULATORY MIXING TANK

This appendix serves as supplementary material to Chapter 2 which, on its own, may be challenging to work through owing to the abstract nature of the concepts underlying RL-based control. To use this appendix optimally, the author proposes referring to it as both a practical example after reading Section 2.3 and as a method of making the theory more tangible by reconciling theoretical understanding with application after reading Chapter 2. The code used to generate the results presented in this appendix is available online at:

https://github.com/Stellenbosch-University-Process-Eng/Feasibility-of-elementary-RL-for-process-control

Figure 46 depicts a self-regulatory mixing tank that may be modelled as in Case Study 1 which involved the investigation of level control for a self-regulatory water tank. The model is replicated in Equation [118], where the cross-sectional area of the mixing tank studied is $A_{tank} = 1.8\,m^2$ and the initial steady state of the model is given in Table 32. The pressure loss parameter (ℓ) and the pressures P_1 and P_2 are not incorporated in the model structure since an appropriate value of c_v has been selected. Liquid density ρ_l does not appear in the model seeing it cancels out when writing the material balance for the mixing tank. In this example, we are concerned with keeping the level of process liquid in the mixer within the range of 1 m and 2 m in the presence of disturbances in the inlet flow rate F_{in}.

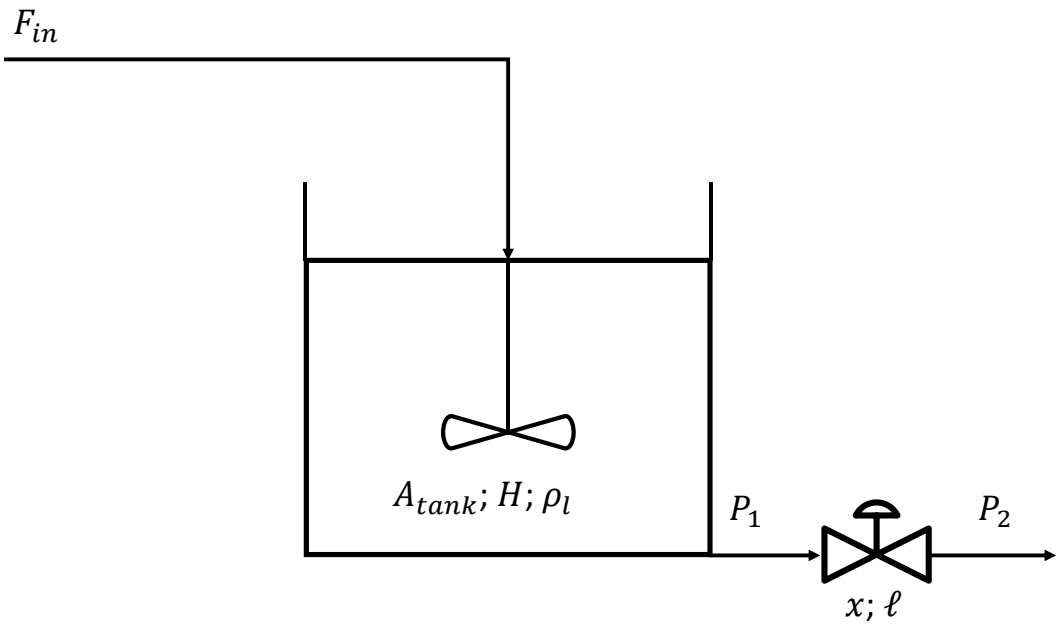


Figure 46: Illustration of the mixing tank which may be modelled using Equation [118]

$$A_{tank}\left(\frac{dH}{dt}\right) = (F_{in}) - c_v x\sqrt{H}\ where\ x \in [0,1] \quad [\ 118\ ]$$

Table 32: Steady state of the mixing tank model

State variable	Name of variable	Steady state	Unit
F_{in}	Inlet volumetric flow rate	31.67×10^{-4}	$\frac{m^3}{s}$
H	Height of the liquid in the tank	1.5	m
c_v	Valve discharge coefficient	0.01	$\frac{m^{2.5}}{s}$
x	Fraction valve opening	0.259	$-$

This control problem is of practical relevance since it may help attenuate feed composition disturbances to the unit receiving the process fluid from the outlet of the mixing tank. The tank therefore has to reduce the magnitudes and frequencies of such disturbances through its averaging effect, and we thus only need the liquid level within the tank to be sufficient to prevent overflow or drying of the tank (Marlin, 2000). This appendix applies a value-based agent known as the State-Action-Reward-State-Action (SARSA) algorithm, as described by Sutton and Barto (2018), to this control problem.

To approach this example problem, we can refer to the simplified time domain block diagram shown in Figure 47, where the effects of final element dynamics and sensor noise on the control law input and output are not considered. As a result, the simplification $u(t) = MV(t)$ may be made, where $u(t)$ is the control law output and $MV(t)$ is the final element value actually applied to the process. If we were to apply a PI/PID controller, it would obtain an error measurement $E(t)$ at each instance of time t and provide a desired final element adjustment $MV(t)$. This error is calculated by subtracting the controlled variable $CV(t)$ from the set point $SP(t)$, and the inlet flow rate affects the process as a disturbance variable $DV(t)$.

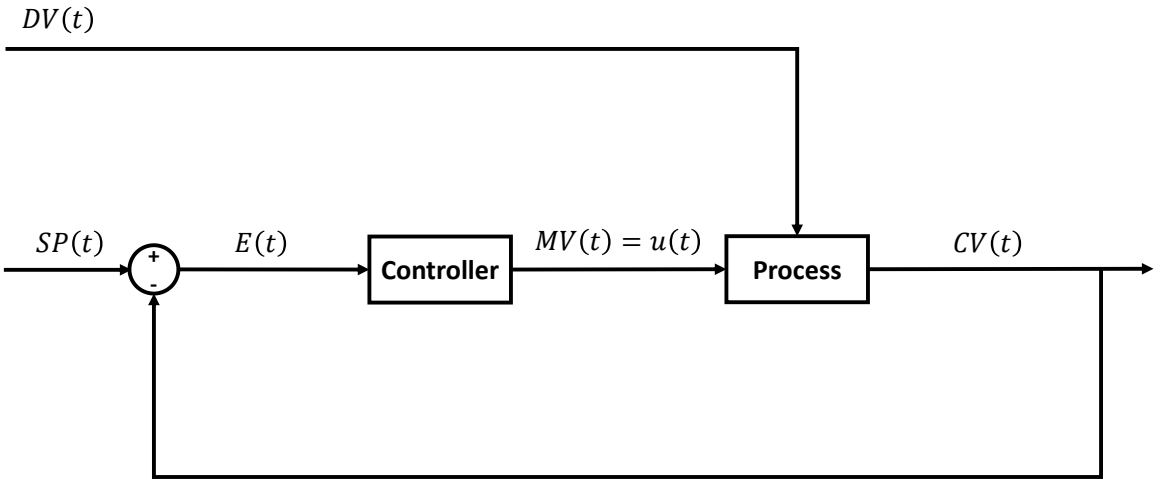


Figure 47: Simplified time domain block diagram of the feedback control system considered in the mixing tank example

The RL agent has a mathematical formulation and interacts with the RL environment (everything excluding the RL agent) during a number of discrete time steps, where each step is denoted by T. After each selection and application of an action, A, the RL environment transitions from state $\boldsymbol{S}$ observed at time step T to state $\boldsymbol{S}'$ observed at time step $T + 1$. As the RL environment leaves state $\boldsymbol{S}$, a reward R_T is provided to the agent as feedback to indicate how beneficial the transition from $\boldsymbol{S}$ to $\boldsymbol{S}'$ is. During interaction with the RL environment, the agent's update is thus calculated with the newest measurement available on the plant corresponding to time step $T + 1$ for the RL agent. A policy π is the decision making unit of the agent. The observed state $\boldsymbol{S}$ is its input and it provides as output the selected action A. Ultimately, we want to approximate the optimal policy π^* which is unique to the process being controlled assuming the process design and physical parameters remain unchanged (Sutton and Barto, 2018).

The sampled times relevant to SARSA are shown in Figure 48, where the first plant measurement used in the RL agent's update is denoted by t in continuous time and T in terms of the discrete time steps of the RL agent. Based on the speed of the dynamic response of the process, indicated by the time constant τ_p for the mixing tank (Marlin, 2000), a sampling period ΔT needs to be selected. Its value must ensure that the RL agent is exposed to the process dynamics to prevent steady-state optimization. As ΔT is decreased, more operational data points become available in the same operating period, but the RL agent experiences the effects of its actions to a smaller extent during ΔT.

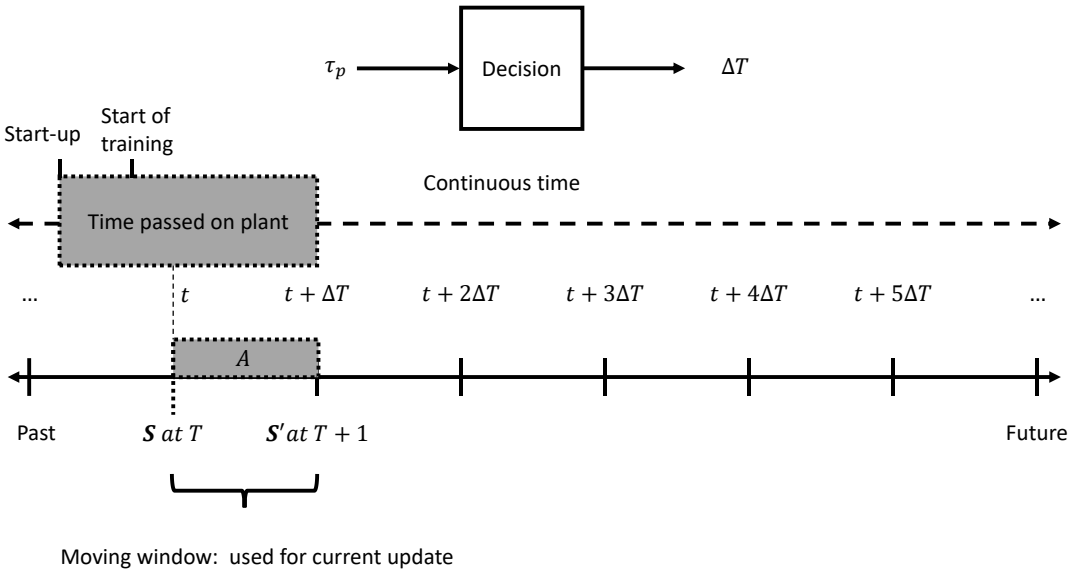


Figure 48: Sampling of times when applying SARSA to the mixing tank problem (offline training by application of Q-learning would occur before online training illustrated in this appendix)

The large grey block in Figure 48 shows qualitatively the time that has actually passed on the plant when performing a training update by using a moving window of discrete time points. The first measurement at time step t is taken after the start of training and is indicated by the thin dashed line connecting the two grey blocks in Figure 48. The RL agent's parameter update can only be performed when the times T and $T + 1$ which comprise this window have physically occurred on the plant, and start-up may be problematic seeing as the RL agent may not have been trained on start-up conditions. Note that the action A (small grey block in Figure 48) is applied from T to $T + 1$ and that the next action A' is selected based on the agent's experience up until time step T applied to the next state $\boldsymbol{S}'$ encountered at time $T + 1$.

The discrete time interaction process between RL agent and RL environment is set within the context of a Markov Decision Process (MDP), where the key assumption made is that the probability of observing $\boldsymbol{S}'$ at time $T + 1$ and the associated reward is only dependent on $\boldsymbol{S}$ and A at time T. This assumption is valid if the state $\boldsymbol{S}$ that constrains the process outputs at each time step T is fully known. Where all components of the state $\boldsymbol{S}$ cannot be measured historical information needs to be included in $\boldsymbol{S}$ to obtain the best approximation of the optimal policy π^* for the RL-based control design (Sutton and Barto, 2018; Moerland et al., 2021).

As a first step towards designing an RL-based controller for this problem, we can map the terminology in Figure 47 to the analogous terminology for RL-based control. This is shown in Table 33.

Table 33: Classical control terminology mapped to analogous RL-based control terminology (Marlin, 2000; Sutton and Barto, 2018)

Term used in classical control	**Analogous RL-based control term**	**Description of RL-based control term**
Feedback error $E(t)$	Component of the observed state $\boldsymbol{S}$	An available measurement that may take on a value from a process-specific set of values and is one of the measurements comprising the state that describes the outputs of a process.
Control law	Policy π	The decision making unit of the agent which takes the observed state $\boldsymbol{S}$ as input and provides the selected action A as output.
Control law output $u(t)$	Action A	Since the controller in its entirety is replaced with an RL-based controller and no final element dynamics are modelled in this example, the action A is the valve position x.
Addressing control objectives through controller design and tuning	Reward function design	The RL agent's only goal is to maximise the cumulative scalar rewards it receives during interaction with the RL environment. By aligning reward function form with the control objectives, the agent can learn behaviour showing cognisance of these objectives.
Modelling of process behaviour in the Laplace domain (analytical/empirical)	Mapping plant measurements to the approximated action-value function, $\tilde{Q}_\pi(\boldsymbol{S}, A)$, or parameterized policy $\pi(A\|\boldsymbol{S}, \boldsymbol{\theta})$ for policy parameter vector $\boldsymbol{\theta}$.	The approximated action-value function, $\tilde{Q}_\pi(\boldsymbol{S}, A)$, contains numerically estimated expected values of the discounted return G_T when following the current policy π from the current state-action coordinate $(\boldsymbol{S}, A)$.

Term used in classical control	Analogous RL-based control term	Description of RL-based control term
		This allows a value-based agent to locate at each T the action corresponding to the greatest cumulative rewards in the future based on the knowledge it has already gained (the greedy action). Exploration is required to gain knowledge at other state-action coordinates. By reading off values from $\tilde{Q}_\pi(\boldsymbol{S}, A)$ and exploring the possible state-action coordinates in a probabilistic fashion, a value-based RL agent instantiates a policy. A value function is unique for a given policy, and the RL agent seeks to update the policy as it updates its approximation to the value function. For Policy Gradient and Actor-Critic control methods, generating $\tilde{Q}_\pi(\boldsymbol{S}, A)$ across the state-action space is not required since the policy is a parameterized probability distribution. The gradient of the policy is sampled using the policy gradient theorem.

Secondly, we need to decide what components to include in the observed state $\boldsymbol{S}$ since we already know that the action applied to the RL environment is simply the fraction valve opening, x – no other process handle is available to affect changes in the CV (the height of liquid H). To define the measurements that need to be included as components of $\boldsymbol{S}$, we need to consider why only feedback error, $E(t)$, is sufficient for a PI/PID controller. For a PI/PID controller, the integral mode ensures steady-state offset and may be shown by applying the final value theorem to a closed-loop system for which PI/PID control suffices (Marlin, 2000).

In contrast to this, an RL agent needs to select appropriate actions directly in the state-action space purely based on experience. Therefore, we need to provide the agent with information about where the liquid level is (H) relative to where it needs to be (SP). Further, the agent needs to know how sensitive the liquid height is to selected value of x depending on the inlet flow rate F_{in} encountered. The required information can be communicated through the use of three components in $\boldsymbol{S}$, potentially $[E(T), F_{in}, SP]^T$, $[H, F_{in}, SP]^T$, or $[E(T), F_{in}, H]^T$, where T in the superscript refers to the transpose operation so that column vectors are defined.

Including all the unique components in these vectors in a single vector, $[H, E(T), F_{in}, SP]^T$ is not efficient – the state-action space in which the learning problem is set becomes much larger by including a fourth dimension, and the agent would need to unnecessarily learn within this space that H and $E(T)$, H and SP, or SP and $E(T)$ would fully constrain both where the liquid level is and where it should be. Similarly, including A as a component of $\boldsymbol{S}$ would require the agent to learn the cause-effect relationship between MV and CV that we already know exists beforehand. It is assumed that the necessary instrumentation is installed for the mixing tank example.

Having said this, in Case Study 1 the RL agent was purposefully provided with only $E(T)$ comprising the scalar-valued observed state S so that the mechanics of value-based control may be investigated without the complications of interpreting an approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S}, A)$ that considers F_{in}, SP, and H as additional state components. For a coarsely discretized state space, the agent will still be able to make sensible decisions while being constrained to the resolution with which the state-action space is modelled. A coarsely discretized state-action space places a restriction on the control behaviour that can be learned by an RL agent.

A binary reward function, as shown in Figure 49, was used to train the SARSA agent. The reward function only communicates to the agent that it is important to keep the liquid height within a $\beta = 0.5\,m$ tolerance band at both sides of the SP by providing a reward of one inside the band and zero outside the band. Clearly, the prevention of excessive final element adjustment is not taught to the agent during training since the reward function of Figure 48 only communicates the importance of achieving CV values sufficiently close to SP, and nothing regarding how the set of available actions should be applied to achieve this result.

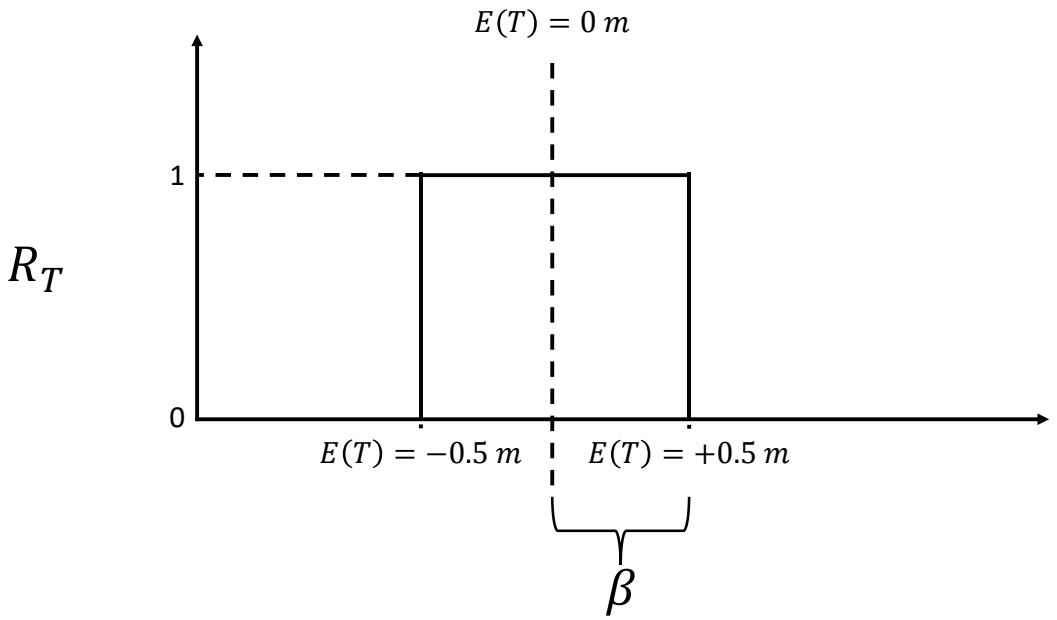


Figure 49: Reward function used when training the SARSA agent to control the liquid level in the mixing tank

Table 34 summarises the state discretization used. The plant measurements $E(T)$, F_{in}, and SP comprise the state components and are mapped to coded state components for the RL agent. The three columns containing coded values, along with the coded actions, form the basis for constructing the action-value hypervolume. This hypervolume is a table in $\mathbb{R}^4$, where each scalar entry in the table is a parameter that needs to be adjusted to learn beneficial behaviour. The table of learned parameters is the approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S}, A)$. Instrument lag was simulated by letting $F_{in}(T+1) = F_{in}(T)$ and $SP(T+1) = SP(T)$, while the only state component for which information regarding $T+1$ was made available during an update at time step T is $E(T+1)$.

Table 34: Definitions of state components (measured online) and their coded counterparts (as "seen" by the SARSA agent) for the mixing tank example

$\boldsymbol{E(T)}$ $\boldsymbol{(m)}$	**Coded** $\boldsymbol{E(T)}$ $(-)$	$\boldsymbol{F_{in}}$ $\boldsymbol{(m^3/min)}$	**Coded** $\boldsymbol{F_{in}}$ $(-)$	$\boldsymbol{SP}$ $\boldsymbol{(m)}$	**Coded** $\boldsymbol{SP}$ $(-)$
-20		0.04		0.4	
	1 (Boundary)		1		1
-2		0.1		0.5	
	2		**2**		**2**
-1.6		0.11		0.6	
	3		**3**		**3**
-1.2		0.12		0.7	
	4		**4**		**4**
-0.8		0.13		0.8	

$E(T)\ (m)$	Coded $E(T)\ (-)$	$F_{in}\ (m^3/min\ )$	Coded $F_{in}\ (-)$	$SP\ (m)$	Coded $SP\ (-)$
	5		**5**		**5**
-0.4		0.14		0.9	
	6		**6**		**6**
0		0.15		1	
	7		**7**		**7**
0.4		0.16		1.1	
	8		**8**		**8**
0.8		0.17		1.2	
	9		**9**		**9**
1.2		0.18		1.3	
	10		**10**		**10**
1.6		0.19		1.4	
	11		**11**		**11**
2		0.2		1.5	
	12 (Boundary)		12		12
20		0.21		2	

The boundary states for the feedback error $E(T)$ capture extremes in liquid height that may be obtained when the SARSA agent learns online purely through interaction without an initial estimate of appropriate entries in $\tilde{Q}_\pi(\boldsymbol{S}, A)$. In the mixing tank model, Equation [118], no limit is placed on the liquid height that may be achieved. As a result, there are a few instances of unrealistic liquid heights which were encountered during the training of the RL agent. Specifically, 0.47 % of the total number of time steps used for training was associated with an overflow condition, where such a condition was specified as $H \geq 3\ m$. This stresses the fact that extremes of operation may be encountered during online learning. The low occurrence of the overflow condition does show that including a liquid level saturation condition in Equation [118] would have resulted in the generation of a similar $\tilde{Q}_\pi(\boldsymbol{S}, A)$ during agent training. The bold numbers in Table 34 indicate the coded state components that we know, from our knowledge of the process, will be encountered often. For F_{in} and SP, which are exogenous inputs to the control system, the boundary states were not encountered in this example.

The overflow condition would be much more problematic if the mixing tank was modelled as an Integrator system. That is, if the liquid in the outlet was pumped so that the rate of change in the mixing tank's liquid height $\left(\frac{dH}{dt}\right)$ becomes independent of the liquid height H (not self-regulatory), exploration would be prone to inducing problematic extremes in the liquid level.

From this it can be seen that the curse of dimensionality quickly becomes problematic in the tabular case if an excess number of components are included in $\boldsymbol{S}$ since this increases the number of degrees of freedom associated with each action A, and the agent needs to deal with this excessive state-action space through the gaining of experience. Ten evenly spaced actions between $x = 0.01$ and $x = 0.892$ were provided to

the agent, where coded $A = 1$ corresponds to $x = 0.01$. These actions were selected by the agent as coded actions, which are then mapped to the corresponding value of x before being applied to the mixing tank. The MATLAB code used to discretize the components of $\boldsymbol{S}$ as shown in Table 34 and the discrete actions are given in Table 35. Note that the discrete actions, "dActions", contains one final entry $x = 0.99$ that is not provided to the agent.

Table 35: MATLAB code excerpt used to create the discretized components of $\boldsymbol{S}$ and the discrete actions

```
%% discretize states and actions
%% max number of states in one dimension = 10; minimum 1
%% fill in (number of states + 1) and (number of actions + 1) below
numError_intervals = 11;
numInletFin_intervals = 11;
numSP_intervals = 11;
numAction_intervals = 11;

%% define lower bounds for discretizations
errorLow = -2;
inletFlowRateLow = 40*(1/1000);
SPLow = 0.4;

%% define "padding" for discretization
numSP_padding = SPLow.*ones((13 - (2 + numSP_intervals)),1)';
error_padding = errorLow.*ones((13 - (2 + numError_intervals)),1)';
inletFlowrate_padding = inletFlowRateLow.*ones((13 -
(2 + numInletFin_intervals)),1)';

dActions = linspace(0.01,0.99,numAction_intervals)';

%% discretize states
% control error
fineRes = linspace(-2,2,numError_intervals);
dStates = [-20,error_padding,fineRes,20]';

% inlet flow rate
fineInletFlowrateRes =
linspace(100*(1/1000),200*(1/1000),numInletFin_intervals);
dStates(1:1:end,2) =
[40*(1/1000),inletFlowrate_padding,fineInletFlowrateRes,210*(1/1000)]';

% height SP
fineHSPRes = linspace(0.50,1.5,numSP_intervals);
dStates(1:1:end,3) = [0.4,numSP_padding,fineHSPRes,2]';


%% create Q-table
% create action-value hypervolume
Reps.action_value = zeros([(size(dStates,1)-1),(size(dStates,1)-1),
(size(dStates,1)-1),...
                  (size(dActions,1)-1)],'double');
```

Table 36 provides a MATLAB code excerpt that was used to define the reward function (two options are predefined and may be selected by changing the "binReward" flag as described in the implementation code. The variable "rlEnv.Terminal" is used to define a stopping condition should it be required that the agent should not train for all the steps predefined for an episode. In all the simulation work of this thesis, such a terminal state is not used, and it is thus set to a numerical value never achieved.

Table 36: MATLAB code excerpt used to create the reward function and to simulate the RL environment, the interaction with which is assumed to be modelled well as a Markov Decision Process

```
% function to create the parameters for the RL environment's model
function rlEnv = createRewardShape(beta)
    rlEnv.continuousReward = @(controlError) exp((-1*controlError^2)/0.01);
    rlEnv.binReward = @(controlError) -1 + 1*(controlError < beta) +
1*(controlError > -beta);
    rlEnv.Terminal = -1000;

end


% function to simulate the MDP
function [nxtH] =
simulateMDP(currentTimeStamp,prevModelStates,action_1,disturbanceValue,p)
    MDPstart = currentTimeStamp;
    MDPstop = currentTimeStamp + 1;
    tspan = linspace(MDPstart, MDPstop, 10);
    p.Fin = p.FinSS + disturbanceValue;
    p.x = action_1;
    [~, HOutput] = ode45(@ (t, H) tankModel(t,H,p), tspan,
prevModelStates);
    nxtH = HOutput(end,1);

end
```

Now that we have decided on the discretization of states and actions, created the discretized hypervolume which will represent $\tilde{Q}_{\pi}(\boldsymbol{S}, A)$, and defined the RL environment and reward function, we can start the training process for the SARSA agent. As described in Section 5.1, a low-pass filter is incorporated to filter the SARSA agent's discrete action selections. This simplifies the simulated control problem for the RL agent, but results in a different and degraded optimal policy π^* being targeted than is the case in state-of-the-art methods.

It should be noted that obtaining an offline first guess of good numerical entries in $\tilde{Q}_{\pi}(\boldsymbol{S}, A)$ before online training requires training on a different RL environment. This is because historical plant data typically will not describe control data degraded owing to the incorporation of such a filter. This is a practical limitation of the methodology used as it is not would not make sense to train off-policy and on-policy using different RL environments.

During the training process of an online RL agent like SARSA, an update rule is applied to the entry of $\tilde{Q}_\pi(\boldsymbol{S}, A)$ corresponding to the current $\boldsymbol{S}$ and A (the state-action coordinate). This update rule is simply a recursive calculation which is applied to a state-action coordinate as it is encountered. The update rule for SARSA is given in Equation [119] for a current state-action coordinate relevant to the mixing tank, where $Q(\boldsymbol{S}, A)$ is the hypervolume entry, $\alpha \in (0,1)$ is a hyperparameter that determines the size of each adjustment (a step size), R_T is the reward obtained when the RL environment reaches the next state $\boldsymbol{S}'$ at which time step, $T+1$, the next action A' is applied to the RL environment, and $\gamma \in [0,1]$ determines whether the agent prioritizes future (γ closer to one) or immediate (γ closer to zero) rewards. Equation [119] is a weighted average between the target value $[R_T + \gamma Q(\boldsymbol{S}', A')]$ and the current table entry $Q(\boldsymbol{S}, A)$. The weighting factor is α.

$$Q(\boldsymbol{S}, A) \leftarrow Q(\boldsymbol{S}, A) + \alpha[R_T + \gamma Q(\boldsymbol{S}', A') - Q(\boldsymbol{S}, A)] \qquad [\ 119\ ]$$

An episode is a predefined number of training steps that occurs before the state is reset and the training process is continued. Each episode consists of a number of time steps with $T = 1$ being the first time step. In this example, each episode starts with state components corresponding to the initial steady-state condition of the mixing tank, i.e., $SP = 1.5\,m$, $F_{in} = 0.19\,\frac{m^3}{min}$, and $H = 1.5\,m$. From Table 34, this corresponds to coded $\boldsymbol{S} = [coded\ E(T), coded\ F_{in}, coded\ SP]^T = [6, 10, 11]^T$.

Since $\tilde{Q}_\pi(\boldsymbol{S}, A)$ is initialized with zero entries, $\tilde{Q}_\pi(\boldsymbol{S}, A) = \boldsymbol{0}$, the first step of training ($T = 1$ for episode one) does not have a unique greedy action. Tie-breaking is performed whereby an action A from the set of available actions is selected at random if more than one action A is greedy at observed state $\boldsymbol{S}$. The RL agent now applies action A to the RL environment which causes a transition from $\boldsymbol{S}$ to $\boldsymbol{S}'$. The online measurements provided by the instrumentation installed for the mixing tank show that $\boldsymbol{S}' = \left[0.12\,m, 0.19\,\frac{m^3}{min}, 1.5\,m\right]^T$. This means that the application of the selected coded $A = 8$ resulted in $E(T) = 0.12\,m$, and therefore the liquid height reduced to $H = 1.38\,m$. Since this is within a $\beta = 0.5\,m$ distance from $SP = 1.5\,m$, $R_T = 1$ is provided to the SARSA agent. By applying Table 34, coded $\boldsymbol{S}' = [7, 10, 11]^T$. We have now completed the $(\boldsymbol{S}', A, R_T, \boldsymbol{S}', A')$ tuple as A' will also be random during this first step of training. We can therefore perform an update to the hypervolume entry $Q(\boldsymbol{S}, A)$ by applying Equation [119]. This is shown in Equation [120].

$$Q(\boldsymbol{S}, A) \leftarrow 0 + \alpha[1 + \gamma(0) - 0] \qquad [\ 120\ ]$$

After the first training step, $Q(\boldsymbol{S}, A)$ contains a non-zero entry α. The greedy action corresponding to $\boldsymbol{S}$ after one time step of experience is coded $A = 8$. This numerical value remains unchanged in the approximator $\tilde{Q}_\pi(\boldsymbol{S}, A)$ until the same state-action coordinate is encountered again, in which case the previous value is adjusted by adding $\alpha[R_T + \gamma Q(\boldsymbol{S}', A') - Q(\boldsymbol{S}, A)]$ (Equation [119]). At each new instance of T, $\boldsymbol{S}'$ and A' of the previous time step become $\boldsymbol{S}$ and A of this new time step. The time steps increase by one after each interaction, therefore the example update of Equation [120] is associated with

$T = 1$, while the next interaction would be associated with $T = 2$, etc. The probability of selecting a random action for the purpose of exploration is a hyperparameter $\varepsilon \in (0,1)$. If a greedy action is present at T, this action is selected with probability $(1 - \varepsilon)$ in which case the agent would be exploiting knowledge already gained. In this example, with probability $\varepsilon = 0.1$, a random action would be selected so that new knowledge may be gained (exploration). MATLAB code that was used to find the greedy action in this ε-greedy exploration strategy is shown in Table 37.

Table 37: MATLAB code excerpt used to select a discrete action (the vector "par.epsilonVec" is used to store the predefined probability of a random action, but only one constant probability 0.1 is used in this example)

```
% function to select action
function Action_1 =
selectAction(Reps,par,state_1,state_2,state_3,numberOfActions)
    t = rand(1);
    if t <= par.epsilonVec(state_3,1)
        % take random action
        Action_1 = randi(numberOfActions);

    elseif t > par.epsilonVec(state_3,1)
        % take greedy action
        vec = Reps.action_value(state_1,state_2,state_3,:);
        index = find(ismember(vec(:),max(vec(:))));
        [~,~,~,Action_1] = ind2sub(size(vec(:,:,:,:)), index);

    end

    % tie breaking
    if size(Action_1,1) > 1
        Action_1 = randi(numberOfActions);
    end

end
```

Interaction steps like the one described above continue until a predefined stopping condition is encountered. The stopping condition used is a fixed number of training episodes and a fixed number of time steps per episode. During each training episode, 10 SP values and 10 DV values were sampled from bounded uniform distributions, where the bounds were $[0.5\ m, 1.5\ m]$ and $\left[0.1\frac{m^3}{min}, 0.2\frac{m^3}{min}\right]$, respectively. This promotes the realisation of many state-action coordinates across the operating range during training. The times of these changes were also sampled from a bounded uniform distribution. Potential instances of small decreases in the numbers of SP and DV values selected during an episode as a result of duplicate time step sampling would not compromise the training process. MATLAB code used to sample DV and SP values at the start of each training episode is shown in Table 38.

Table 38: MATLAB code excerpt used to sample DVs and SPs for a training episode ("rv" contains the ranges of the DVs and SPs

```
% function to create vector of sampled DVs, SPs and times for these changes
% SPs and DVs sampled using scaled uniform distributions.
function [SP_times,SP_steps,DV_times,DV_steps] =
generateEpRVSamples(rv,upperTimeBound)
    %% generate SP sampling data
    SP_times = sort( ceil(1 + upperTimeBound*rand(1,rv.numSP)) );
    for SP_cntr = 1:1:(rv.numSP)

SP_steps(SP_cntr) = rv.lower_SP + (rv.upper_SP - rv.lower_SP)*rand(1,1);

    end

    %% generate DV sampling data
    DV_times = sort( ceil(1 + upperTimeBound*rand(1,rv.numDV)) );
    for DV_cntr = 1:1:(rv.numDV)

DV_steps(DV_cntr) = rv.lower_DV + (rv.upper_DV - rv.lower_DV)*rand(1,1);

    end

end
```

The tuning and hyperparameter settings used for RL agent training are given in Table 39 where τ_f is the filter time constant (see Section 5.1 for the filter form used). The sampling period is the actual process time that passes as the training process progresses from time step T to time step $T + 1$.

Table 39: Tuning and hyperparameters for the mixing tank example

Parameter	Numerical Value	Description
ε	0.1	probability of taking a random action
α	0.7	step size hyperparameter
γ	0.99	discount factor
β	0.5 m	reward function width specification
τ_f	20 min	time constant used for the low-pass filter of Section 5.1
ΔT	1 min	sampling period

The SARSA agent was trained to control liquid level in the mixing tank for 3 000 episodes with 1 200 steps per episode (1 199 transitions from T to $T + 1$). Including the $T = 1$ steps in the count, approximately 10 years of operational data was used during training. The wall time using serial computation for training was approximately 2 000 seconds (33.33 min), which is largely driven by the low computational expense associated with solving the mixing tank model (Equation [118]). After training,

6 672 of the entries in $\tilde{Q}_\pi(\boldsymbol{S}, A)$ had non-zero numerical values. The bold coded state components of Table 34 show that 10^4 entries are likely to be experienced often during training. Hence, 66.72 % of the non-boundary state entries of $\tilde{Q}_\pi(\boldsymbol{S}, A)$ have non-zero values. Importantly, the discount factor γ causes non-zero values to propagate throughout $\tilde{Q}_\pi(\boldsymbol{S}, A)$ and not just at state-transitions directly producing increased rewards as the training process progresses.

After training, the approximated action-value function $\tilde{Q}_\pi(\boldsymbol{S}, A)$, which initially contains its 6 672 non-zero entries obtained from training, was used with the same hyperparameters as reported in Table 39 to attenuate for a step decrease in F_{in} by adjusting the valve fraction open, x. Two cases were simulated. These are a success case where the RL agent was able to maintain the liquid height within the desired range despite the step change in F_{in} and a failure case where the RL agent did not manage to do this. The SP, F_{in}, and rewards obtained are given in Figure 50 and Figure 52 for the success and failure case, respectively. The discrete x settings selected by the agent and the MV values applied to the process after being sent through the low-pass filter, as well as the behaviour of liquid height within the mixing tank are given for the success and failure cases in Figure 51 and Figure 53, respectively.

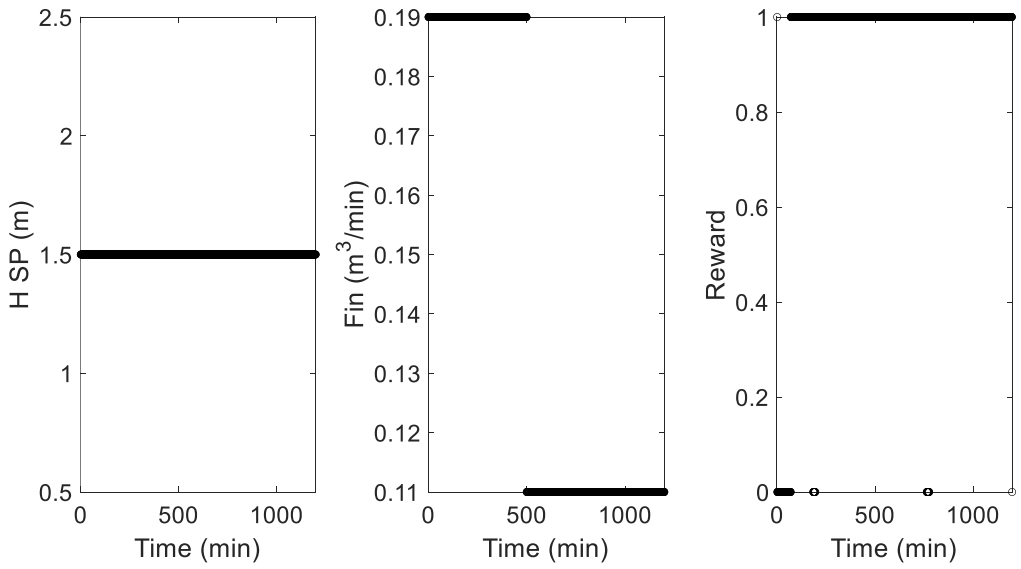


Figure 50: Liquid level SP (left panel), F_{in} step change (central panel), and rewards obtained (right panel) for success case

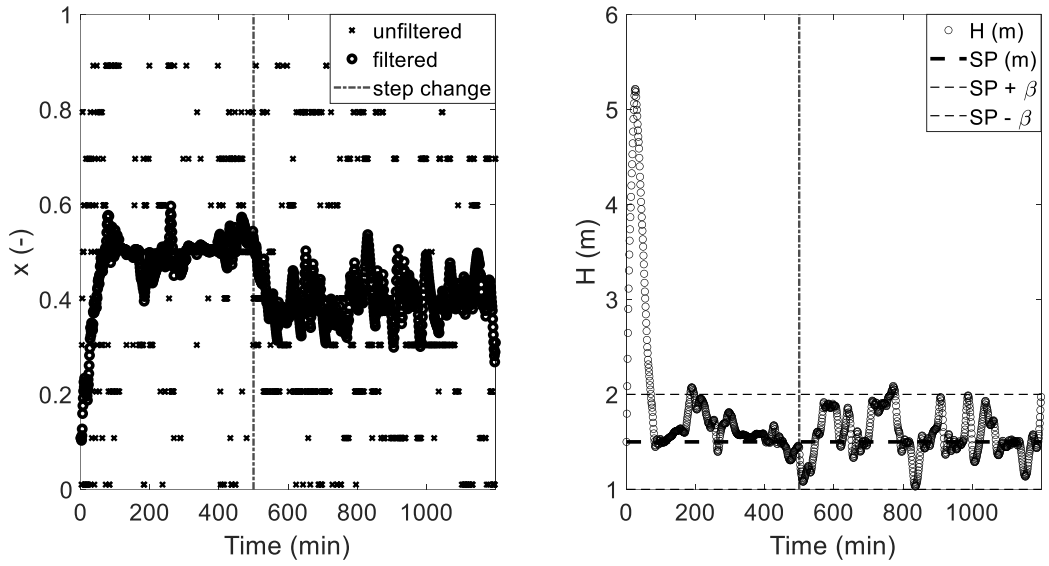


Figure 51: Fraction valve opening x before and after undergoing low-pass filter calculation (left panel) and liquid level progression (right panel) for success case – vertical lines indicate time of step change in F_{in}

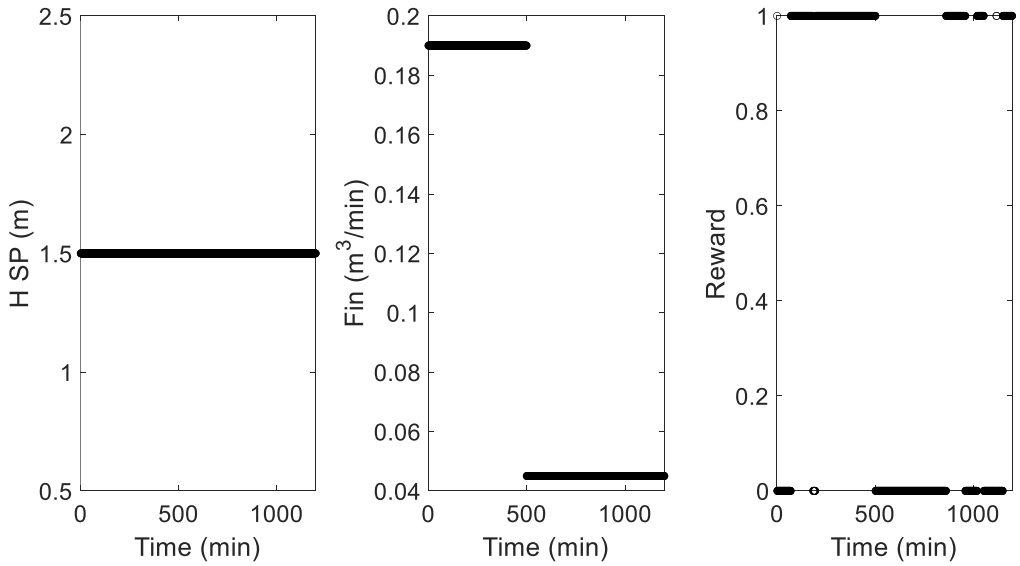


Figure 52: Liquid level SP (left panel), F_{in} step change (central panel), and rewards obtained (right panel) for failure case

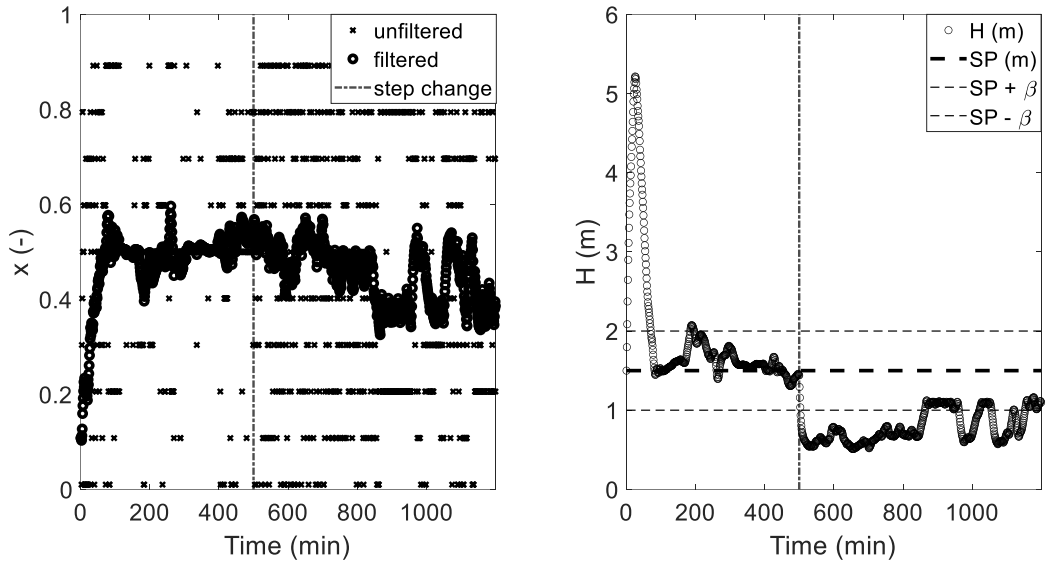


Figure 53: Fraction valve opening x before and after undergoing low-pass filter calculation (left panel) and liquid level progression (right panel) for failure case – vertical lines indicate time of step change in F_{in}

For the success case, it is observed that there is initially an overflow condition present in the mixing tank, since the simulated liquid height rises in excess of $5\ m$. This is a result of the RL agent not learning how to match the initial steady state of the plant before making sensible adjustments to x and such cases are expected to make up a significant proportion of the overflow cases recorded during training. The RL agent intentionally decreases x when the inlet flow rate is decreased and its capacity to do so stems from the value of F_{in} achieved after the step input being encountered owing to the sampling range of $\left[0.1\frac{m^3}{min}, 0.2\frac{m^3}{min}\right]$ containing $F_{in} = 0.11\frac{m^3}{min}$ – the coverage of the encountered states was sufficient during training.

For the failure case, there is one key difference in the data reported in Figure 53. The step change in F_{in} was not incorporated during the state component discretization of Table 34. Since the tabular SARSA agent never experienced such a state component, the slice of $\tilde{Q}_\pi(\boldsymbol{S}, A)$ associated with $F_{in} = 0.045\frac{m^3}{min}$ still mostly contains zero-valued entries. Hence, the RL agent's decision making becomes random just after the step change in F_{in} – this is evident just to the right of the vertical dashed line in the left panel of Figure 53. The coverage of the encountered states was insufficient during training.

One method of improving the coverage of the states and actions during training is to replace the use of discrete scalar entries in a tabular approximator $\tilde{Q}_\pi(\boldsymbol{S}, A)$ with a linear basis function model. This is applied to the One-Step Actor-Critic algorithm in this thesis to represent the required value function and policy. The principle is to have a vector containing parameters that are adjusted during each step of training. Each entry in this vector is a scalar weight that is multiplied with a function that maps the state-action coordinate encountered to $\mathbb{R}^1$. For basis functions $\mathcal{X}_1(\boldsymbol{S}, A)$ through $\mathcal{X}_5(\boldsymbol{S}, A)$, Equation [121] shows the output at time step T for such an approximator with parameter vector $\boldsymbol{\theta}$ containing only elements θ_1 through θ_5. In

contrast to tabular methods, each state does not require a new parameter to be added to the set of parameters that must be learned by the RL agent. This is because each basis function $\mathcal{X}(\boldsymbol{S}, A)$ has a perceptive field in the state-action space. Updates to parameters in one region of the state-action space therefore generalize to other regions of the state-action space (Sutton and Barto, 2018).

$$output = \theta_1 \mathcal{X}_1(\boldsymbol{S}, A) + \theta_2 \mathcal{X}_2(\boldsymbol{S}, A) + \theta_3 \mathcal{X}_3(\boldsymbol{S}, A) + \theta_4 \mathcal{X}_4(\boldsymbol{S}, A) + \theta_5 \mathcal{X}_5(\boldsymbol{S}, A) \qquad [\ 121\ ]$$

APPENDIX B – ANALYTICAL EXPRESSION FOR THE ELIGIBILITY VECTOR

The goal of this appendix is to clarify how $\frac{\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}$ is simplified to obtain an analytical direction for Actor-Critic updates.

Let x and y denote scalars in $\mathbb{R}^1$ related through Equation [122] and a function applied to a vector denote the function applied to each element within the vector. Equations [123] through [128] may then be used to express the derivative of a natural logarithm with respect to x.

$$y = \ln(x) \quad [\ 122\ ]$$

$$x = \exp(y) \quad [\ 123\ ]$$

$$\frac{d}{dx}(x) = \frac{d}{dx}(\exp(y)) \quad [\ 124\ ]$$

$$1 = \exp(y)\frac{dy}{dx} \quad [\ 125\ ]$$

$$\frac{dy}{dx} = \frac{1}{\exp(y)} \quad [\ 126\ ]$$

$$\frac{dy}{dx} = \frac{1}{\exp(\ln(x))} \quad [\ 127\ ]$$

$$\frac{d}{dx}\ln(x) = \frac{1}{x} \quad [\ 128\ ]$$

The scalar x may be replaced by a function f of a vector $\boldsymbol{x}$, which gives rise to Equation [129]. Importantly, the output of f must be a scalar. Substituting $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{P}}$ for $\boldsymbol{x}$ and $\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})$ for f, Equation [130] is obtained. After substituting Equation [26], Equation [131] is obtained. Each element of Equation [131] has the same form and hence simplifying the eligibility vector now reduces to obtaining a simplified expression for $\frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta}$. The final steps in the simplification are shown in Equations [132] through [134] for the i^{th} element of $\frac{\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}$.

$$\nabla_{\boldsymbol{x}}\ln\big(f(\boldsymbol{x})\big) = \frac{\mathbf{1}}{f(\boldsymbol{x})}\nabla_{\boldsymbol{x}}f(\boldsymbol{x}) \quad [\ 129\ ]$$

$$\nabla_{\boldsymbol{\theta}}\ln\big(\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})\big) = \frac{\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})} \quad [\ 130\ ]$$

$$\frac{\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})}{\pi(\boldsymbol{A}_T|\boldsymbol{S}_T,\boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right) = \begin{bmatrix} \frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta_1} \\ \frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta_2} \\ \vdots \\ \frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta_p} \end{bmatrix} \quad [131]$$

$$\frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta_i} = \frac{1}{\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}} \left(\frac{\left[\frac{\partial\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\partial\theta_i}\right]\sum_b \exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta})) - \exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))\sum_{\boldsymbol{b}}\frac{\partial\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}{\partial\theta_i}}{\left(\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))\right)^2}\right) \quad [132]$$

$$\frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta_i} = \frac{1}{\frac{\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})\right)}{\sum_{\boldsymbol{b}}\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)}} \left(\frac{\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})\right)\mathcal{X}_i(\boldsymbol{S},\boldsymbol{A})}{\sum_{\boldsymbol{b}}\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)} - \frac{\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{A})\right)\sum_{\boldsymbol{b}}\left[\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)\mathcal{X}_i(\boldsymbol{S},\boldsymbol{b})\right]}{\left(\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))\right)^2}\right) [133]$$

$$\frac{\partial \ln\left(\frac{\exp(h(\boldsymbol{S},\boldsymbol{A},\boldsymbol{\theta}))}{\sum_{\boldsymbol{b}}\exp(h(\boldsymbol{S},\boldsymbol{b},\boldsymbol{\theta}))}\right)}{\partial\theta_i} = \left[X_i(\boldsymbol{S},\boldsymbol{A}) - \frac{\sum_b\left[\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)\mathcal{X}_i(\boldsymbol{S},\boldsymbol{b})\right]}{\sum_{\boldsymbol{b}}\exp\left(\boldsymbol{\theta}^T\boldsymbol{X}(\boldsymbol{S},\boldsymbol{b})\right)}\right] \quad [134]$$

APPENDIX C – SHOUKAT CHOUDHURY ET AL. (2005) STICTION MODEL

The code given below must be run as a MATLAB script:

```
%% short description
% script used to validate the implementation of the data-driven stiction
% model of Choudhury et.al. as described in:
% "Modelling valve stiction" by M.A.A. Shoukat Choudhury, N.F. Thornhill
% and S.L. Shah
clc
clear
%% create sine wave
f = @(x) (50*sin((3*2*pi/200)*x) + 100)/200; % normalized sine wave of
similar frequency to article (range 0.25 to 0.75)
time_To_Simulate = 200; % number of wave steps to simulate
%% specify larger bounds to prevent valve saturation...
MV_low_bound = 0;
MV_high_bound = 1;
%% model parameters
S = 20;  % deadband plus stick band parameter
J = 10;  % slip jump parameter

for currentTimeStamp = 1:1:time_To_Simulate
    %% stiction calculation starts
    %% initialize actions in MV units
    if currentTimeStamp == 1
        crntAction = f(currentTimeStamp);
        nxtAction = crntAction;

    elseif currentTimeStamp > 1
        crntAction = f(currentTimeStamp - 1);
        nxtAction = f(currentTimeStamp);

    end
    %% stiction model applied to valve output

    if currentTimeStamp == 1
        xss = 0;    % initialize memory variable for output signal when valve
becomes stuck
        % previous output as a % of the final element range
        MV_output_previous = ( (crntAction - MV_low_bound)/
      (MV_high_bound -  MV_low_bound) )*100;%crntAction;
        MV_incoming_present = crntAction; % initialize present MV signal
        MV_incoming_previous = crntAction;% initialize previous MV signal
        vnew_prev = 0; % initialize previous local gradient of control signal
        I = 0; % initialize flag for valve stuck during trajectory
    else

        MV_output_previous = MV_output; % previous output as a % of the final
element range
        MV_incoming_present = nxtAction;% present MV signal
        MV_incoming_previous = crntAction;% previous MV signal
        vnew_prev = vnew; % previous local gradient of control signal

    end
    %% stiction calculation, MV_output is a % of MV range
    [MV_output,I,xss,vnew,MV_Percentage_present] = stictionModel(J,S,xss,...
```

```
                            MV_output_previous,I,...
                            MV_incoming_present,...
                            MV_incoming_previous,...
                            MV_low_bound,MV_high_bound,...
                            vnew_prev);
    controlLawSignal(currentTimeStamp) = f(currentTimeStamp);
    % convert MV_output at current timestamp to actual MV value
    crntAction = (MV_output/100)*(MV_high_bound - MV_low_bound)+
MV_low_bound; % output after stiction model at current timestamp
    % apply stiction model output as current action
    finalElementAdjustment(currentTimeStamp) = crntAction;

end

subplot(2,1,1)
plot(controlLawSignal,'LineWidth',1); hold on
plot(finalElementAdjustment,'LineWidth',2);
legend('control law signal','valve adjustment')
xlabel('time (s)')
ylabel('normalized amplitudes of signals')

subplot(2,1,2)
plot(controlLawSignal,finalElementAdjustment,'k-','LineWidth',2); hold on
xlabel('control law signal'); ylabel('valve adjustment'); grid on
axis square

%% function to model valve stiction
function[MV_output,I,xss,vnew,MV_Percentage_present] = stictionModel(J,S,...
                                                    xss,MV_output_previous,...
                                                     I,MV_incoming_present,...
                                                      MV_incoming_previous,...
                                                        MV_low_bound,...
                                                      MV_high_bound,vnew_prev)

%% saturated condition and scaling of present MV input signal
if MV_incoming_present <= MV_low_bound
    MV_Percentage_present = 0;
    MV_output = 0;

elseif MV_incoming_present >= MV_high_bound
    MV_Percentage_present = 100;
    MV_output = 100;

elseif ( MV_incoming_present < MV_high_bound ) && ( MV_low_bound <
MV_incoming_present )
    MV_Percentage_present = ( (MV_incoming_present -
MV_low_bound)/(MV_high_bound - MV_low_bound) )*100;

end

%% scaling of previous MV input signal
if MV_incoming_previous <= MV_low_bound
    MV_Percentage_previous = 0;

elseif MV_incoming_previous >= MV_high_bound
    MV_Percentage_previous = 100;

elseif ( MV_incoming_previous < MV_high_bound ) && ( MV_low_bound <
MV_incoming_previous )
```

```
    MV_Percentage_previous = ( (MV_incoming_previous -
MV_low_bound)/(MV_high_bound - MV_low_bound) )*100;

end

vnew = (MV_Percentage_present - MV_Percentage_previous)/(1); % gradient of
incoming control signal (consistently i.t.o. MDP time)
%%
if MV_Percentage_present > 0 && MV_Percentage_present < 100
    vold = vnew_prev;
    vnew = (MV_Percentage_present - MV_Percentage_previous)/(1); % gradient
of incoming control signal (consistently i.t.o. MDP time)
    if sign(vnew) == sign(vold)
        if I ~= 1 % if not stuck during valve transient behaviour
            % absolute difference between current signal and previous stuck
signal
            DIFF = abs(MV_Percentage_present - xss);
            if DIFF > S
                % adjust output from valve if DFF is greater than
                % (dead-band and stick band), i.e. valve is at
                % beginning of trajectory
                MV_output = MV_Percentage_present - sign(vnew)*( (S - J)/2 );
            else
                % keep output at stuck value
                MV_output = MV_output_previous;
            end
        elseif I == 1 % if in stuck condition during moving phase
            DIFF = abs(MV_Percentage_present - xss);
            if DIFF > J % if DIFF is greater than slip jump
                I = 0; % remove flag for being stuck during moving phase
                % adjust MV_output
                MV_output = MV_Percentage_present - sign(vnew)*( (S - J)/2 );
            else
                % keep input at the same value
                MV_output = MV_output_previous;
            end
        end
    elseif sign(vnew) ~= sign(vold) % if direction of valve movement changes
        if sign(vnew) == 0 % if valve reaches a stop position during a
transient
            I = 1; % indicate this stop condition with flag
            xss = MV_Percentage_previous;%MV_incoming_previous; % update
memory variable for previous stuck position
            MV_output = MV_output_previous; % keep MV output signal constant
        else
            % do the same for a stuck position not reached during the
            % valve's moving phase
            xss = MV_Percentage_previous;%MV_incoming_previous;
            MV_output = MV_output_previous;
        end
    end

end % end check for final element saturation
end % end stiction function
```

APPENDIX D – LEARNING CURVE CONSTRUCTION FROM EPISODE INSTANCES

As described by Sutton and Barto (2018) and Poole and Mackworth (2010), the extents and rates of convergence of different RL algorithms may be compared through the construction of learning curves. The first learning curve type is a plot of the rewards obtained for a pre-defined window length as a function of the number of training episodes. For a fixed terminal state, such a learning curve will reach a plateau when the optimal performance for the particular agent design has been achieved. Alternatively, the total accumulated reward may be drawn. When the agent reaches its best performance, the curve will achieve a constant slope which will be directly proportional to the plateau reached in the window-based learning curve and is more prone to accuracy problems when comparing agents.

When a stochastic training scheme is used for parallel training, it becomes important to consider how the extent of convergence may be compared for difference instances of state-action space discretization. This task may be quite difficult if a limited number of operational hours are available for training and the RL environment behaviour is stochastic.

Recall from Section 5.3.3 that the parameters of the algorithm used to generate learning curves in this scenario are the fixed episode length, Ep, and the window size W. Let the number of episodes for which policy instances have been generated be denoted as N_{ep} for a fixed window size W and fixed episode length Ep. The value selected for N_{ep} must be significantly smaller than the length of Ep. This ensures that the policy represented by each episode instance does not change significantly during learning curve construction. The algorithm used for learning curve construction is shown in the pseudocode of Table 40. In Table 40, $\tilde{Q}_{\pi}(\boldsymbol{S},\boldsymbol{A})_{ep_{cntr}}$ refers to the approximated action-value function generated specifically during the newest episode of training.

Table 40: Pseudocode for learning curve construction from parallel episode instances for observed state $\boldsymbol{S}$ and action $\boldsymbol{A}$

1. $\#\ learning\ curve\ entries = \left(\frac{N_{ep}}{W}\right)$
2. $Initialize\ \tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A}) \leftarrow \mathbf{0}$
3. $Initialize\ Ep \ll N_{ep}$
4. $for\ cntr = 1{:}\,1{:}\,N_{ep}$
    i. $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A}) \leftarrow \tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A}) + \tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})_{ep_{cntr}}$
    ii. $for\ inner = 1{:}\,1{:}\,Ep$
        a. Run training function
        b. Obtain updated $\tilde{Q}_{\pi}(\boldsymbol{S}, \boldsymbol{A})$ through serial SARSA execution
    iii. end
    iv. $if\ multiple\ of\ W\ is\ reached$
        a. Sum all rewards for the current window instance
    v. end
5. end

APPENDIX E – FACTOR SCREENING EXPERIMENTAL DESIGN RAW DATA PROCESSING FOR HYPERVOLUME CHARACTERIZATION

Table 41: Experimental design and raw IAE data generated during 2^4 factorial experiment

A	B	C	D	A	B	C	D	AB	BC	AC	DC	BD	AD	ABC	BCD	ABD	ACD	ABCD	Average IAE	
L	L	L	L	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1	0.072	1)
H	L	L	L	1	-1	-1	-1	-1	1	-1	1	1	-1	1	-1	1	1	-1	0.069	a
L	H	L	L	-1	1	-1	-1	-1	-1	1	1	-1	1	1	1	1	-1	-1	0.048	b
H	H	L	L	1	1	-1	-1	1	-1	-1	1	-1	-1	-1	1	-1	1	1	0.060	ab
L	L	H	L	-1	-1	1	-1	1	-1	-1	-1	1	1	1	1	-1	1	-1	0.055	c
H	L	H	L	1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	-1	1	0.056	ac
L	H	H	L	-1	1	1	-1	-1	1	-1	-1	-1	1	-1	-1	1	1	1	0.045	bc
H	H	H	L	1	1	1	-1	1	1	1	-1	-1	-1	1	-1	-1	-1	-1	0.057	abc
L	L	L	H	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	-1	0.056	d
H	L	L	H	1	-1	-1	1	-1	1	-1	-1	-1	1	1	1	-1	-1	1	0.084	ad
L	H	L	H	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	-1	1	1	0.041	bd
H	H	L	H	1	1	-1	1	1	-1	-1	-1	1	1	-1	-1	1	-1	-1	0.059	abd
L	L	H	H	-1	-1	1	1	1	-1	-1	1	-1	-1	1	-1	1	-1	1	0.048	cd
H	L	H	H	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	-1	1	-1	0.052	acd
L	H	H	H	-1	1	1	1	-1	1	-1	1	1	-1	-1	1	-1	-1	-1	0.045	bcd
H	H	H	H	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.052	abcd

Table 42: Sum of squares and effect calculation results for action-value hypervolume discretization – 2^4 factorial experiment

Number of replicates	**Effect coefficient**	**SS coefficient**	
1	0.125	0.0625	
Interaction term	**Effect estimate**	**Sum of squares**	**% Contribution to sum of squares**
A	0.010	0.0004	21.54
B	-0.011	0.0005	24.27
C	-0.010	0.0004	21.27
D	-0.003	0.00004	2.28
AB	0.002	0.00002	1.03
AC	-0.004	0.0001	2.72
AD	0.004	0.0001	3.91
BC	0.008	0.0002	12.83
BD	0.000	0.000000001	0.00
CD	-0.001	0.000004	0.22
ABC	0.001	0.000005	0.25
ABD	-0.004	0.0001	3.31
ACD	-0.005	0.0001	4.93
BCD	0.001	0.00001	0.45
ABCD	0.002	0.00002	0.99

Table 43: Data for normal probability plot for the 2^4 factorial experiment

	Effect estimate	Number of observations	z dist. p	z value
B	-0.011	1	0.03	-1.83
C	-0.010	2	0.10	-1.28
ACD	-0.005	3	0.17	-0.97
ABD	-0.004	4	0.23	-0.73
AC	-0.004	5	0.30	-0.52
D	-0.003	6	0.37	-0.34
CD	-0.001	7	0.43	-0.17
BD	0.000	8	0.50	0.00
ABC	0.001	9	0.57	0.17
BCD	0.001	10	0.63	0.34
ABCD	0.002	11	0.70	0.52
AB	0.002	12	0.77	0.73
AD	0.004	13	0.83	0.97
BC	0.008	14	0.90	1.28
A	0.010	15	0.97	1.83

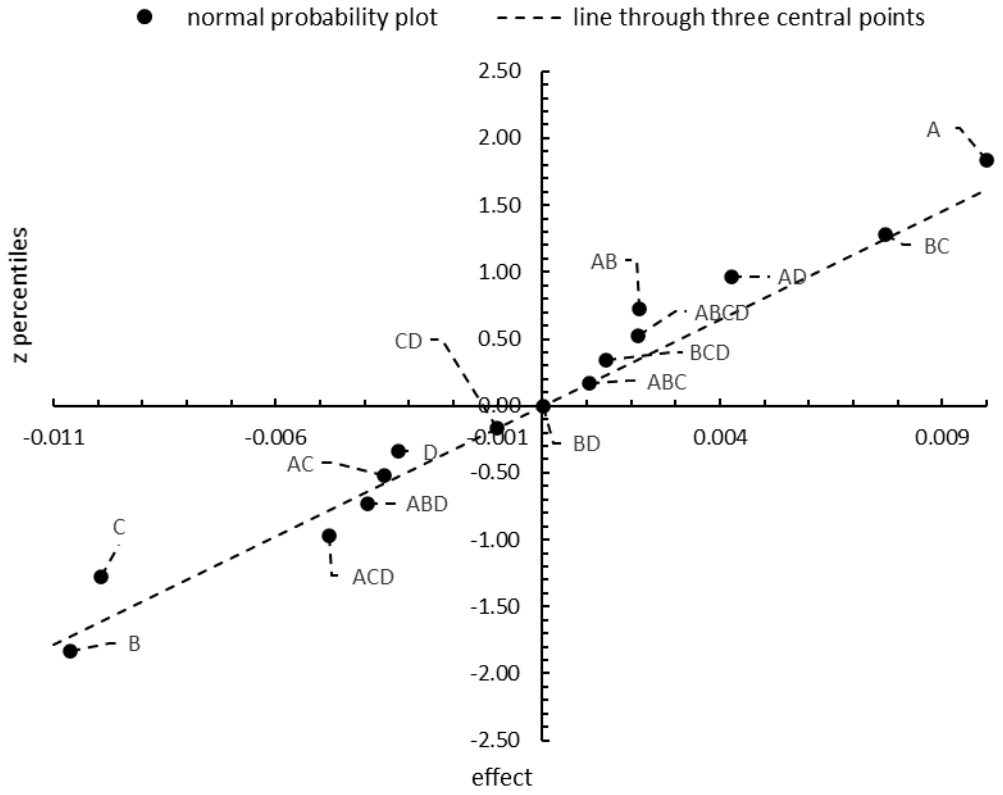


Figure 54: Normal probability plot for the 2^4 factorial experiment

Table 44: ANOVA table for the 2^4 factorial experiment

Source of variation	SS	DOF	Mean square	F statistic	P-value
A	0.00040	1	0.0004	46.3	0.09
B	0.00045	1	0.00045	52.2	0.09
C	0.00039	1	0.00039	45.7	0.09
D	0.00004	1	0.00004	4.9	0.27
AB	0.00002	1	0.00002	2.2	0.38
AC	0.00005	1	0.00005	5.9	0.25
AD	0.00007	1	0.00007	8.4	0.21
BC	0.00024	1	0.00024	27.6	0.12
BD	0.00000	1	0.00000	0.0	0.99
CD	0.00000	1	0.00000	0.5	0.62
ABC	0.00000	1	0.00000	0.5	0.60
ABD	0.00006	1	0.00006	7.1	0.23
ACD	0.00009	1	0.00009	10.6	0.19
BCD	0.00001	1	0.00001	1.0	0.51
ABCD	0.00002	1	0.00002	2.1	0.38
Error (SS_{BD}+SS_{CD}+SS_{ABC})	0.00001	1	0.00001		
Total SS		15			

APPENDIX F – NUMBERS OF PARAMETERS FOR SECTION 6.5

The works cited in Section 6.5 either have neural networks that are specified as having densely connected layers or as feedforward neural networks that are fully connected. Both of these terminologies enable the determination of the number of parameters. The number of connections between neurons (nodes) represents the weights while each neuron after the input layer is also associated with a bias parameter. The same calculation procedure was therefore applicable to the deep RL representations as well as the grinding circuit case study where SANE was applied (Conradie and Aldrich, 2001; Ma et al., 2019; Shipman and Coetzee, 2019). An example calculation is shown for the DDPG implementation of Ma et al. (2019) in Table 45. The calculations for the actor and critic networks are identical. The total numbers of parameters for the actor and critic networks are determined by summing the number of neurons in the hidden layers and output layer (these are the biases) and adding the calculated number of weights.

Table 45: Example calculation of the number of parameters in the RL representation of Ma et al. (2019)

Actor					
fully connected (feedforward)	Input layer	First Hidden Layer	Second Hidden Layer	Output	
Neurons	142	400	400	2	
Weights		56800	160000	800	
	Calculations	(400)(142)	(400)(400)	(2)(400)	
				Total Actor	218402
Critic					
fully connected (feedforward)	Input layer	First Hidden Layer	Second Hidden Layer	Output	
Neurons	142	300	300	2	
Weights		42600	90000	600	
				Total Critic	133802
				Total	352204

In the entries of Table 29 corresponding to the work of Conradie (2000), the maximum number of neurons allowed in the hidden layer for SANE was stated to have 80 connections (weights). It was assumed that the output layer was densely connected. To determine the number of parameters used for the training of SARSA agents in this thesis, the details of Table 20 and Table 24 were used for the Van de Vusse reaction

and the grinding circuit, respectively. The number of parameters counted from these details are shown in Table 46 and Table 47.

Table 46: Calculation of the number of parameters used to represent $\tilde{Q}_\pi(\boldsymbol{S}, A)$ for the Van de Vusse reaction scheme simulations described in Sections 5.3.2 and 6.3.1

$E(T)$	11
$SP(T)$	1
$DV(T)$	2
A	10
Total (product of the four rows above)	220

Table 47: Calculation of the number of parameters used to represent $\tilde{Q}_\pi(\boldsymbol{S}, A)$ when applying SARSA to the grinding circuit model

$E(T)$	6
$SP(T)$	12
$DV(T)$	11
A	5
Total (product of the four rows above)	3 960