Hamburg University of Applied Sciences

Faculty of Life Sciences

Implementation and assessment of Advanced Reinforcement Learning methods for control of chemical processes

Master's Thesis

Process Engineering Master

submitted by

Gary Simethy
[redacted]

Supervisors: : Prof. Dr. Margret Bauer (HAW Hamburg)
Dr. Fabian Buelow (ABB AG)

The thesis was prepared and supervised in cooperation with ABB.

Contents

List of Figures

List of Tables

1 Introduction

In a time characterized by progress in automation, utilizing intricate systems like Advanced Process Control (APC) has become an essential requirement for every manufacturing operation, irrespective of the sector. Of these systems, Model Predictive Control (MPC) is well established and has been used as the standard to address challenging multivariate control tasks for continuous processes in the process industry[5]. However, these controllers are model based and therefore lacks the adaptability to changing process dynamics [6]. As the performance of the controller deteriorates, frequent system identification is required to update the model to capture new dynamics. This is time consuming and introduces costly disruptions to regular operations [7].

This work evaluates Reinforcement Learning (RL) as an alternative control approach capable of manipulating multiple variables simultaneously to attain predefined control objectives while adapting to the dynamics of the process as well as the dynamics of existing controllers such as PI/PID.

Reinforcement Learning (RL) is regarded as a viable alternative due to its demonstrated ability to perform effectively when encountering variations in input conditions and disturbances that can affect the dynamics learnt by the RL agent. This has been demonstrated in applications such as self – driving cars, robotics and games. However, unlike games and robots trial and error is very costly and dangerous on a real chemical process. The next best option is to apply RL on a simulation of a chemical process and then use the trained RL agent on a real process. For this work, an open source simulation of a penicillin production process [8] is used as an environment for RL based control. The simulation selected was validated with historical data from the real process and are in good agreement with each other [2].

In this work, the literature review section examines existing open-source benchmarked simulators and provides rationale for selecting the Indpensim process simulator as a candidate for RL. In the experiments section, the configuration of the Indpensim process simulation as a RL environment is detailed. This includes outlining the adjustments made to render it compatible for training with RL agents.

The results section presents the outcomes of training several state-of-the-art RL algorithms on the configured RL environment. Its primary objective is to identify the most effective RL algorithm that can simultaneously meet the following objectives:

- Maximize the yield of penicillin
- Ensure safe operation

Furthermore, the same RL algorithm (or agent) is evaluated under varying input conditions and vessel dimensions of the process, and simulated process faults using real process data. This is done to assess the generalization capabilities of the selected algorithm to unseen circumstances without the need for retraining.

Finally, the discussion section provides insights into the observed results and compares them with alternative control strategies employed by others for the same process.

2 Literature review

In this section, various open-source process simulators that were benchmarked in a study [1] are briefly discussed. One simulation is selected for the remainder of this study, and the criteria behind

Benchmark	**Operation**	**Platform**
TEP	Continous	Fortran, MATLAB, Modelica
Pensim	Batch	MATLAB
RAYMOND	Continous	MATLAB
DAMADICS	Continous	MATLAB
Indpensim	Batch	MATLAB
FCC	Continous	MATLAB

Table 1: Benchmarked open-source process simulators [1]

this selection are also discussed. Following that, the selected simulation is discussed in detail. This includes topics such as how the simulation was developed and its fidelity to reality, how the process is described and operated, critical parameters of the process determined through sensitivity analysis [9], and the current control strategy. Additionally, related works that were focused on improving upon existing control strategies for the selected simulator are discussed.

Since this study predominantly utilizes RL, the theory behind RL and its advantages over classical control strategies, as well as the different classes of RL algorithms, particularly those suitable for this study, are also discussed.

2.1 Selection of chemical process simulation

Training a reinforcement learning (RL) agent by acting directly on a real chemical plant poses significant risks especially when exploratory actions are considered. Either, data from the plant can be used to train the RL agent (offline RL) or a simulation can be used as an environment for the RL agent to act on. The former is less exploratory because it learns only from experiences (imitation learning) that had happened and cannot generate entirely new experiences. Whereas with a simulation, it is feasible to generate a variety of experiences through random actions, enabling greater exploration. By simulating a chemical process, potential dangers can be mitigated while still providing a training environment for the agent that somewhat mimics the dynamics of the real chemical process. Once trained on simulation, the RL agent can then be potentially deployed to the real process.

To select a suitable chemical process simulation as an environment for the RL agent, the results from a study [1] are utilized. The study analysed and reviewed open source simulations (shown in Table 1) and datasets based on factors such as the fidelity, accessibility of actual data, user-friendliness, challenges to be acknowledged, and potential for scientific progress. The results of that study for each simulation is discussed below.

Among the process simulators listed in the table, a batch operated process would make a good case to apply RL for process control. This is primarily because of the inherently dynamic behavior from continually shifting operating conditions that occur from start-up to batch completion [10]. This operational mode is characteristic of unit operations where the processed volume is relatively limited compared to those operating continuously, and it involves multiple objectives across processing steps to achieve the desired final product. Consequently, it is difficult to capture these dynamics and poses several hurdles to adopting a control framework, particularly regarding technical issues outlined in [11] [12] [13].

The FCC fractionator modelled in 2022 serves as a simulation tool designed for refining operations, featuring a fluidized bed catalytic cracker coupled with a fractionator [14]. The process flowsheet shown in Figure 1 encompasses essential components such as reactor, fractionator, preheater, regenerator, wet gas compressor, catalyst circulation lines, combustion air blower, lift air blowers and process controllers. For accessibility, a MATLAB simulator is readily available for implementing the FCC Fractionator model [14].

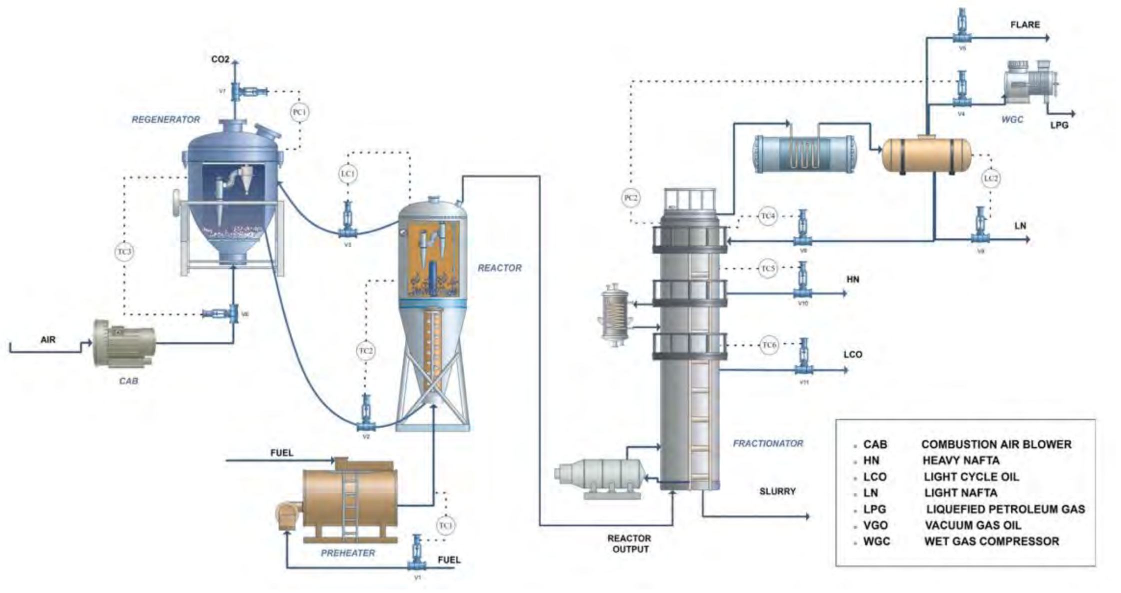

Figure 1: FCC flowsheet [1].

The Tennessee Eastman Process (TEP) stands as a widely accepted simulation model, meticulously designed to replicate the intricate operations within the Eastman Chemical Company, serving as a robust benchmark for continuous process control and monitoring applications [15]. Its inception marked a significant milestone in the realm of modeling studies, garnering widespread acceptance and adoption within the academic community. Over the years, the TEP model has solidified its position as the foremost benchmark, continuously utilized for testing and validating various process monitoring techniques. Initially introduced as a Fortran process simulator, the TEP model has since evolved and adapted to contemporary technological landscapes, finding implementations across multiple platforms including MATLAB/Simulink [16] and Modelica [17].

PenSim is as a simulation framework linked to the penicillin production process. It is based on an unstructured model of penicillin fermentation originally conceptualized in 1980 [18], the framework has been further refined and expanded upon in 2002 [19]. Notably, PenSim has emerged as the preeminent benchmark for evaluating techniques tailored for batch processes. Its significance lies in its ability to simulate the inherent complexities of penicillin fermentation, offering researchers a reliable platform for testing and validating various methodologies. Of particular importance is the availability of a standalone dataset associated with PenSim, as noted by [20] in 2015.

RAYMOND (RAYpresentative MONitoring Data), introduced by [21], emerged as a MATLAB package designed to facilitate the generation of comprehensive monitoring data through simulations. Within this package, users can readily access established benchmarks such as the Tennessee

Eastman Process (TEP) and the PenSim models. Additionally, RAYMOND offers the flexibility for users to incorporate their own models, allowing for customization of sensor characteristics, process variability, input fluctuations, control strategies, and fault scenarios. One notable advantage of employing RAYMOND for simulating the PenSim model is its capability to introduce noise to all variable measurements, thereby enhancing the realism of the generated data. It was further exemplified by leveraging RAYMOND to produce a reference dataset comprising 1,600 normal batches and 90,400 faulty batches, segmented into subsets of varying complexity [20]. The initial subset represented the base case, drawing initial conditions from Gaussian distributions, while the subsequent subsets introduced non-Gaussian variations and batch-to-batch variability, respectively.

DAMADICS (Development and Application of Methods for Actuator Diagnosis in Industrial Control Systems) is a benchmark designed with a combination of a process simulator and authentic datasets depicting electro-pneumatic actuators used in sugar production [22]. Initially created using MATLAB/Simulink, the simulator was enhanced with real-world data obtained from the Lublin Sugar Factory in Poland. The underlying models, meticulously validated using historical data and grounded in principles of thermodynamics and mechatronics, are organized into four primary functional blocks: the positioner, servomotor, valve, and bypass. This comprehensive framework encompasses 19 distinct faults, ranging from positioner and servomotor faults to control valve and general/external faults. DAMADICS is specifically positioned as a benchmark for evaluating fault diagnosis techniques, necessitating a structured sequence of steps for fault detection and diagnosis, as outlined in the original paper. This process involves preliminary assessment and subsequent analysis of fault detectability and isolability across simulated and real data environments. Compared to benchmarks such as TEP and PenSim, DAMADICS offers a greater depth of information due to its integration of real-world data and a comprehensive testing methodology within the simulator. However, its focus remains narrow, concentrating solely on a particular type of equipment within industrial processes [1].

IndPenSim represents a model of the Penicillium chrysogenum fermentation process [2]. In contrast to the traditional PenSim benchmark, IndPenSim represents an industrial scale, accommodating volumes of up to 100,000 L compared to the typical 100 L vessels in PenSim. Moreover, it is supplemented with real historical data and uses a structured model developed by Paul and Thomas in 1996 [23]. This model accounts for various intricacies including growth, morphology, metabolic production, and biomass degeneration, thereby offering a more comprehensive representation of the fermentation process. The Indpensim process diagram is depicted in Figure 2 , has sensors monitoring key parameters such as dissolved oxygen, pH, temperature, foaming, and pressure. Additionally, offline measurements encompass concentrations of penicillin, biomass, ammonia, and phenylacetic acid, enriching the dataset with important process insights. Implemented initially in MATLAB R2013B and subsequently updated to MATLAB R2018B [2]. Furthermore, a standalone dataset comprising 100 batches has been generated and made available in a later work [9], extending the simulation capabilities to include a Raman spectroscopy device. The model affords users a multitude of options, including various control strategies, disturbances on concentrations, and inhibition effects. Potential process faults encompass agitation, aeration, and sensor malfunctions. Moreover, this simulation was validated with the historical data [1]. Table 1 shows an overview of the discussed benchmark simulators/datasets and their operation modes along with their available platforms.

The decision to choose Indpensim over Pensim as the environment for reinforcement learning (RL) in this study is based on Indpensim's richer feature set and higher fidelity. While both are batch processes, Indpensim offers a more comprehensive representation, incorporating a broader

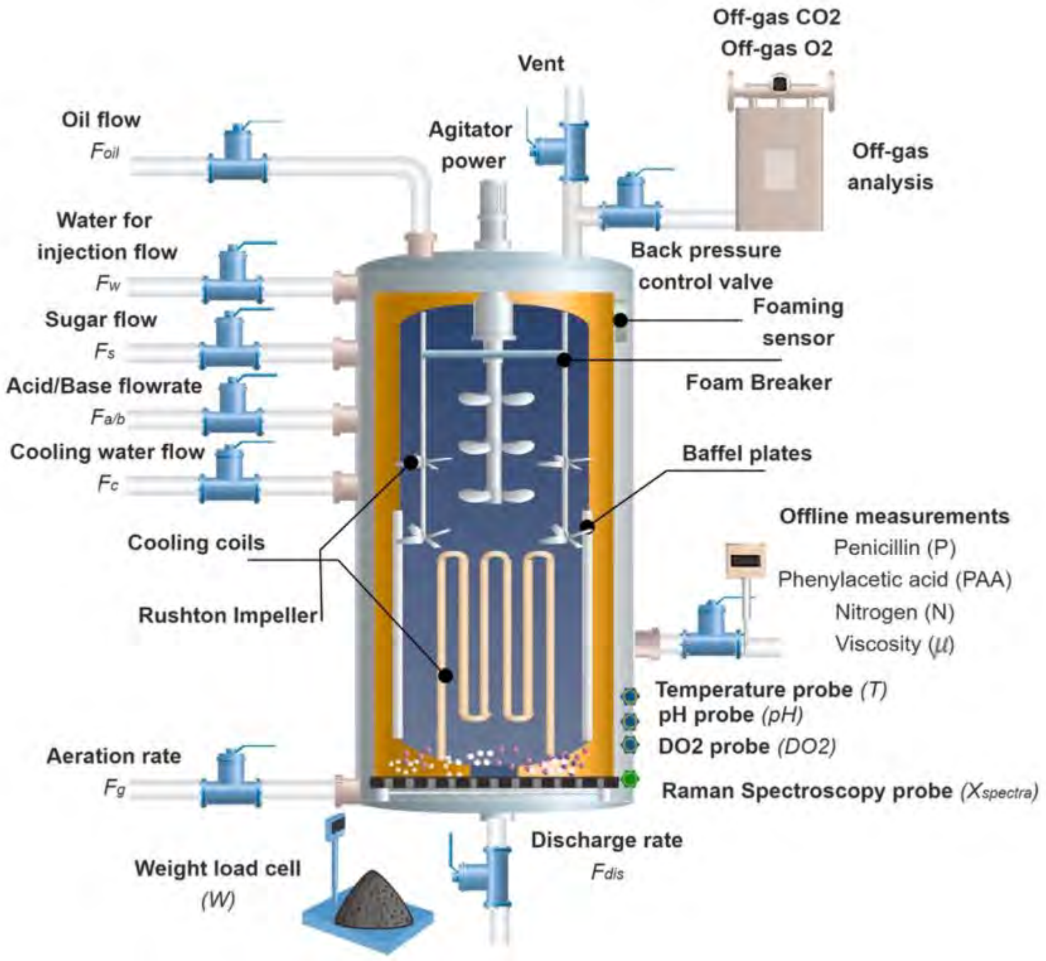

Figure 2: Indpensim process diagram [2].

range of features and complexities. This increased fidelity allows for a more realistic simulation environment, better reflecting real-world industrial processes and enabling a more nuanced analysis of the RL agent's performance.

2.2 Indpensim

This section discusses the Indpenim process, including its development, features, operation of the process, and sensitivity analysis by Goldrick et al. [2]. The Key features modelled such as dissolved oxygen, dissolved carbon dioxide, nitrogen, phenyl acetic acid, temperature and pH are described here.

2.2.1 Development of a realistic simulation

The development of a realistic simulation of an industrial scale fermentation was carried by Goldrick et al. [2] based on prior work [23] where a structured model for penicillin concentration was established. The simulation was further developed to be used as a reliable benchmark for studies in optimization and process control by validating the simulation against fed-batch penicillin fermentation records from ten 100,000 L vessels. Typical issues that arise from large scale fermentation such as prolonged increase of viscosity that can affect the rate of dissolved oxygen, regulating key nutrients particularly when delayed offline measurements are utilized and variations between batches as a result of deviations in input concentrations are considered in the simulation. Additionally, it addresses common process faults such as sensor noises, foaming, agitator malfunctions and others.

The simulation accounts for various phases of the biomass which are growth, morphology, metabolic production and degeneration [2]. To depict the internal structure of the biomass accurately, component balances involving 36 parameters were conducted to characterize actively growing regions A_0, non-growing regions A_1, degenerated regions A_3, and autolyzed regions A_4 (see Figure 3).The remainder of this section delves deeper into the modeling of key parameters and elucidates their influence on the process.

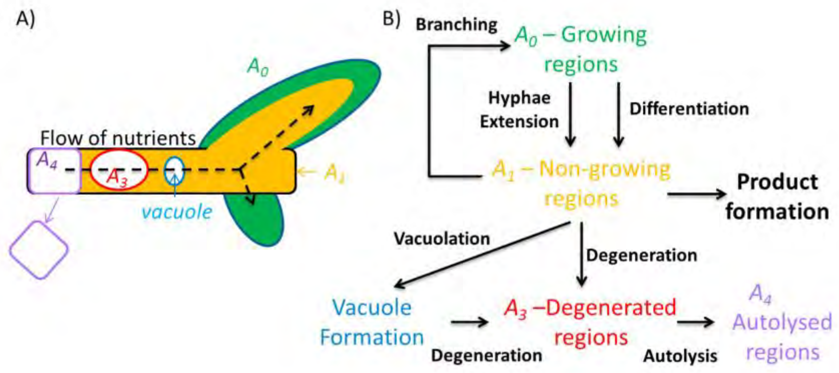

Figure 3: (A) A schematic diagram of the four distinct regions of the Penicillin chrysogenum fungus, with emphasis on the flow of nutrients. (B) Mechanisms of formation for each region [1]

Dissolved oxygen was identified as a key parameter as this nutrient stands as a pivotal macronutrient utilized by microorganisms for growth, maintenance and metabolic production [2]. The equations outlining its utilization were formulated by Bajpai et al. [18] and considered the oxygen consumption during both the growth and maintenance phases of the biomass, alongside the production of penicillin. Incorporated in the equations is a constant which is derived from the geometrical parameters of the vessel and the type of stirrer employed. The significance of the vessel dimensions and design were highlighted in determining the efficiency of dissolved oxygen utilization [2]. Maintaining the concentration of dissolved oxygen above a critical level was crucial, as falling below this threshold lead to a drastic decline in biomass growth and metabolic production, ultimately risking batch failure. The investigation conducted by Vardar et al. [24] explored the determination of these critical values and found that when air saturation fell below 30%, the specific growth rate of penicillin experienced a sharp decrease, and dropping below 10% air saturation resulted in irreversible impairment of penicillin production. Similar findings were reported in [25]. This physical phenomenon was also incorporated into the model through the utilization of a hyperbolic tangent function.

Dissolved carbon dioxide (CO_2) was also identified as a critical parameter [2]. The aggregation of CO_2 in the broth has been identified as detrimental to both cell growth and productivity [26]. The concentration of dissolved CO_2 is influenced by the volumetric mass transfer coefficient, which is dependent on the geometrical parameters of the vessel and the impellers. Additionally, factors such as pH and vessel pressure impact dissolved CO_2 levels [27]. However, for this particular model, the effect of pH was deemed negligible as it remained constant. Operators manually controlled dissolved CO_2 levels by monitoring the percentage of CO_2 in the off-gas, thereby approximating periods when dissolved CO_2 was excessively high and implementing corrective measures. Similar to dissolved oxygen, if the concentration of dissolved CO_2 exceeded a critical threshold, the specific growth rate of biomass was modeled as zero, employing a hyperbolic tangent function.

Nitrogen, identified as a critical parameter, played a crucial role in both growth and metabolic production, ranking as the second most abundant nutrient in fermentation media [28]. Typically, the primary nitrogen source for each batch was initially present in the starting media and gradually consumed throughout the process. The simulation takes into account the nitrogen composition of input feeds, such as the flow rates of Oil and Phenyl Acetic Acid, along with ammonia sulfate salts added to rapidly elevate nitrogen concentration. The significance of nitrogen as a pivotal macromolecule for industrial penicillin production was highlighted in [29], where a decrease in biomass growth was observed during periods of nitrogen limitation. Similar findings [2] illustrated a decline in biomass growth rate for nitrogen concentrations below 200 mg/L. To depict this relationship, the simulation assumed that if the nitrogen concentration remained above a critical value, the specific growth rate remained unaffected. However, for nitrogen concentrations below this threshold, the biomass concentration was modeled to approach zero, utilizing a hyperbolic tangent function ranging from 0 to 1.

In certain fermentations, the addition of a precursor is necessary to ensure the metabolic production of the required product.This is notably significant in the industrial case study [2], where phenylacetic acid (PAA) is introduced to provide the required side chain for penicillin synthesis. Research by Dirk et al. [30] has examined the uptake rate of PAA, revealing that it penetrates the cell membrane via passive diffusion and is influenced by environmental factors such as pH. However, due to limited data availability, the developed simulation simplifies the PAA uptake rate by considering biomass growth, penicillin production, and penicillin maintenance, while ignoring environmental conditions. Effective control of PAA flowrate poses a major challenge in penicillin fermentations, as elevated levels of PAA within the culture can be toxic to the biomass, inhibiting both growth and penicillin production [30]. Goldrick et al. [9] demonstrated that PAA concentration as a critical parameter was maintained within an optimal range of 200 to 2000 mg/L across all ten batches by manipulating the flow rate of PAA.

Viscosity in fermentation broth is intricately linked to its rheological properties. For filamentous fungi, viscosity is influenced by its morphology, which can be characterized as either pellet or filamentous [10]. The pellet form arises from branching hyphae intertwining to form stable aggregates, resulting in lower viscosity. However, nutrient limitation within pellets can pose challenges. Conversely, the filamentous form arises from the extension of long branching hyphae, resulting in a complex three-dimensional structure, which in turn leads to higher viscosity. The model presumed that the concentration of the branching region, associated with the filamentous form, was the sole determinant of viscosity. To establish a link between the growth of the A_0 region depicted in Figure 3 and viscosity, the growth of this form was characterized by a "cube root" growth relationship. This choice was informed by prior studies discussing the growth pattern observed in fungal pellets [31]. Additionally, a growth lag was incorporated into this relationship to better reflect the perceived lag in viscosity reported for this process. This lag growth model, as demonstrated by [32], accurately models microbial cell growth in batch cultures. Furthermore, it was observed that viscosity reduced with the injection of water during the process. Similar behaviour was also observed from batch records of the real process. Therefore, this effect was also incorporated into the model [2].

Maintaining control over temperature is crucial in industrial-scale bioreactors, as fluctuations can significantly impact microbial activity [33]. The dynamic expression for temperature was derived through an energy balance analysis of all major process inputs and outputs of the bioreactor. To develop a realistic model of heat dissipation via cooling coils, it must be correlated with the two primary manipulated variables: coolant flow rate and inlet coolant temperature, as demonstrated by Gulnur et al. [19].

Even minor pH fluctuations can significantly impact fermentation processes, with deviations as small as 0.2 or 0.3 potentially disrupting a batch [28]. Consequently, pH was meticulously controlled around an optimal set-point. pH modeling involves performing a hydrogen ion [H+] balance, which takes into account the generation of hydrogen ions throughout growth, metabolic production, maintenance activities, as well as acid/base additions and other process inputs. It was examined that the pH control was exceptionally precise, with a calculated standard deviation of 0.06 from the pH profiles of the ten industrial batches studied [2]. Building upon this observation, the simulation assumes that any pH values beyond the calculated standard deviation could be detrimental to biomass production. To reflect this, adjustments were made to the pH inhibition terms resulting in a much narrower pH inhibition range of 6.5 ± 0.2, aligning more closely with pH operating constraints.

Off-gas analysis entails inspecting the exhaust gas exiting the vent of the fermenter. This non-invasive process avoids compromising the sterility of the culture, presenting a distinct advantage over in-situ probes that come into contact with the broth. The method involves tracking the concentrations of CO_2 and O_2 in both the inlet and outlet gas streams.

2.2.2 Description and operation of the real process

The considered industrial-scale penicillin fermentation was conducted in a 100,000 L bioreactor, employing an industrial strain of P. chrysogenum. The bioreactor configuration remained consistent with the conventional 100,000 L bioreactor described in [33].

The bioreactor featured a tank radius (r) of 2.1 m and was equipped with three Rushton

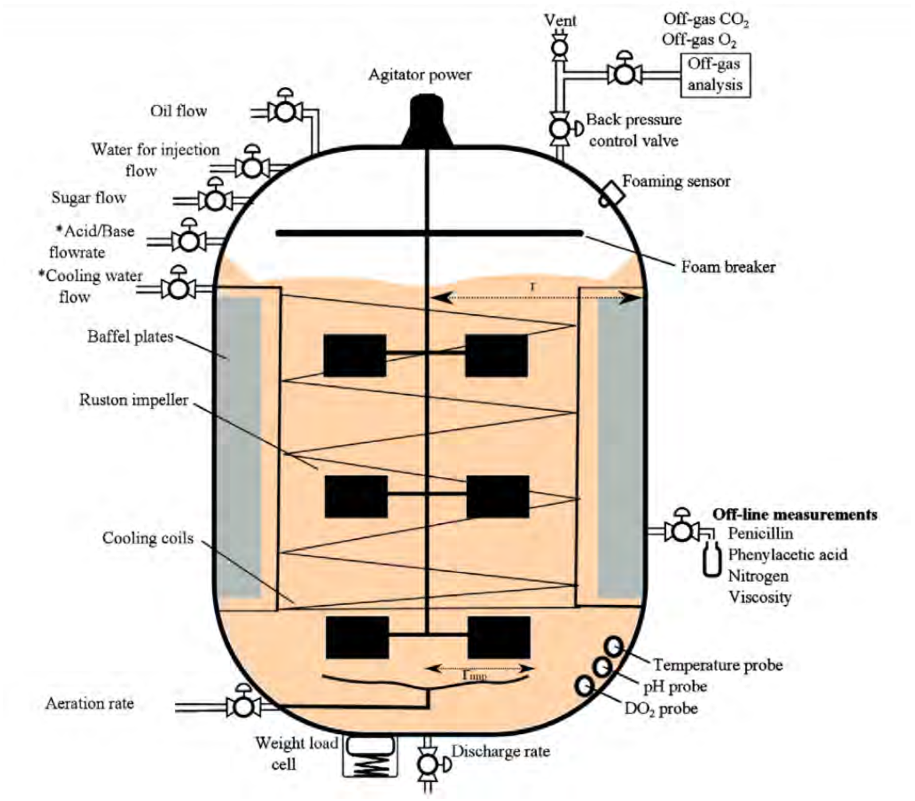

Figure 4: Indpensim process schematic [2].

impellers, each having an internal radius (r_{imp}) of 0.85 m, operated at a constant agitation of 100 rpm. Various sensors including temperature, pH, dissolved oxygen, foaming, and pressure sensors were installed on the vessel. Off-line readings of nitrogen, penicillin and viscosity were taken and collected every 24 hours, while phenylacetic acid measurements were recorded every 12 hours. Additionally, oxygen and carbon dioxide concentrations were measured through off-gas analysis.

The feed rates of both substrate and soy bean oil were governed through sequential batch procedures, adhering to a preset optimal recipe that operators manually fine-tuned according to process dynamics. Similarly, the aeration rate and vessel back pressure were managed using sequential batch strategies to sustain the desired dissolved oxygen concentration and mitigate operational challenges like foaming and elevated CO_2 levels. Additionally, soybean oil functioned as a supplementary carbon source and played a role as an anti-foaming agent.

The adjustment of phenylacetic acid flow relied on offline concentration measurements, while nitrogen levels, initially introduced in the starting medium of each batch, underwent continuous monitoring through offline measurements. In instances of low nitrogen levels, ammonia sulfate shots were administered for rectification.

Temperature was regulated at 298 K, and pH was maintained at 6.5 by their respective PID controllers, with coolant through internal cooling coils and the addition of acid/base solutions respectively. Vessel weight was monitored online using a load cell to schedule discharges, ensuring that the bioreactor's capacity was not exceeded and allowing for longer batches to be achieved.

2.2.3 Sensitivity analysis

A sensitivity analysis was conducted by Goldrick et al. [2] on all simulation parameters to assess the impact of input uncertainties on the simulation outcomes. Certain parameters such as the inlet concentrations of substrate and oil exhibited significant influence across multiple outputs. However due to the lack of measurements, these concentrations were assumed to be constant for the remainder of the analysis [2]. The significance of manipulated variables such as inlet coolant temperature and sparged oxygen inlet gas concentration was demonstrated, as they exerted considerable influence on process outputs such as temperature and dissolved oxygen levels, respectively.

Moreover, vessel dimensions were found to affect specific model outputs, with impeller radius and vessel radius impacting dissolved oxygen concentration. The simulation demonstrated accurate predictions for batches up to 320 hours, surpassing previous simulations validated with shorter batch lengths (160 hours [23], 90 hours in [34] and 200 hours in [35]).

2.2.4 Control strategy

The control strategy implemented standardizes input flow rates according to a predefined optimal profile (Figure 5). Notably, a marked rise in substrate flow occurs uniformly by hour 20, ensuring an abundance of substrate concentration to sustain the biomass in its initial "rapid-growth" phase. Following this, a reduction in substrate flow facilitates the transition to the productive phase of the batch, where substrate concentration is minimized to optimize penicillin production, aligning with findings in [18]. Furthermore, the simulation integrates PID controllers to oversee temperature and pH regulation.

One limitation of the proposed simulation is its capacity to precisely forecast dissolved oxygen and viscosity. Process measurements obtained from a single sensor may not offer a comprehensive representation of the entire bioreactor, potentially introducing errors into the model, which assumes a perfectly mixed vessel.

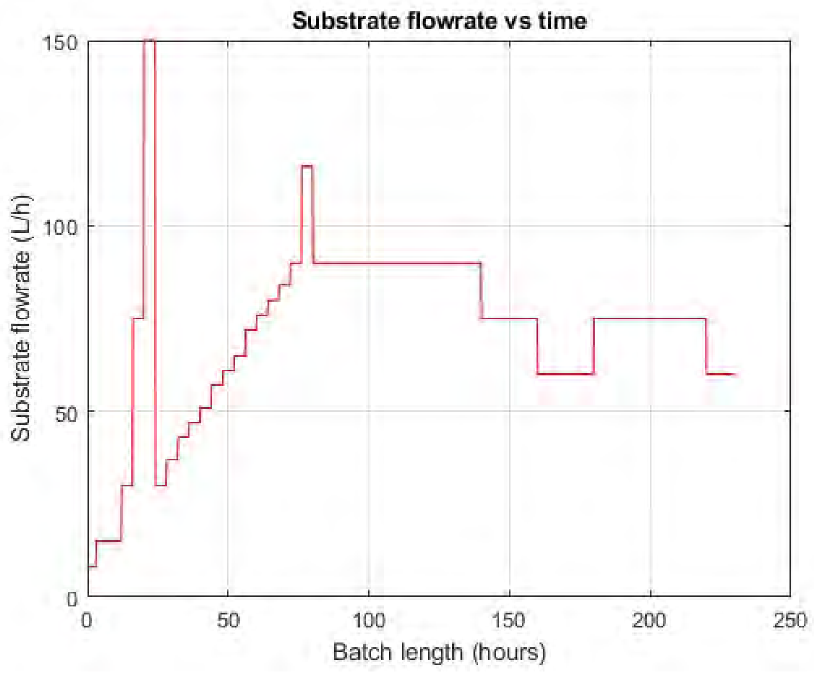

Figure 5: Substrate flow profile.

Structured models are frequently deemed suitable only for highly instrumented laboratory-scale bioreactors, where stable, highly sensitive, and accurate measurements can be obtained [36]. However, this simulation effectively showcases the capability of structured models to elucidate complex industrial processes utilizing routine and commonly recorded measurements.

2.2.5 Related works based on Indpensim

In the previous section, Sequential Batch Control (SBC) served as the control strategy to maximize yield. Separate PID controllers were employed for temperature and pH control to ensure precision, while online measurements for other key parameters, such as concentrations of PAA and biomass, were lacking. Although some measurements, such as off-gas analysis, could be sampled every 12 minutes, the absence of online tools for monitoring PAA and biomass concentrations presented a challenge. This section explores related works that address these limitations by incorporating online analyzers and additional control loops to enhance process control.

2.2.5.1 Indpensim with simulated raman spectroscopy

The existing industrial-scale penicillin fermentation simulation was extended further by incorporating a simulated Raman spectroscopy measurement [8]. This addition serves the purpose of developing, evaluating, and implementing advanced control solutions relevant to biotechnology facilities. The extended simulation allows multiple modes of operations: Operator controlled and recipe-driven; generating large volumes of realistic fermentation data. The simulation replicates real-world processes by incorporating delays in off-line assay measurements, manual intervention by operators in feeding strategies, inaccurate sensor readings, and random fluctuations in growth and production levels. Moreover, a realistic model of a Raman spectroscopy device has been integrated into IndPenSim to facilitate innovative and sophisticated control strategy.

Each batch is expected to achieve a target production yield of 2000 kg of penicillin. Batches failing to meet this specification are deemed below target, necessitating an investigation into their subpar performance.

Control objectives include:

- Designing a control strategy to optimize annual penicillin production and mitigate batch yield variations.
- Identifying critical process parameters (CPPs) and critical quality attributes (CQAs) that impact penicillin production.
- Developing an improved control strategy for pH and temperature variables to minimize their fluctuations relative to the existing PID control loops.
- Creating a control strategy to adjust one or more of the following flow rates: substrate, nitrogen, or phenylacetic acid, to ensure these variables remain within the acceptable ranges.
- Utilize the spectra captured by the Raman spectroscopy device to create a soft-sensor that enables real-time prediction of phenylacetic acid, biomass, or penicillin concentration.

In order to identify the critical process parameters (CPPs) and subsequent critical quality attributes (CQAs), all the process data recorded underwent analysis using multivariate data analysis (MVDA). This analysis encompassed 26 batches, with 5 of them failing to meet the required target penicillin production yield of 2000 kg. Throughout these 26 batches, the operator controlled the flow rates of substrate and phenylacetic acid. The notable deviations observed in these primary control variables led to significant variations in penicillin and biomass profiles. The data from all 26 batches were utilized to construct a partial least squares (PLS) regression model, with cross-validation employed to ascertain the optimal number of latent variables to retain. The substantial impact of the off-line concentrations of phenylacetic acid suggests that this variable played a crucial role in determining the final penicillin yields and therefore identified as the primary CPP.

Due to the limited frequency of off-line PAA concentration measurements and the subsequent 4-hour delay in assay results, the control of this critical process parameter (CPP) remains suboptimal. To address this issue, a Raman spectroscopy analyzer model was integrated into IndPenSim to explore the feasibility of developing a soft sensor for real-time PAA concentration predictions. To initiate the analysis, a calibration batch was conducted on IndPenSim, during which the simulated process analytical technology (PAT) analyzer recorded a Raman spectrum every 12 minutes. Simultaneously, off-line PAA concentrations were routinely measured every 12 hours and utilized to construct the soft sensor using a partial least squares (PLS) model. The validation batch confirmed the soft sensor's capability to predict PAA concentration in real-time, demonstrating comparability to off-line PAA samples.

Subsequently, in the implementation phase of the PAT framework, a proportional-integral (PI) control loop was introduced to adjust the flow rate of PAA and maintain the PAA concentration

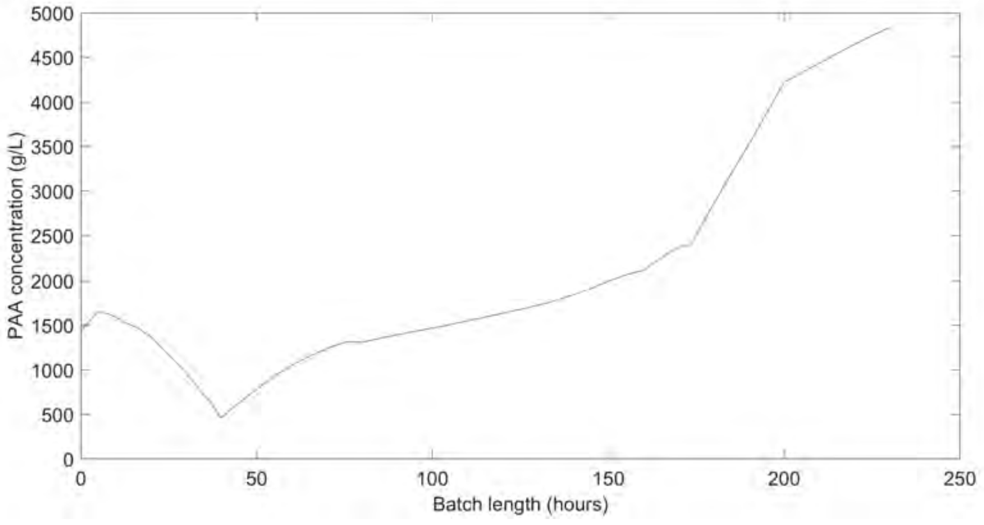

Figure 6: PAA concentration without PI control and raman spectroscopy model.

at its set-point. Initially, the raw soft sensor signal underwent filtering using a three-point moving average to minimize high-frequency fluctuations and avoid unnecessary control actions. The PI controller was activated after a batch time of 25 hours and manipulated the PAA flow rate to sustain PAA concentration at the target level of 1250 mg/L. This advanced process control (APC) solution was applied to IndPenSim over the course of a year, comprising 26 batches. Notably, no batches failed to meet the production target of 2000 kg during this period. The average penicillin yield per batch amounted to 3517 ± 315 kg, representing a significant 20% increase in annual penicillin yields compared to the average of the previous five campaigns. Figure 6 and 7 shows the PAA concentration with and without the previously mentioned APC solution.

2.2.5.2 Alternative control strategies using the PAT analyzer of Indpensim

The use of the PAT analyzer in the form of a simulated raman spectroscopy [8] to predict the concentration of the substrate to achieve better control of the process was investigated [37]. This approach involved generating spectra at 12-minute intervals, serving as a soft sensor. These spectra were then utilized for online modeling of substrate concentration, subject to successful calibration model development. Real-time application of the calibration model allowed for the prediction of substrate concentration, subsequently filtered to reduce prediction noise using a three-point moving average. The filtered signal served as the controlled variable in a proportional-integral (PI) controller, which adjusted the substrate feed rate. The PI controller was activated after 100 hours, indicating the transition from the growth phase to the stationary phase for each batch.

Following simulation validation, the performance of the proposed PI controller system was assessed. Despite some noise in the prediction, the calibration model accurately tracked changes in substrate concentration throughout the batch. The controlled variable set-point was maintained at 5.5×10^{-3} g/L. Implementation of the PI controller led to significant increases in final penicillin yield: 35%, 20%, and 9% for three test batches.

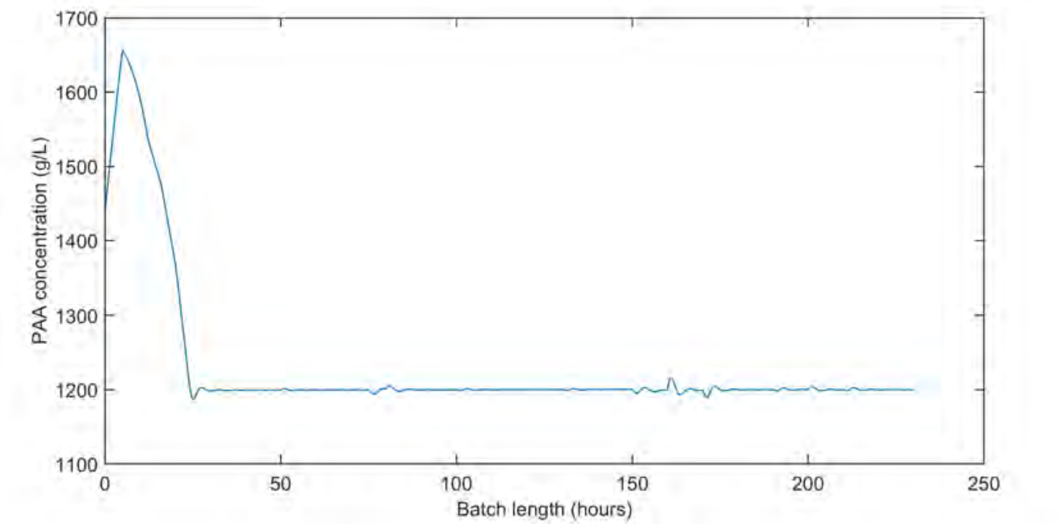

Figure 7: PAA concentration with PI control and raman spectroscopy model.

Moreover, the application of the PI controller strategy for volume control demonstrated notably tighter regulation of the bioreactor level compared to sequential batch control. This tight control is particularly advantageous in large-scale batch fermentations, as large fluctuations resulting from operator-controlled volume adjustments can disrupt other process variables and reduce product concentration.

2.2.5.3 Reinforcement Learning based control

In a very recent study [38] RL was applied as a controller in the Indpensim simulation, aiming to enhance average batch yields and reduce batch-to-batch variations amid uncertainties like fluctuating initial conditions and external disturbances, mirroring the objectives of this study. The employed RL algorithms are namely Deep Deterministic Policy Gradient (DDPG) and Soft Actor Critic (SAC), which adopts an actor-critic architecture. More on this architecture is elaborated in Section 2.4.5.

To mitigate numerical instabilities resulting from exploratory actions by the RL agent, the framework integrated prior knowledge in the form of historical operation recipes, minimizing excessive exploration. To determine the maximum theoretical batch yield, Non-linear Model Predictive Control (NMPC) was implemented, basing its prediction model on the ODEs from the Indpensim simulation. This ideal NMPC disregarded measurability constraints, encompassing online, offline, and unmeasured states within the feedback loop.

Training results of RL agents revealed SAC outperforming DDPG, with the latter exceeding upper bounds for PAA and viscosity. Excessive PAA concentration can inhibit biomass growth and penicillin production [2]. SAC achieved a total yield of 4008 Kg, nearing the maximum theoretical yield of 4042 Kg from NMPC.

Upon evaluation against batches with varying conditions and uncertainties, NMPC and SAC yielded average outputs of 3987 kg (with a variance of 265 kg) and 3944 kg (with a variance of 272 kg), respectively. Online prediction times for a single control step were 3 to 6 minutes for NMPC and 0.01 to 0.03 seconds for the RL agents.

2.3 Drawbacks of classical control strategies

For industrial process control applications model-based predictive control (MPC) and real-time optimization (RTO) represent established technologies especially when decision-making is required in minutes or hours time scale [39]. A significant advantage that comes from model-based control techniques is that the constraints and objective function can be adjusted dynamically, allowing for essential flexibility in rapidly evolving industrial operational settings. These technologies require an accurate complex model which is used along with constraints of the process and economic constraints to determine the most optimal and safe control action. Due to the shifting process dynamics, periodic model re-calibration is required to avoid infeasible solutions or continuously degrading performance from plant-model mismatch. Also, there is a substantial demand for online computational resources and its limited capacity to integrate information regarding uncertainties like parameter inaccuracies and fluctuations [40]. The computational cost of obtaining a converged global solution can escalate depending on the complexity of the model and the number of constraints involved [41] [42] [43].

With RL, a model free approach is possible where learning can take place on a simulation of the real process, enabling the simulation-derived optimal strategy to be applied to the real process and then trained further for fine-tuning (if required). This approach mitigates the potential adverse effects of the learning process on the real system and significantly reduces the computational load during online operations [40]. This approach also known as transfer learning eliminates the necessity for numerous evaluations on the true system, which can be expensive, time-consuming and unsafe. To manage both the unpredictable dynamics, process disturbances and plant-model mismatch, RL is implemented in this study as an alternative to standard control methodologies.

2.4 Reinforcement Learning

Like in section 2.2.5.3, this study will also explore RL based control on the Indpensim simulation to maximize the yield. However the RL algorithms involved, environment configuration and reward function used will be different. This section delves into the theoretical foundations of RL and explores various classes of RL algorithms that are compatible with this simulation.

The three main classes of Machine Learning (ML) include:

- supervised learning
- unsupervised learning
- reinforcement learning

With ML methods such as supervised learning and unsupervised learning, the performance is always limited and cannot outperform the subject matter expert. This is known as bayes error rate [3]. This is because these methods train on the knowledge base from the subject matter expert [44]. With Reinforcement Learning, exploratory actions can generate and train on data points beyond the knowledge base of the subject matter expert and can therefore overcome the bayes error rate.

2.4.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) stands as a dynamic machine learning approach inspired by the way living organisms learn from experience. It involves an agent interacting with an environment, making sequential decisions to achieve specific goals while aiming to maximize rewards [3]. RL operates within the framework of a Markov Decision Process (MDP), where the agent's actions lead to transitions between states, with rewards serving as feedback for its decisions. Through a balance of exploration and exploitation, RL algorithms seek to determine the most effective policy for decision-making in diverse scenarios [3]. RL finds application across various fields, including robotics,[45] gaming [46] and industrial control systems [10]. Taking the chemical process domain as an example, an agent observes a range of states that describe the current process (such as temperature, flow-rate, pressure,etc.) within an environment (chemical plant or simulation). Based on those states the agent takes actions such as adjusting a manipulated variable/disturbance variable or changing the set point to a controller which leads to new states. A reward (good or bad) is received for taking that action. Using the reward as feedback, the agent iteratively refines its actions over time with the aim of optimizing a long-term objective.

RL combines two active areas of research - optimal control in the form of dynamic programing and trial-and-error from animal psychology [44]. In an effort to control a dynamic system, the optimal control problem was proposed. It involved designing a controller that could effectively reduce the loss function of a dynamic system over time [47]. During the 1950's the contributions from Hamilton and Jacobi was extended by Richard Bellman to address the optimal control problem. He introduced dynamic programming as a way to optimize a sequence of actions using a functional equation (Equation 1) derived from the information of the systems state along with a value function [48] .

$$V(s) = r(s) + \gamma \sum P(s'|s,a) \cdot V(s') \tag{1}$$

The value function $V(s)$ of a state s is the combination of the reward $r(s)$ for that state and the discounted sum of future rewards for visiting the next states s'. The discount factor γ serves to incorporate uncertainty regarding future rewards.

The value function assesses the desirability of a specific state based on the actions taken according to an optimal policy to transition to subsequent states. A higher value denotes a more favorable state. However, dynamic programming encounters a significant drawback: as the number of states grows, so does the computational burden. To address this challenge, approximate dynamic programming (ADP) techniques were introduced [49]. By employing trial and error, the repercussions or rewards associated with various actions initiated from a particular state are discerned. Actions yielding positive outcomes are reinforced, while those yielding negative outcomes are eliminated.

2.4.2 Markov Decision Process (MDP)

The main principle followed during the formulation of an RL problem is to define a system with components that create a Markov Decision Process (MDP) [50]. Those components are:

- $s \in S$, where s is the current state and S is the state space (continuous or discrete)
- $a \in A$, where a is the action taken by the agent and A is the action space (continuous or discrete)

- $P(s_{t+1}|s_t, a_t)$ which represents the transition probability from a new state given the current state and action chosen
- $r : S \times A \to R$, where r is the reward from taking action a and R is the reward space
- π is policy which maps state(s) s to action(s) a for a given time step

In an MDP, the agent uses the above components to learn an optimal policy. At a given state s the agent chooses an action a from an intial policy π that takes the agent to a new state s_{t+1}, collecting a reward along the way for that action. This sequence is repeated, and the agent reaches new states while collecting rewards for every transition. During this process, the agent adjusts it's policy π to maximize the discounted future reward G represented in Equation 2 [3].

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

The discount factor $\gamma \in [0, 1]$ is a parameter that controls the emphasis on immediate rewards or future rewards. When $\gamma = 0$, the agent adopts a 'greedy' approach, aiming to maximize the reward obtained immediately in the next step, while a $\gamma = 1$ setting implies the agent prioritizes all future rewards equally or the statistical expectation of the reward. $\pi(a|s)$ denotes the policy mapping of states to actions, governing the agents behaviour. The state-value, denoted as $v_\pi(s)$ and the action-value, denoted as $q_\pi(s, a)$ can be computed using the bellman expectation equations 3 and 4

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \forall s \in S \tag{3}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a], \forall s, a \in S \times A \tag{4}$$

Equation 3 represents the value function which estimates the cumulative return across all potential transitions from taking actions according to policy π. The value function only takes the current state s as input whereas if the current state s and the following action a is taken, we get equation 4. Once the value functions are estimated for each state, the optimal value functions can be determined by equations 5 and 6. Finally, the optimal policy is calculated using equation 7.

$$v^*_\pi(s) = \max_\pi v_\pi(s), \quad \forall s \in S \tag{5}$$

$$q^*_\pi(s, a) = \max_\pi q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v^*_\pi(S_{t+1})|S_t = s, A_t = a], \quad \forall s, a \in S \times A \tag{6}$$

$$\pi^*(s) = \text{argmax}_u q^*_\pi(s, a) \tag{7}$$

Given a simple MDP, the value functions can be computed in a tabular format known as Q-table and be optimized to produce a policy that maximizes future reward [51]. However, this works only when the action and state spaces are discrete. When dealing with a continuous space for actions and observations, discretizing the space will lead to optimizing very large Q tables which is very complex due to the curse of dimensionality [52]. This can be solved by using function approximators such as neural networks. The combination of RL integrated with neural networks led to the development of Deep RL.

An MDP is identified as a Fully Observable Markov Decision Process (FOMDP) given that all the states of the system are observable. However, in reality it is not always the case. In industrial processes not all states or variables can be fully measured due to hardware limitations or other constraints. This leads to a Partially Observable Markov Decision Process (POMDP), where the agent at a given time step can only observe a set of possible observations rather than the full state, unlike in a Fully Observable MDP (FOMDP). These observations typically come from available sensors in the context of process control. To handle unmeasured states, probabilistic inference methods like the Kalman filter can be used to approximate them using the available observations [44]. However, finding the optimal policy for a POMDP is much more challenging than for a FOMDP, even when all the value functions are known, as not all current states are observable [44] [53].

To optimize behavior within a Partially Observable Markov Decision Process (POMDP), one effective approach is to employ belief states. These states, expressed as probability distributions over possible states, represent the agent's inferred current state based on past observations and actions. By computing value functions for each state-action pair using these probabilities, agents can make optimal decisions, effectively converting the POMDP into a Fully Observable MDP (FOMDP). This strategy shares similarities with observer design in control theory, where techniques like the Kalman filter are utilized to estimate unknown states using probabilistic or data-driven models. In reinforcement learning (RL), recurrent neural networks (RNNs) serve a similar purpose by estimating belief states [44]. Previous studies have conducted comparisons between RNNs and Kalman filters, revealing similar performances in this context [54].

In reinforcement learning (RL), three primary techniques are employed to approximate the value and action-value functions: Dynamic Programming, Monte Carlo methods, and Temporal Difference (TD) learning. Of these, dynamic programming stands out as the sole class of algorithms that demands transition probabilities or a perfect model of the system. This prerequisite significantly escalates computation costs, particularly in scenarios featuring a vast array of states, as seen in industrial process control.

On the other hand, Monte Carlo (MC) and Temporal Difference (TD) methods do not demand a perfect model as they approximate the dynamic programming solution. MC methods establish the optimal policy by averaging the value function across multiple sampled trajectories of states, actions, and rewards. Nevertheless, the variability within these trajectories leads to notable variance in the final outcomes. Furthermore, MC methods delay the calculation of averages until the episode has concluded.

In contrast, TD learning integrates elements from both MC and DP. Unlike MC methods, it doesn't wait until the end of the episode to adjust its estimates; instead, it engages in mid-trajectory learning. However, because TD learning depends on future estimates or bootstrapping, it is prone to high bias. The next 3 sections go into more detail about these algorithm classes.

2.4.2.1 Dynamic Programming

Dynamic programming algorithms, when equipped with a reliable model, possess the capability to discern optimal policies. However, due to their extensive computational demands, they are often avoided. Dynamic programming approaches complex problems by breaking them down into more manageable components and addressing them individually. To avoid redundant computations, the solutions are stored. Consequently, resolving problems with high dimensionality and complexity requires substantial computational resources.

Among various DP methods, the most popular ones include value iteration and policy iteration. Policy iteration comprises two primary steps that alternate between each other: policy evaluation and policy improvement. In the policy evaluation step, the current policy undergoes evaluation by estimating its value function through iterative solution of the Bellman optimality equation until convergence is achieved Equation 8. Initially, the value function of all states is set to zero and is iteratively updated by the successor states obtained from following the current policy. This value function signifies the anticipated return starting from a specific state and continuing with the current policy thereafter.

$$v_{k+1,\pi}(x) = \mathbb{E}_\pi[R_{t+1} + \gamma v_{k,\pi}(x_{k+1})] \quad \text{and} \quad v_0(x) = 0, \quad \forall x \in X \tag{8}$$

Once the value function has been evaluated for the current policy, the subsequent step, known as policy improvement, aims to enhance the policy by selecting better actions at each state. This is achieved by prioritizing actions that maximize the anticipated return according to the estimated value function. Consequently, the policy is adjusted to favor selecting the action with the highest estimated value at each state. These two steps, policy evaluation and policy improvement, are repeated iteratively until convergence is achieved. Convergence is reached when the policy no longer changes between iterations or when changes are negligible. A critical aspect of policy iteration is its guarantee of convergence to an optimal policy, provided there are enough iterations. Nonetheless, convergence might be sluggish, particularly in extensive state spaces, due to the iterative alternation between policy evaluation and improvement steps [3].

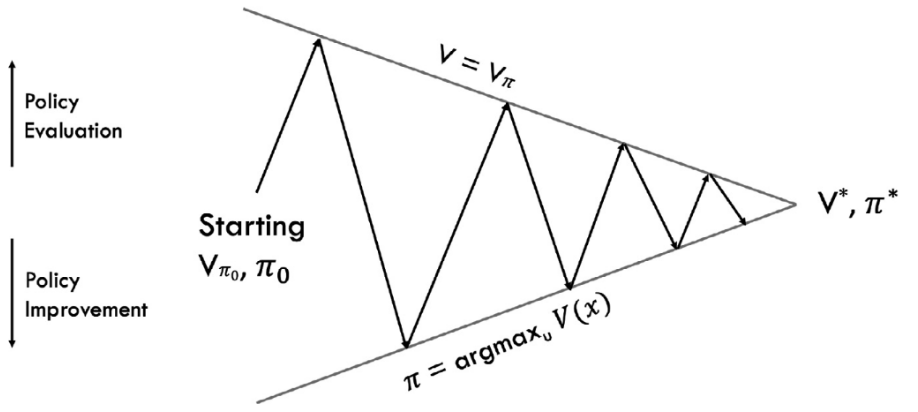

Figure 8: Policy iteration [3]

Value iteration, in contrast to policy iteration, concentrates solely on computing the optimal value function without the need to maintain a policy explicitly. This approach iteratively determines the optimal value functions for each state and then selects actions associated with the highest values to identify the optimal policy Equation 9. For value iteration to be effective, it relies on having an accurate model of the system or environment. This model enables the determination of state transition probabilities, which, in turn, allows for identifying actions with the highest probabilities of transitioning to states with high values. However, in scenarios where a model is unavailable, the action-value function Equation 10 must be determined instead of directly extracting the optimal policy.

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(s_{t+1})] \tag{9}$$

$$q_{k+1}(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a_{t+1}} q_k(s_{t+1}, a_{t+1})] \tag{10}$$

Both policy and value iteration utilize bootstrapping to enhance data efficiency and capture the dynamics of long trajectories, leading to the discovery of the optimal value function $V^*(s)$. These iterative methods ensure that each value is updated using only the maximizing action. Consequently, after convergence, an agent can behave optimally by consistently selecting the maximizing action in each successive state, irrespective of the initial state. Despite their effectiveness, bootstrapping introduces bias in the updates, which is a notable drawback of these algorithms.

In industrial scenarios, dealing with a high number of dimensions is common, posing challenges for the application of policy and value iterations. These traditional methods update all states simultaneously, leading to significant computational expenses. To mitigate this issue, Asynchronous Dynamic Programming (ADP) methods selectively update frequently visited states while avoiding updates to rarely visited ones. While this approach reduces computation time, there's a risk of performance loss if the agent encounters those infrequent states.

2.4.2.2 Monte Carlo methods

Monte Carlo methods, in contrast to dynamic programming, are model-free techniques that do not rely on a system model. Without access to a model for state transitions and value function calculation, exploration becomes essential. These methods determine optimal policies by estimating average returns for various policies through sampling sequences of states, actions, and rewards while exploring under a specific policy. The average returns are updated after each trajectory, making Monte Carlo methods suitable for finite episodes with a terminal state. As samples accumulate, the value function converges toward the optimal value function for all states in the state space.

Unlike dynamic programming, where bootstrapping introduces bias in value function estimation, Monte Carlo methods completely avoid this issue as bootstrapping is unnecessary. The value function is computed from updates that occur after each episode. However, these methods are prone to high variances, especially in noisy systems.

Exploration is crucial in Monte Carlo methods due to the absence of a system model, allowing agents to discover value functions and determine optimal policies. Typically, exploration starts from a random state for each episode until all states are adequately explored over multiple episodes.

Policy search in Monte Carlo methods resembles policy iteration but with three key distinctions: updates are not simultaneous across all states, value functions are updated using sampled data from agent-environment interactions, and action-value functions are identified instead of value functions. Action-values provide explicit information to agents about the expected returns for each action in every state, facilitating effective policy determination.

The optimal policy entails selecting actions associated with the highest expected return in each state. Monte Carlo (MC) methods enable agents to learn through trial and error without needing a system model, provided the episodes are finite. However, this approach has drawbacks, particularly with very long or continuous episodes. Furthermore, it deviates from human learning behavior, as humans typically assimilate feedback immediately rather than at predetermined intervals.

To overcome these limitations, temporal difference (TD) methods integrate concepts from dynamic programming (DP) and Monte Carlo (MC) methods. Unlike MC, TD methods do not wait until the end of the episode for updates but instead rely more on immediate feedback, akin to human learning. By updating value estimates based on the observed reward and the estimated value of the subsequent state, TD methods enable agents to learn from each interaction with the environment. This approach facilitates more efficient learning in continuous systems and better aligns with human learning patterns.

2.4.2.3 Temporal Difference learning

Temporal difference learning, characterized by its computational efficiency and simplicity, has become the preferred class of reinforcement learning (RL) algorithms. These methods combine the benefits of learning from experiences, similar to Monte Carlo (MC) methods, with bootstrapping, a hallmark of dynamic programming (DP) methods. Unlike traditional approaches, TD methods do not rely on a pre-existing model of the system but instead learn the system dynamics directly from collected trajectories.

TD methods update their value functions immediately upon receiving new state and reward information, without waiting for the entire episode to complete. This immediate update allows them to be more responsive and adapt to the dynamics of the environment much quicker compared to MC methods, which wait until the episode is terminated to make updates. The update equations for value and action-value functions are defined in equations (11) and (12) [3].

$$V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{11}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{12}$$

α is a parameter that governs the extent to which the existing value functions are adjusted using the TD errors. These TD errors, represented as δt, quantify the disparity between the agent's perceived value function $R_{t+1} + \gamma V(s_{t+1})$ and the previous value function $V(s_t)$. With multiple visits to each state-action pair, the estimated values gradually converge to their true values. This is identical for the convergence of action-values. Upon achieving convergence, the optimal policy can be derived using Equation 7.

When the agent adjusts its value functions after every action, the algorithm is referred to as TD(0), a particular instance within the broader TD(λ) algorithm, offering a more adaptable learning methodology. Like dynamic programming, TD(0) exhibits notable bias due to bootstrapping. However, as the algorithm advances toward TD(1), bootstrapping diminishes, gradually resembling the Monte Carlo (MC) method. Similar to Monte Carlo methods, TD methods function without a model, requiring the learning of action-values and exploration. Exploration is often conducted through ϵ-greedy policies, wherein the agent randomly chooses actions with a probability of $\epsilon \in [0, 1]$, gradually transitioning towards selecting the action with the highest returns as training proceeds. Initially, ϵ begins at a high value when the agent's knowledge is limited, gradually decreasing as training progresses, and ultimately converging to a low value as training nears completion.

SARSA (State-Action-Reward-State-Action) and Q-learning are two prominent TD methods, each featuring distinct update procedures [3]. SARSA operates as an on-policy algorithm, implying that its behavior policy aligns with its target policy, typically the optimal policy. A behavior policy is the policy that selects actions to interact with the environment during training. This policy could be an ϵ-greedy approach, which involves a trade-off between exploration (choosing random actions) and exploitation (choosing actions based on current value estimates). When both the target and behavior policies align, the agent is considered on-policy. However, an on-policy agent, especially during training, may quickly converge to a local optimum and neglect exploration. This can lead to sub-optimal solutions, as exploratory policies often deviate from optimality. In contrast, for off-policy agents like Q-learning, convergence to an optimal policy is assured as long as each state-action pair is sufficiently visited and the probability of selecting the optimal action under the behavior policy is non-zero. To encourage exploration, Q-learning agents may initially adopt an equiprobable random policy during training to facilitate extensive exploration before transitioning to an optimal policy.

In SARSA, which operates as an on-policy method, the action-value functions are updated using a quintuple comprising the current state, action, reward, next state, and next action, as outlined in Equation 12. Conversely, in off-policy methods like Q-learning, Equation 13 is utilized to update the action-value functions, incorporating only four parameters and excluding the next action, as it is deemed a decision variable for maximizing the action-value function.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s,a) \right] \tag{13}$$

The exclusion of the next action is deliberate as it may differ from the action selected by the target policy. By employing the max operation, Q-values are consistently adjusted towards the optimal policy, ensuring that the update process integrates the optimal next action. Essentially, TD methods amalgamate the benefits of dynamic programming and Monte Carlo methods, enabling agents to learn solely from experiences while also leveraging recent learnings through inter-episode updates.

A crucial distinction between RL algorithms is whether they are model-free or model-based. Model-based algorithms require a prior model of the process as an MDP. This means the transition probabilities P are known and is used to compute value function and policies. Conversely, model-free methods learn policies or value functions directly from interactions with the environment to create its own model. These interactions can be represented as trajectories or experiences in a format (states, actions, reward, next states) similar to that of an MDP. Model free methods can further be classified into value based algorithms and policy-based algorithms. Actor -critic algorithms are the combination of both value based and policy based algorithms. The following sections cover these classes of RL algorithms in more detail.

2.4.3 Value based algorithms

Approaches like Q-learning [55] [51] [56] and SARSA [57], categorized as value based methods, rely on a state-action value function without an explicit policy function. In cases with continuous state and action spaces, this function serves as an approximate state-action value representation. The objective of these methods is to learn the optimal value function by iteratively approximating solutions to the Bellman Equation 10.

Q-learning is characterized by its exploration-intensive nature, ensuring convergence to the optimal policy irrespective of the exploration policy employed, provided each state-action pair is encountered an infinite number of times, and the learning rate is systematically decreased [58].

Unlike the off-policy Q-learning algorithm, SARSA represents an on-policy variant of TD learning [57]. Off-policy algorithms have the capacity to learn about the value of a different policy than the one being executed, whereas on-policy algorithms approximate either the state-value function $V_\pi(s)$ or the action-value function $Q_\pi(s,a)$, which reflects the value of the current policy π being followed [58]. SARSA aims to estimate the optimal policy $\pi*$ by approximating $Q_\pi(s,a)$ for the current behavior policy π across all states s and actions a.

2.4.4 Policy based algorithms

While value-based algorithms indirectly parameterize the policy by estimating state or action values, policy-based methods, also known as direct policy-search or actor-only algorithms, directly store a policy and attempt to update it to approximate the optimal policy $\pi*$ [58]. Model-free and policy-based approaches adjust the parameters θ to increase the likelihood of trajectories τ with higher rewards, thereby enhancing the average return J_θ given by Equation 14. The total accumulated reward for a sampled trajectory τ is defined as Equation 15, and π_θ typically represents a parameterized stochastic policy, such as a Gaussian policy. Various strategies exist for updating the policy, including policy gradient (PG) methods and expectation-maximization-based methods. Here, we focus on PG methods, which employ gradient ascent to maximize the objective in Equation 14. In gradient ascent, the parameter update direction is determined by the gradient $\nabla_\theta J_\theta$, as it indicates the direction of steepest ascent of the expected return.

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau) \mid \theta] = \int \pi_\theta(\tau)\, r(\tau)\, d\tau \tag{14}$$

$$r(\tau) = \sum_{t=0}^{T} r(s_t, a_t) \tag{15}$$

2.4.5 Actor - Critic algorithms

Actor-critic methods merge the benefits of value based and policy based approaches [59] [60]. Similar to policy based methods, they can generate continuous actions. However, they address the substantial variance in the policy gradients encountered in policy based methods by incorporating a critic. This critic's role is to assess the effectiveness of the current policy determined by the actor by evaluating the TD-error using Equation 12. These TD errors, represented as δt, quantify the difference between the agent's perceived value function $R_{t+1} + \gamma V(s_{t+1})$ and the previous value function $V(s_t)$.

The critic estimates and refines the value function through sampled data. Subsequently, the value function updates the actor's policy parameters to enhance performance. Unlike value based approaches, actor-critic methods typically maintain the favorable convergence properties of policy gradient methods. The policy is directly derived from the value function using Equation 7 and is adjusted in the policy gradient direction with a small step size. This approach ensures that changes in the value function result in only minor adjustments to the policy, reducing or eliminating oscillatory behavior in the policy, as detailed in [61].

2.4.6 RL algorithms suitable for implementation

Figure 9 shows how different RL algorithms are classified. For this study, model-free algorithms are utilized as we want the agent to learn the dynamics of the process from scratch. Since continous values are used in the Indpensim simulation, algorithms such as DQN, C51 and QR-DQN are not compatible as they work only with actions with discrete values (discrete action space). An important differentiation between DQN and algorithms such as C51 and QR-DQN is that the latter comes under distributional reinforcement learning methods. In DQN, the action-state value or Q-value is calculated from the expected return whereas for distributional reinforcement learning, the entire distributions of returns is learnt. In C51, the distribution is categorical with a fixed number of

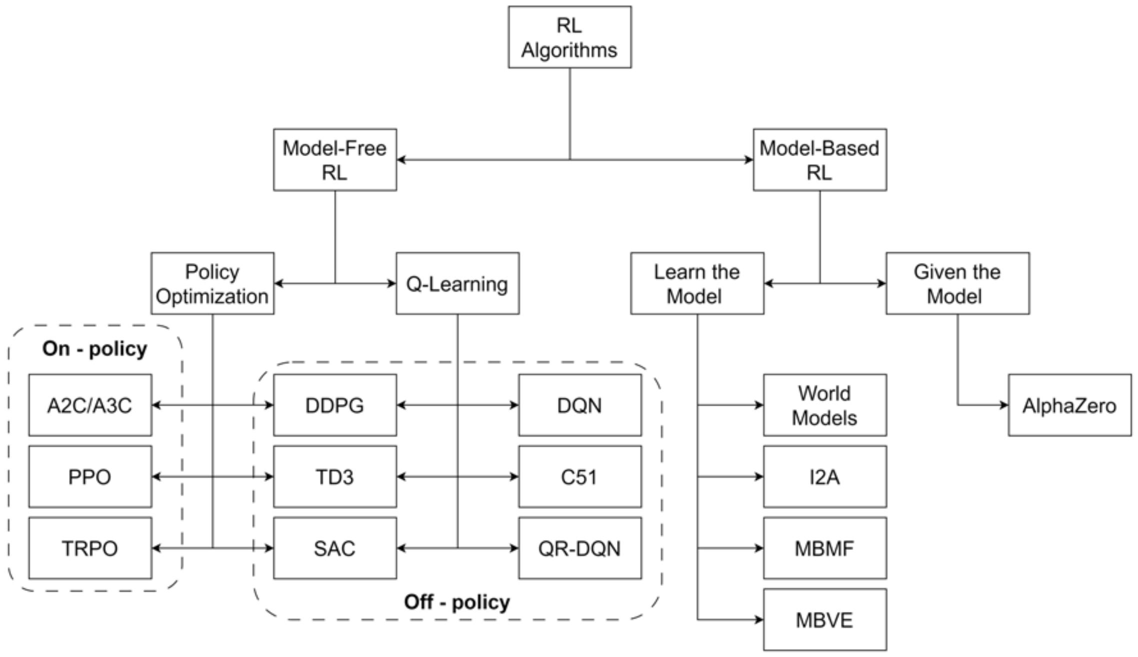

Figure 9: Classification of some state of the art RL algorithms [4].

bins (51 bins) and learns to predict the probabilities under each bin, indicating the likelihood of encountering different return values. Conversely, in QR-DQN, the distribution is shaped using quantiles instead of discretizing the distribution like in C51.

Table 2 summarizes some key features that differentiate between the model-free algorithms. The role of the replay buffer is to store trajectories or experiences so that they can be reused enabling better sample efficiency. It is used with RL algorithms that are off-policy. The main differences between off and on policy is the balance between how the policy is updated, exploration - exploitation balance and how past experiences are used. As seen in Figure 10, an off-policy RL agent has two policies : the target policy and the behavior policy. The role of the behavior policy is to interact with the environment with a random policy and collect experiences. This is how exploration is introduced. A replay buffer stores all the environment interaction as a state, action, reward, next observation tuple. The policy that learns is the target policy. This happens by making updates from randomly sampled experiences. By randomly sampling experiences the correlation between the consecutive updates are reduced and allows for a diverse set of past experiences.

On-policy uses only a current policy as if the target and behavior policy were the same. It immediately learns from the experiences and therefore does not use a replay buffer. Exploration is carried out with the current policy. However, since the policy is updated using recent experiences, the exploration can be limited which makes it less exploratory compared to off - policy algorithms. Since a replay buffer is not used, to collect a large set of experiences multiple workers in parallel are commonly used. Each worker creates a copy of the environment from which experiences can be collected simultaneously. It also improves exploration by exploring a wider range of actions and states therefore reducing the chances of getting stuck in a local optima.

RL algorithm	**Continuous actions**	**Continuous observations**	**Replay buffer**	**Policy type**
Advantage Actor Critic (A2C)/ Aysnchronous Advantage Actor Critic (A3C)	Yes	Yes	No	On-policy
Proximal Policy Optimization (PPO)	Yes	Yes	No	On-policy
Trust Region Policy Optimization (TRPO)	Yes	Yes	No	On-policy
Deep Deterministic Policy Gradient (DDPG)	Yes	Yes	Yes	Off-policy
Twin Delayed Deep Deterministic Policy Gradient (TD3)	Yes	Yes	Yes	Off-policy
Soft Actor Critic (SAC)	Yes	Yes	Yes	Off-policy
Deep Q-Network (DQN)	No	Yes	Yes	Off-policy
C51	No	Yes	Yes	Off-policy
Quantile Regression DQN(QR-DQN)	No	Yes	Yes	Off-policy

Table 2: Features of model free RL algorithms.

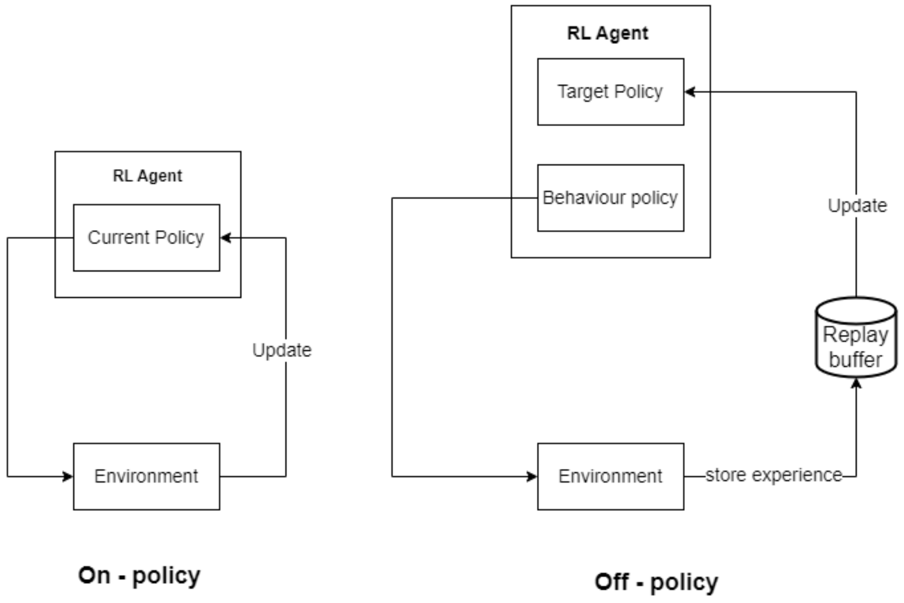

Figure 10: Off-policy vs On-policy RL.

The environment used for this study is more focused on off-policy algorithms that can handle continuous actions and observations firstly because it can be more exploratory than on-policy methods while being decoupled from the learned policy. This is important because bad experiences has much less of an immediate impact on the learned policy whereas for on-policy the impact is more immediate. This narrows down the choices of algorithms to DDPG, TD3 and SAC.Considering that TD3 builds upon an enhanced architecture of DDPG, it renders the inclusion of DDPG unnecessary. For the sake of comparison, an on-policy algorithm such as PPO is also used for training. Additionally, algorithms such as Robust Predictable Control (RPC) which is based on SAC is also used for this study [62].

2.4.7 PPO

PPO is an on-policy actor critic RL algorithm with a main focus on avoiding large policy updates as it easily leads to instability. This is done by limiting the amount of change made to a policy during each training epoch. With smaller updates the convergence to on optimal solution is more likely given longer training.

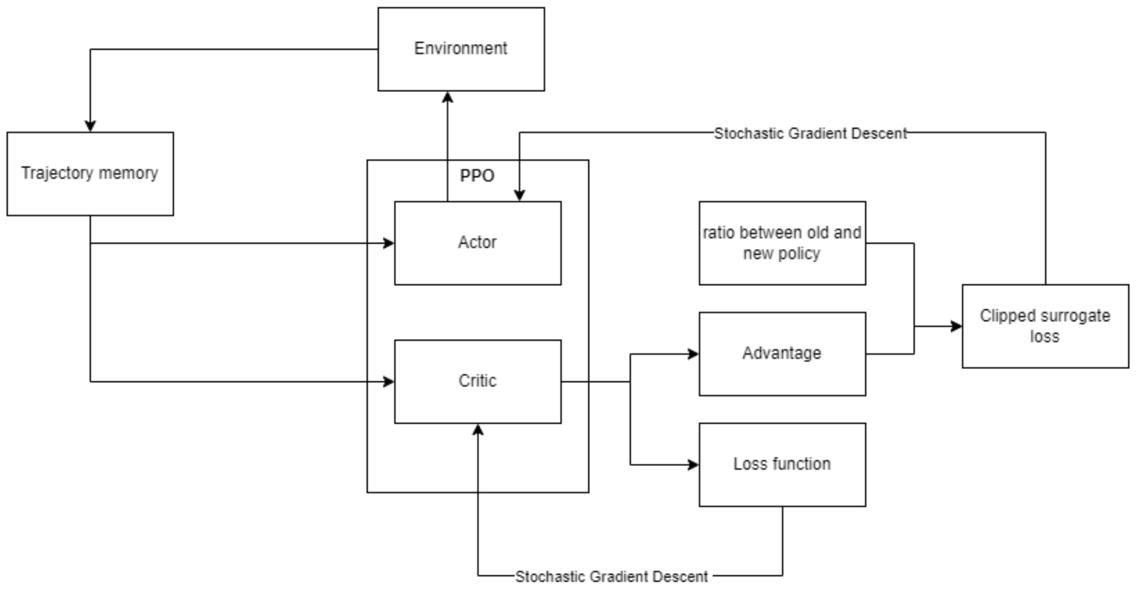
Environment
Trajectory memory
Stochastic Gradient Descent
PPO
Actor
ratio between old and new policy
Clipped surrogate loss
Critic
Advantage
Loss function
Stochastic Gradient Descent

Figure 11: PPO architecture.

One of the hyper parameters used is the clipping ratio. This ensures that the ratio between the old and the new policy lies between a certain range therefore restricting the size of the policy updates. Figure 11 below shows a simplistic version of the PPO algorithm. The agent interacts with environment with actions from the actor and generates experience in the format of (current state, action, reward and next state). The critic evaluates the action be estimating the advantage. The advantage indicates if the action led to a better state compared to the previous state. The loss function of the critic is the TD error. The clipped surrogate loss is calculated by using importance sampling to obtain the expectation of the ratio of sample between the old and new policy and clipped based on the clipping ratio before being multiplied by the advantage. The losses are minimized using stochastic gradient descent.

By clipping the objective function, PPO does not act greedily in selecting actions with a high positive advantage nor quickly avoid actions that give a negative advantage.

2.4.8 TD3

TD3 is an off - policy actor critic algorithm however it does not have a behaviour policy. This is because a beahviour policy is stochastic which mean it gives out a distribution of actions instead of definite continous values for the actions. The prime goal of TD3 is to determine a deterministic policy hence does not require a behaviour policy. The exploration takes place by adding noise to the deterministic actions that come from the actor network.

Figure 12 shows the architecture of TD3. Two critic networks are used to estimate the state-action-value function or Q-value function. The use of two critic networks tackles the overestimation bias which is a drawback from DDPG. Target networks are used for the actor and critic networks and are copies of the main networks. They are updated much slower and periodically using soft updates. These kind of arrangement makes the training more stable.

For the target critic networks, future actions are used for the estimation of target Q values for additional stability to the learning by smoothing the updates to the critic networks. Including the future action makes the target Q-values less responsive to minor alterations in the current policy's actions, which decreases the variability in critic updates and enhances the algorithm's convergence characteristics and makes it robust to noisy environments [63].

2.4.9 SAC

SAC is an off-policy actor critic RL algorithm where the actor network is a stochastic policy unlike TD3 which has a deterministic policy. The aim is to learn a stochastic policy which maximizes the long term reward while simultaneously maximizing the entropy of the actions to encourage a more exploratory behaviour.

To reduce the overestimation bias TD3 incorporated two critics however in SAC a different approach is followed. It penalizes large deviations from the expected value ensuring that the critic network provides accurate and reliable state-action value estimates. This approach along with the use of target networks that undergo soft updates allow for more stable training. The actor network is updated through stochastic gradient descent from TD error loss function evaluated by the critic. Figure 13 depicts the architecture of SAC.

2.4.10 Robust Predictable Control (RPC)

The Robust Predictable Controller (RPC) [64] stands as an RL algorithm built on top of actor-critic algorithms like SAC, prioritizing robustness, generalization, and computational efficiency. It combines concepts from information bottlenecks, model-based RL, and bits-back coding to acquire a model in the latent space and generate compressed policies.

Information bottlenecks restricts the amount of information the RL agent can rely upon, diminishing the risk of over-fitting to the training task. This prompts a shift in the agent's behavior and policy, as minimizing the bits inclines the agent toward states with predictable dynamics.

The algorithm emphasizes temporally extended behavior of policies by leveraging information from one time step to predict information for the next. Given that the predictions are accurate, the

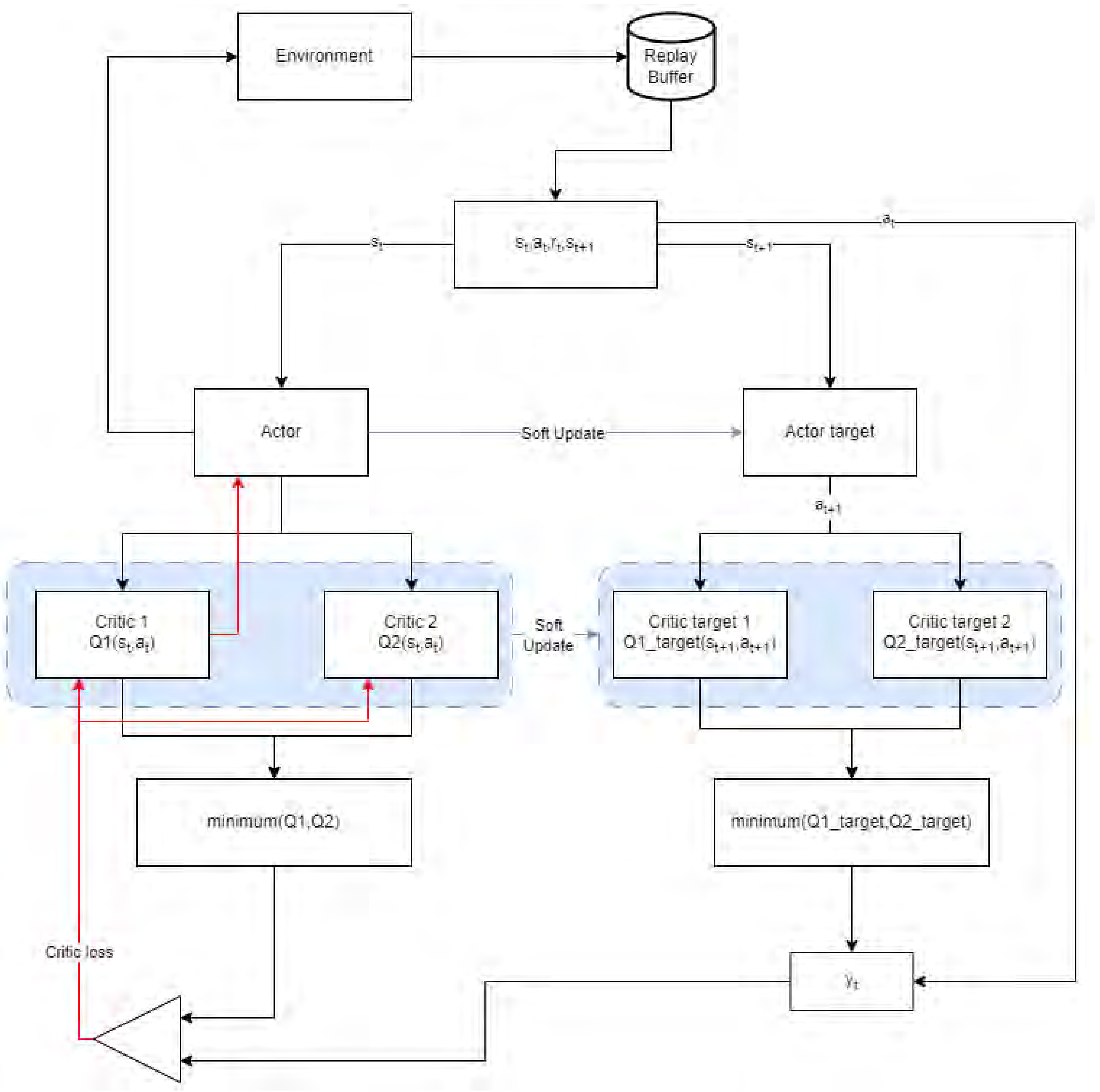

Figure 12: TD3 architecture.

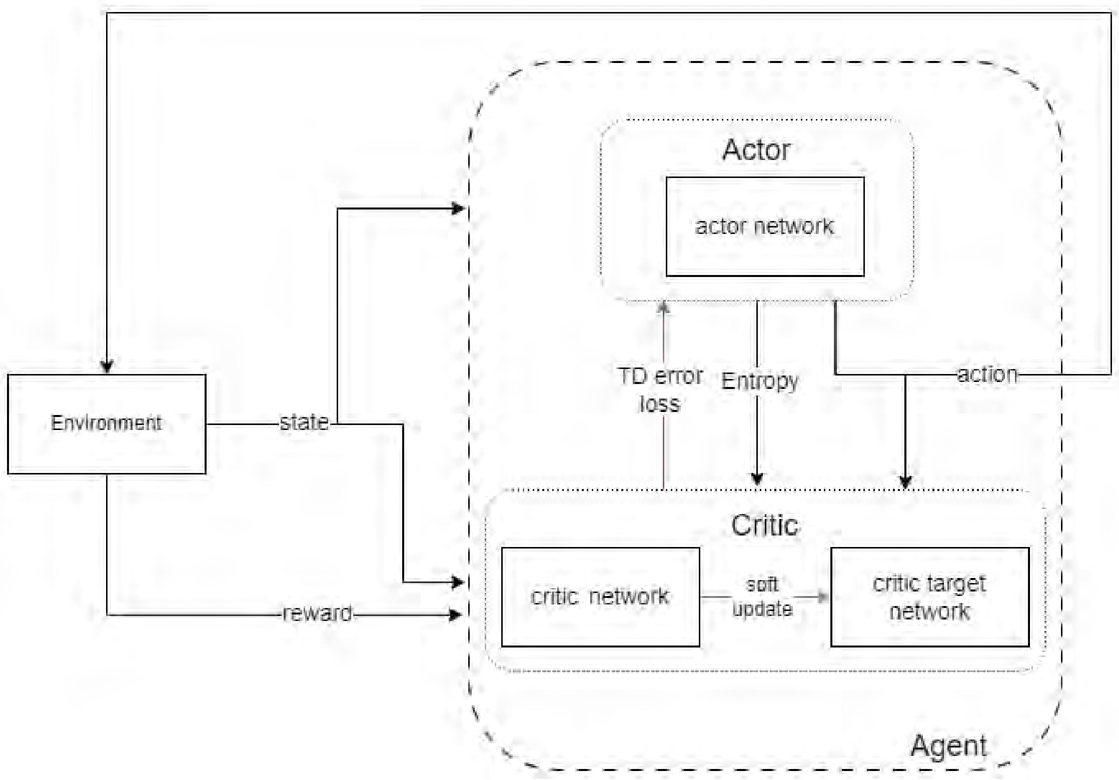

Figure 13: SAC architecture.

agent can rely on information from these predictions instead of obtaining it from the environment. Moreover, the agent can alter the distribution over states by opting for behaviors that traverse states more easier to compress.

The agent's architecture involves learning a policy by training an encoder to generate a representation of the current state and a high-level policy to decode that representation into actions. The aim is to maximize rewards while minimizing the bit count. Compression is applied to the sequence of states through the Variational Information Bottleneck (VIB). States with intricate dynamics necessitate more bits, prompting the agent to favor states where its learned model accurately predicts the subsequent state. Results show that when compared to other actor-critic algorithms RPC learnt policies that more robust to missing observations and noises while achieving similar or higher rewards [64].

3 Environment configuration

This section addresses the configuration of the Indpensim simulation as an environment suitable for the RL agent to train on. The simulation in MATLAB cannot be used directly and has to be configured according to Open AI's gym environment format which is the environment standard for RL [65]. Figure 14 shows a closed loop control of the Indpensim process with the RL agent taking control actions.

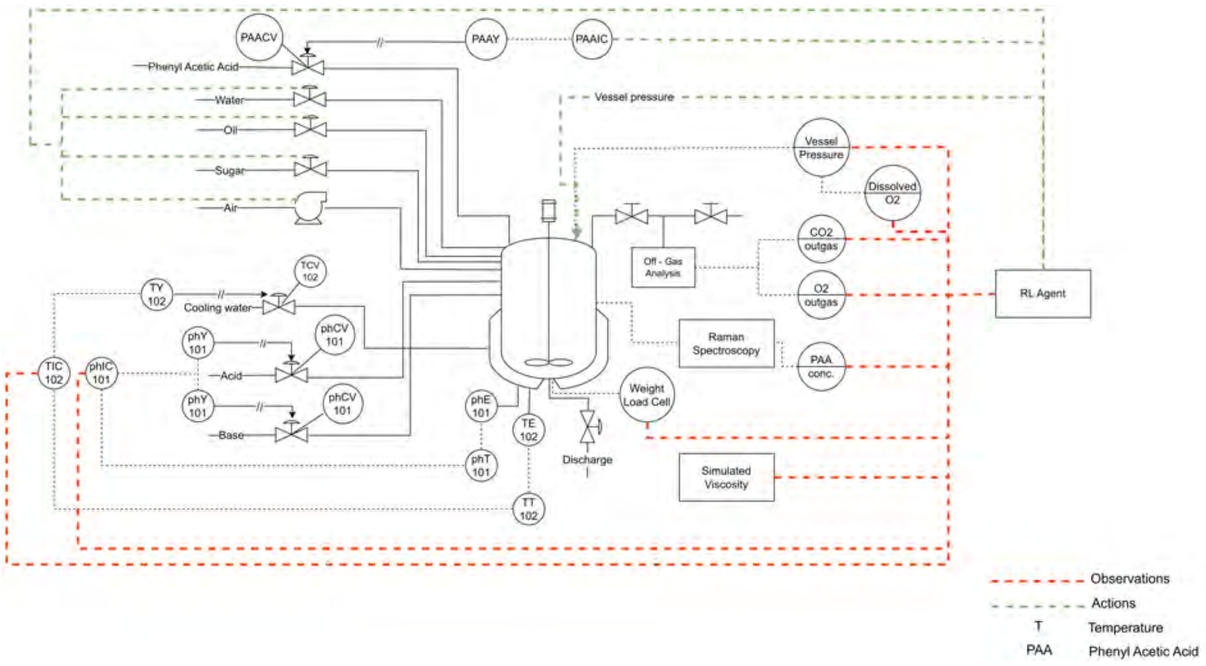

Figure 14: P&ID of closed loop system with an RL agent.

A RL Environment constitutes a simulation or a model where the agent engages by executing actions, receiving rewards or penalties, and transitioning to different states accordingly. The green dashed line in Figure 14 represents the actions taken by the agent on the environment. The actions are listed as follows:

- Manipulate the flowrate of the substrate (sugar)
- Manipulate the flowrate of Oil
- Manipulate the flowrate of water
- Manipulate the flowrate of air
- Manipulate the pressure of the vessel
- Adjust the set point to the PI controller of PAA

The environment in the form a simulation takes in these set of actions to produce observations. For the indpensim, 35 observations are produced, however, for this environment setup, a certain number of important observations which are critical to the process and which can be measured in a real process are chosen which therefore makes this a POMDP. The following observations are chosen:

- Temperature
- Dissolved O_2
- O_2 from outgas
- CO_2 from outgas
- PAA concentration
- Viscosity
- Vessel weight
- Penicillin concentration
- pH

Observations such as the temperature, pressure, pH, O_2 from outgas, CO_2 from outgas and vessel weight are sampled every 12 minutes whereas the rest of the observations were originally sampled every 12 hours. For the RL agent to take actions every hour, it needs these observations as feedback to learn and take the next action. This can lead to missing observations which affect the training performance of the RL agent. Therefore, to have a consistent sampling rate, the simulated Raman spectroscopy and simulated viscosity which has been already implemented in the simulation and validated is activated and used.

The RL algorithms used by the agent are written in python whereas the simulation used for Indpensim is written in MATLAB. For the architecture in Figure 14 to work, both the environment and the agent should be compatible to be able to communicate with each other. To get the environment and agent to work together the following options were evaluated:

- Option 1 : The environment in MATLAB can be used with the RL toolbox of MATLAB

	Python	**MATLAB**
ODE solver	ODEint	ODE45
Penicillin yield after 1 hour (Kg)	0.057	0.058
Penicillin yield after 120 hours (Kg)	1421.28	1808.47

Table 3: Comparison between Indpensim simulation written in python vs MATLAB.

- Option 2 : A python version of the MATLAB environment can be created and used with RL python libraries
- Option 3 : Use the MATLAB engine API to run MATLAB scripts in a python environment and use it with RL python libraries.

The RL toolbox in matlab has a wide selection of algorithms to choose from and provides documentation and examples to create a custom environment. However, option 1 is not used here because there are much more python libraries that offer more choices in terms of algorithms and new ones are updated more frequently. For example, Open AI has algorithms that use RNN and CNN based architectures which are not yet available in MATLAB. Therefore, for this study the option 2 and 3 are better.

To decide which among option 2 and 3 are better, the evaluation criteria used was the similarity of the simulations between the python version and the original matlab version. The python version created in [66] includes same physical ODEs (Ordinary Diffferential Equation), constants and default operator recipes. The only difference was the type of ODE solver used. The ODE solver used in MATLAB is the in-built ODE45, whereas for python Scipy's ODEint was used. This is where differences between the environments were noticed as well as the numerical stability of the ODE solvers.

As seen from Table 3, for a total batch length of 230 hours, the values are close to each other at the beginning of the batch but as it approaches towards the end of the batch, the deviation between the yields increases. Moreover, the python environment was numerically unstable after 120 hours of batch run time. Adjusting the step size of the ODE solver can increase the batch length few hours more than 120 but still not enough to meet the required batch length of 230 hours. Therefore, the only option remaining is option 3 which uses a more reliable and stable simulator in MATLAB and the RL agent in python.

The MATLAB simulation is converted to a gym format environment through the following steps:

- Set up the action and observation space: Every gym environment must have an action space and observation space initialized. The observation space can range from negative infinity to positive infinity, however for this configuration the ranges are possible observations obtained from literature (Table 4. The same applies to the action space and this helps the agent to select actions within these bounds so as to avoid unsafe or illegal actions. The spaces can be initialized in a discrete format or continuous format. Since the simulation works with continuous values, the spaces are initialized as continuous. Table 4 show the upper and lower bounds for the selected actions and observations.
- Set up a reset function: As explained in section 2.2.4, out of the 230 hours of batch operation the first seventy hours always stays the same so that the biomass remains in its "rapid-growth" phase. The remaining 160 hours of batch operation is controlled by the RL agent. By taking

	Variable	**Minimum**	**Maximum**
Action	Sugar flowrate (L/h)	0	160
Action	Oil flowrate (L/h)	0	36
Action	Aeration rate (L/h)	0	76
Action	Water flowrate (L/h)	0	510
Action	Vessel pressure (bar)	0	1.2
Action	PAA concentration set point (g/L)	0	1800
Observation	Vessel weight (Kg)	0	111000
Observation	pH	0	14
Observation	Temperature (K)	0	350
Observation	CO_2 outgas (%)	0	100
Observation	O_2 outgas (%)	0	100
Observation	PAA concentration (g/L)	0	1800
Observation	Dissolved oxygen (mg/L)	0	30
Observation	Penicillin concentration (g/L)	0	50
Observation	Viscosity (cP)	0	100

Table 4: Lower and upper bounds for the actions and observations

an action every hour the episode length for the environment is set as 160 steps. Figure 15 gives an overview of flow of actions, observations and rewards collected between the gym environment and the RL agent. The policy of the RL agents represents a neural network that accepts observations as input and outputs actions based on the input. The role of the policy is to take actions that maximizes the overall reward. The episode is started by first executing the reset function. Here the MATLAB environment executes a predetermined recipe to enable the previously mentioned "rapid growth" phase until 70 hours. From hour 71, the RL takes its first action by executing the step function.

- Set up step function: Depicted in Figure 15, the step function is designed to accept actions from the RL agent as input and returns a tuple consisting of the new observation generated by the simulation, reward and a 'done' flag to indicate if the episode is completed or terminated. In the step function, the action from the policy is passed to the MATLAB simulation using the MATLAB engine API where it executes the new action for a period of 1 hour. New observations such as the ones mentioned in the Figure 15 are generated. These observations correspond to the states for the next hour which is then fed back to the agent to get the new action for the consecutive hour. This process is repeated until the 'DONE' variable is set as True which indicates either that the episode is completed as it has completed all 160 steps or the episode got terminated due to other reasons such as violation of set constraints (Figure 16) in the step function or simulation crashes due to numerical instability that comes from the policy taking extreme actions during exploration.

  The reward is calculated based on the observations received from the simulations executing the actions from the policy. The reward can be positive or negative depending whether constraints were violated. If no violations, then a positive reward is given proportional to the penicillin concentration. The above flowchart shows the constraints corresponding to each observation. Violating those constraints incurs a negative reward and the 'Done' variable can be set as

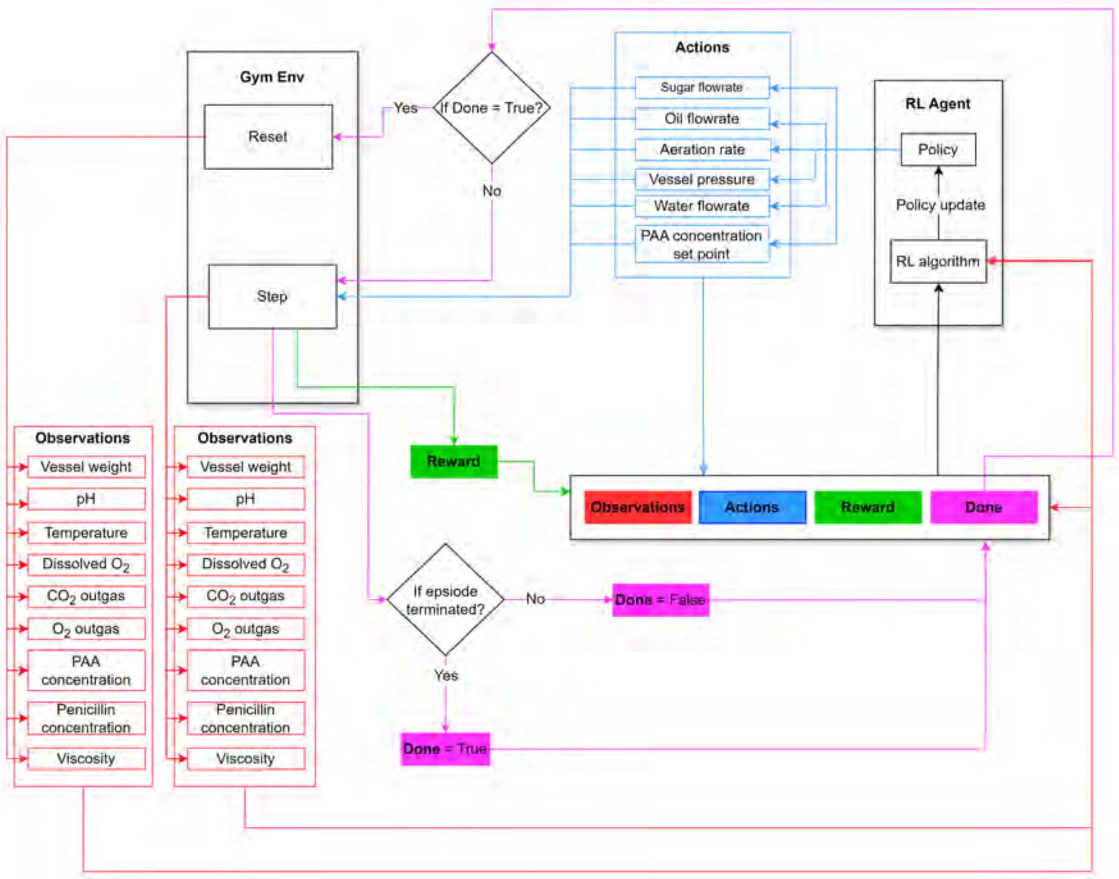

Figure 15: RL environment configuration.

True. Setting it as True terminates the episode and therefore the step function. As shown in Figure 15, the episode is restarted again from the reset function and then the step function is executed thereafter. Setting 'Done' as False, allows the episode to keep continuing. If more constraints are violated along the way, then the negative reward accumulates. This allows the episode to run for the desired number of steps and the constraints in this case become soft constraints. The positive reward function is the penicilin concentration scaled by a factor of 1000. Since we want to also maximize yield, the yield can also be used as a reward function. The impact of using different reward functions is discussed in section 5.

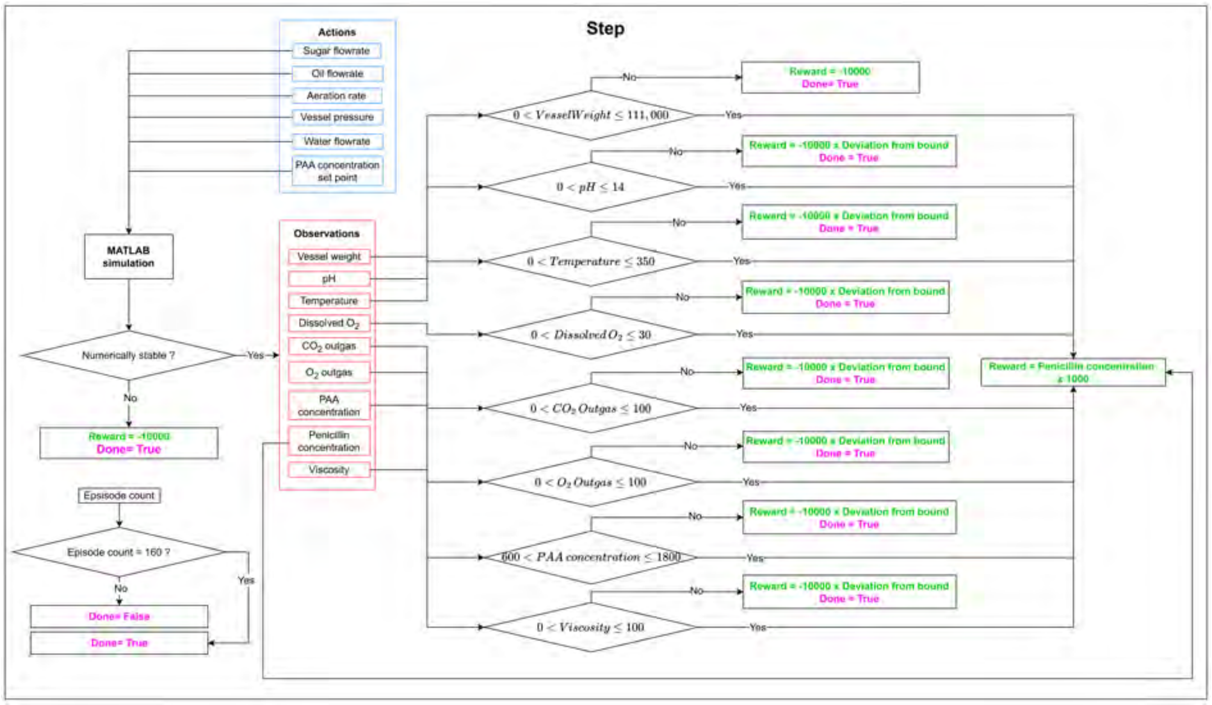

Figure 16: Step function and reward calculation.

4 Results

In this section, the results from training the RL agents discussed in section 2, on the configured environment in section 3 is illustrated here.

4.1 Training with TD3

The TD3 reinforcement learning agent underwent training for 50,000 steps. Analysis of the rewards averaged over 100 episodes, as depicted in Figure 17, reveals an intriguing pattern: the agent achieves its highest average reward approximately halfway through training, around the 25,000th step. However, this peak is followed by a notable decline, after which the average reward stabilizes at a consistent level for the remainder of the training duration.

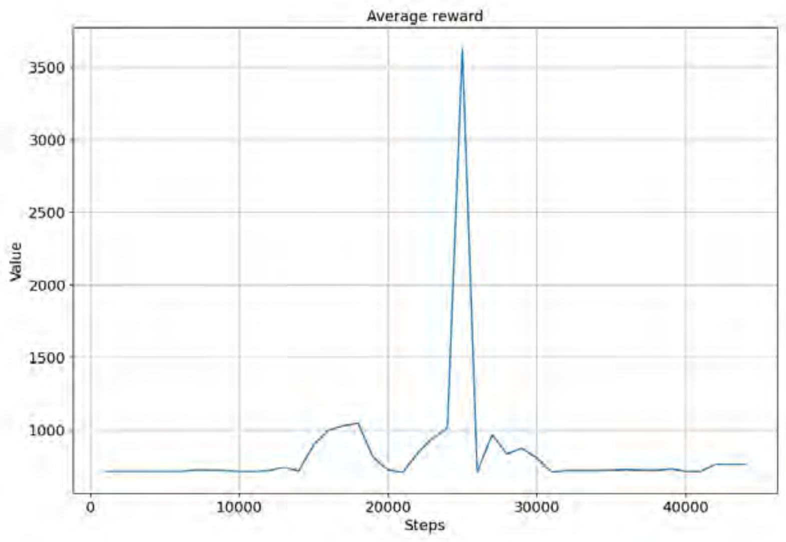

Figure 17: Rolling average reward over 100 episodes (TD3).

The deterministic nature of TD3's policy dictates that exploration is executed through the introduction of noise to the deterministic actions. Despite this mechanism, the observed stagnation in average reward suggests a tendency for the agent to become trapped in local maxima, thereby showing limited exploration. Although hyperparameter tuning offers a potential solution to improve exploration, it warrants attention beyond the scope of this study, which instead prioritizes the examination of alternative algorithms.

Additionally, the critic loss depicted in Figure 18 exhibits high levels of oscillation throughout the training process. This oscillatory behavior signifies a lack of stability in the training dynamics, potentially hindering the agent's learning progress.

4.2 Training with SAC

SAC learns a stochastic policy which therefore is inherently exploratory and unlike TD3 does not need noise to be added to the actions for exploration. SAC leverages parallelization to enhance training efficiency. By employing multiple workers simultaneously, experiences are collected and stored in the replay buffer. This approach not only accelerates data collection but also facilitates more efficient utilization of computational resources, ultimately leading to faster convergence and improved training speed. Each worker maintains a dedicated copy of the environment and obtains actions from the agent's stochastic action distribution. Consequently, multiple experiences are collected concurrently, contributing to a more diverse and efficient exploration of the environment.

For the initial training strategy, 16 workers were considered. Figure 20 shows the reward averaged over 100 episodes of the agent and trained for 100k steps with 16 workers. Around

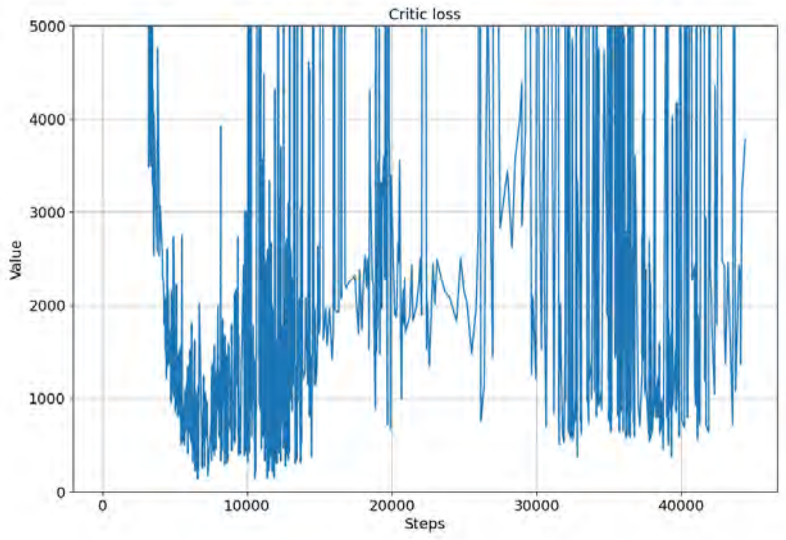

Figure 18: Critic loss (TD3).

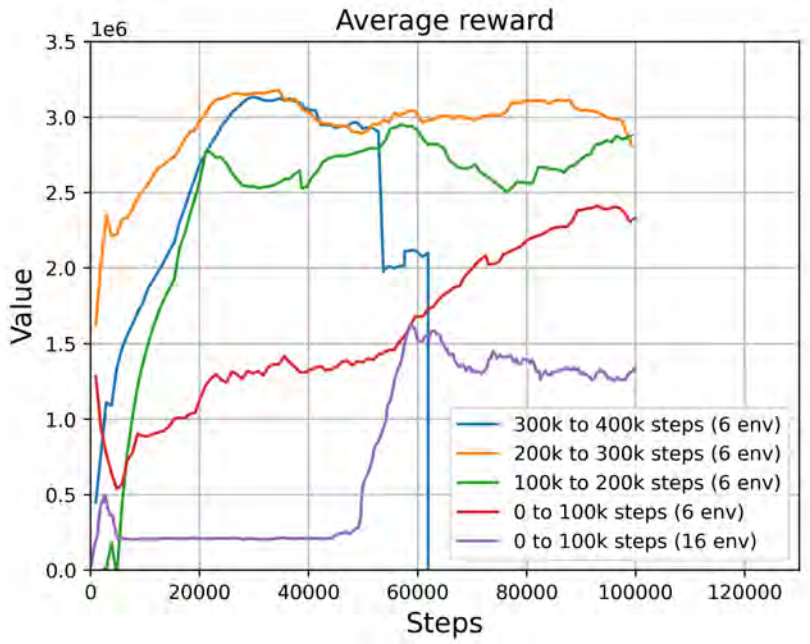

Figure 19: Comparison of average rewards between SAC agent with 16 workers and SAC agent with 6 workers.

the 50,000th step, the agent's exploratory nature helps it avoid getting stuck in the same average reward or trapped in local maxima. This adaptability allows the agent to keep exploring different possibilities in the environment, preventing it from settling too quickly on a sub-optimal solution. Extending training beyond 100,000 steps was not considered because the critic loss started to become unstable, as shown in figure 21.

To evaluate the impact of reducing the number of workers on training performance in terms of average reward and stability, 6 workers were considered, as the environment accepted 6 actions at a time. Figure 19 shows the difference between the rewards averaged over 100 episodes for 16 workers and 6 workers. Note that stability was observed during training with 6 workers from 0 to 100,000 steps. Subsequently, further training was extended until 300,000 steps, after which instability became apparent, as evident in Figure 22. This observation underscores the potential consequences of favoring too much exploration, as it can result in instability and significantly poorer performance.

Therefore, instead of continuing training from 300,000 steps onwards with 6 environments, the decision was made to continue training with just 1 environment. Figure 22 highlights a significant difference in the final yield resulting from the reduction in exploration by reducing the number of environments. This shift also emphasizes exploitation, striking a balance between exploration and exploitation that is crucial for the agent to converge and perform effectively.

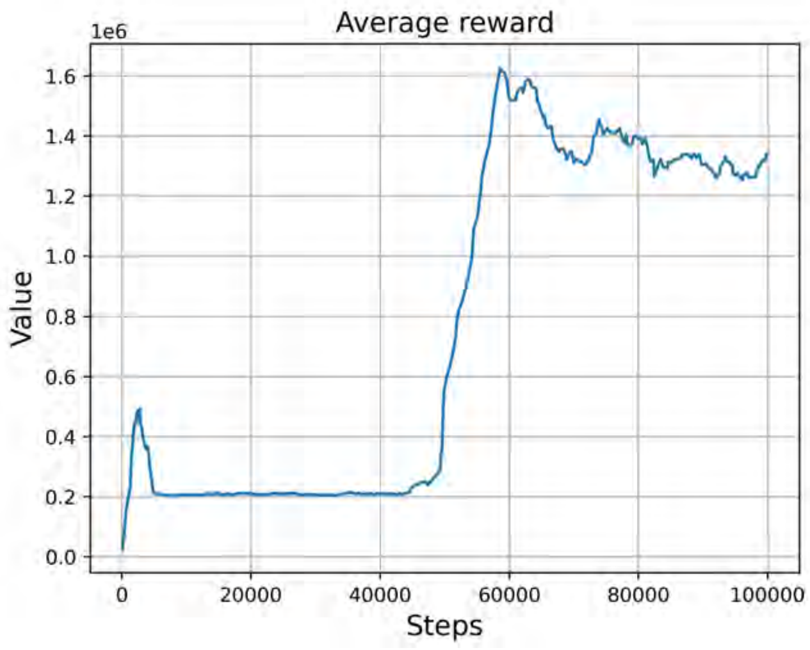

Figure 20: Average reward for SAC agent with 16 workers.

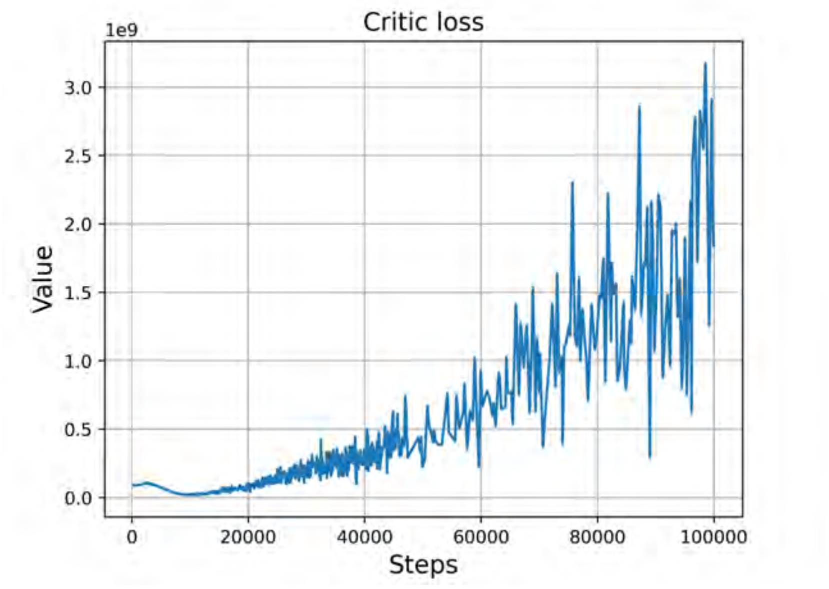

Figure 21: Critic loss for SAC agent with 16 workers.

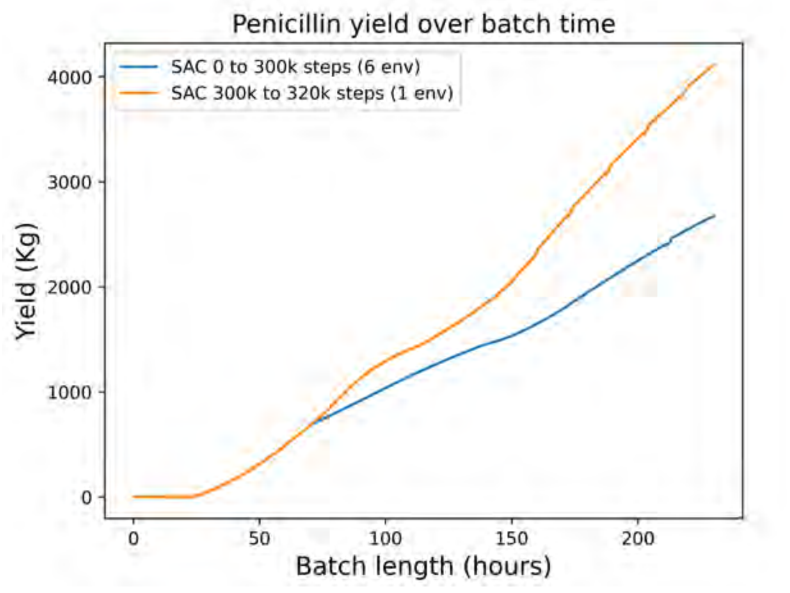

Figure 22: Comparison of yields between SAC agent trained with 6 workers and SAC agent trained with 6 workers initially and then with 1 worker for additional steps.

Exploitation, in this context, meant maximizing rewards by exploiting known states rather than continuously exploring new, unvisited states. This shift in focus allowed the agent to consolidate its learning and refine its strategies based on previously acquired knowledge, ultimately leading to more stable and effective training outcomes.

4.3 Training with RPC

Here the environment is trained with the agent that uses the RPC algorithm. As discussed in section 2.4.10, this algorithm uses information bottlenecks to train compressed policies. The rate of compression can be adjusted by selecting the number of bits. To determine the how the selection of bits affects the training and results, experiments were performed with 15 bits, 10 bits and 1 bit. Furthermore, for each of these experiments, the impact of different reward functions are also analyzed. The two reward functions considered here are the penicillin yield and penicillin concentration which are related to each other by Equation 16.

$$\text{Total penicillin yield} = V_{\text{end}} \times C_{\text{end}} + \sum_{i=1}^{n} V_i \times C_i \tag{16}$$

- V_{end} represents the vessel volume at the end of the batch.
- C_{end} represents the penicillin concentration at the end of the batch.
- V_i represents each vessel discharge volume.
- C_i represents the corresponding penicillin concentration of each vessel discharge.
- n represents the total number of vessel discharges.

Figure 23 shows the impact of the reward function and number of bits chosen for compression on the total penicillin yield. It was observed that for all the cases, having the penicillin concentration as the reward function gave a better performance. The agents were trained for a total of 50000 steps each. The 1-bit and 10-bit RPC agent achieved higher yield than the 15-bit agent irrespective of the reward function. As a result, further experimentation was carried out only with the 1 and 15 bit RPC agents.

The aim of this study is not only to maximize the penicillin yield but also to reduce batch-to-batch variation by rejecting disturbances to the process. This is analyzed by training the agents (1 bit and 15 -bit) with a specific set of initial conditions and process parameters and then validating the trained model against different conditions and process parameters which the agents have not seen before. To demonstrate the sensitivity of the process to slightly different initial conditions, Table 5 and Figure 24 indicates how the initial conditions change and how the performance in terms of penicillin yield is affected.

Figure 24 shows the performance of 1-bit and 10-bit RPC for the train and test conditions. The agents were trained to take actions every one hour (as discussed in section 3). All the agents except the 1-bit did not violate any constraints and thus completed the required batch length of 230 hours. The 1-bit RPC for test conditions terminated the batch prematurely due to violation of the viscosity constraint. Interestingly, when the same agent trained to take actions every 1 hour was instead directed to take actions every 2 hours, it performed better when evaluated under test conditions without requiring any retraining. The reasoning behind this behaviour is discussed in

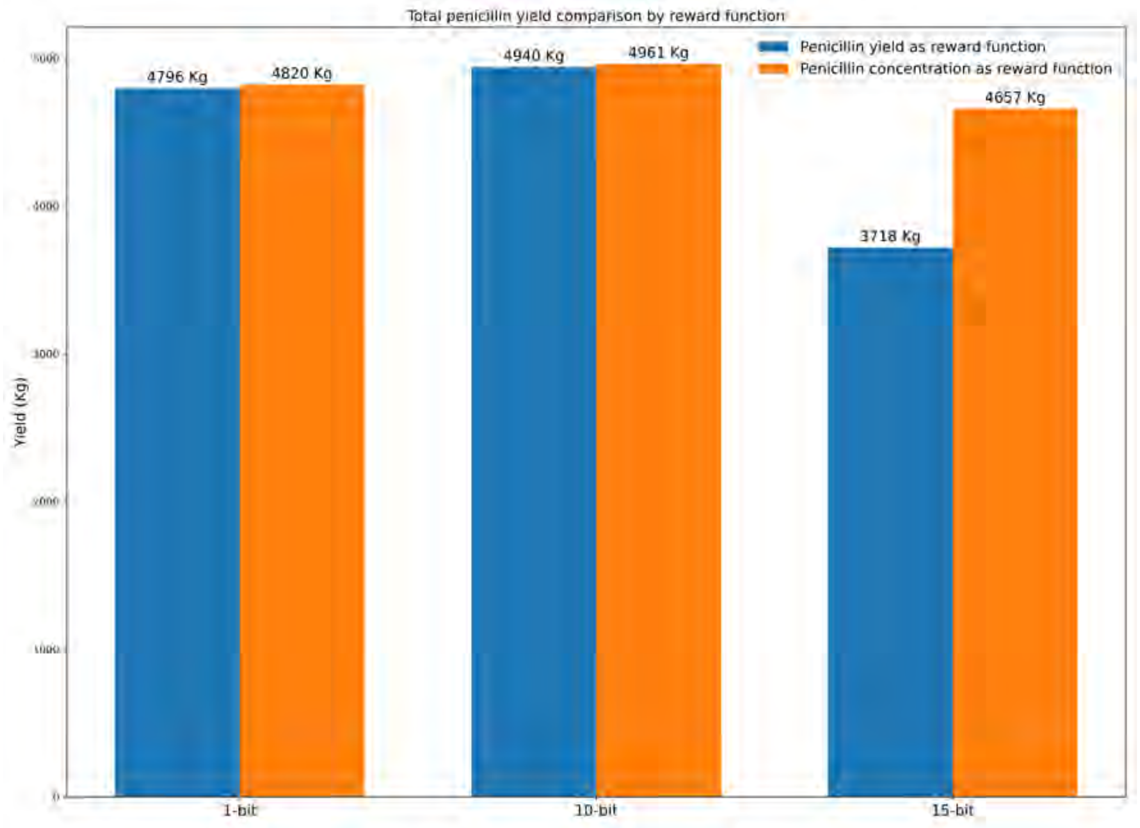

Figure 23: Comparison of penicillin yields for different compressions (bits) and reward functions.

	Train conditions	**Test conditions**
Initial substrate feed concentration	0 (g/L)	0 (g/L)
Initial dissolved oxygen concentration	14.67 (mg/L)	14.58 (mg/L)
Initial biomass concentration	0.45 (g/L)	0.60 (g/L)
Initial vessel volume	57700.02 (L)	57332.02 (L)
Initial vessel weight	61997.46 (Kg)	61662.32 (Kg)
Initial pH	6.60	6.45
Initial temperature	297.67 (K)	297.91 (K)
Initial type a0 biomass concentration	0.15 (g/L)	0.21 (g/L)
Initial type a1 biomass concentration	0.29 (g/L)	0.39 (g/L)
Initial type a3 biomass concentration	6.77 (g/L)	1.11 (g/L)
Initial type a4 biomass concentration	4.05 (g/L)	6.65 (g/L)
Initial cell culture age	0.09 (hours)	0.12 (hours)
Initial PAA concentration	1445.62 (g/L)	1254.87 (g/L)

Table 5: Initial conditions for train and test case.

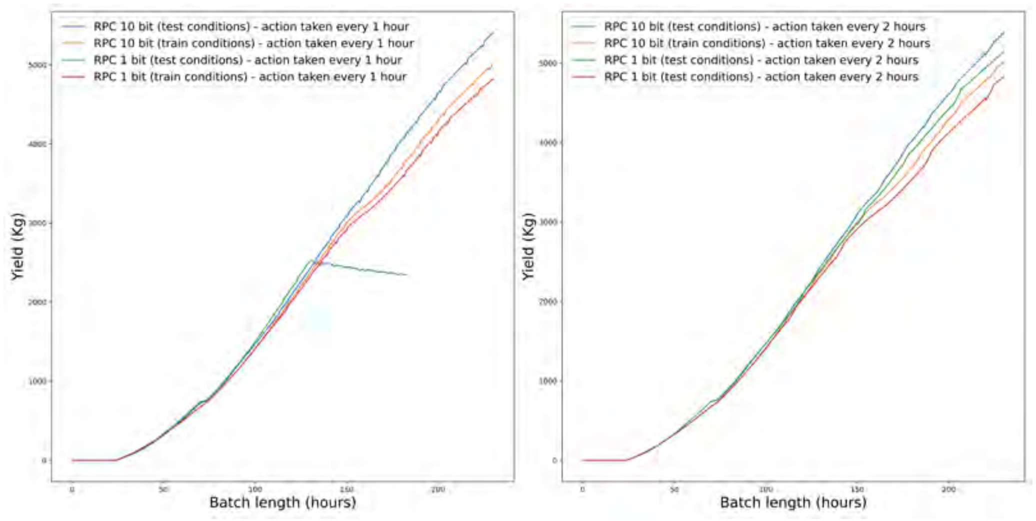

Figure 24: Penicillin yield for 1-bit and 10-bit RPC for different action intervals

section 5. Due to time and resource constraints, only the 10-bit RPC was trained for an additional 50000 steps which could be the reason why it performed well for both 1 hour and 2 hour action intervals.

As discussed in section 2.2.5.1, PAA is a critical parameter to the process. So the effect of the different initial PAA concentrations is used to evaluate the performance of the 10-bit RPC for 1 hour and 2 hour action intervals. From Figure 25 it is quite evident, the impact of PAA on the yield and performance of the agent. Though for all PAA concentrations the yields are above target (2000 Kg), in the case of 1 hour action intervals some batches were quite far from the mean. With 2 hour action intervals the performance is much superior with much less variability in the yields. This indicates the importance of knowing the best interval for when an action must be taken.

In a real process, disturbances to the process in the form of sensor noises and faults can affect the overall process. To see how RL agent behaves under such conditions, it is tested against the following faults (which are taken from a real process [9]):

- Aeration flowrate fault
- Base flowrate fault
- Coolant flowrate fault
- Vessel back pressure fault
- Substrate flow rate fault
- All of the above faults combined
- Temperature sensor error
- pH sensor error

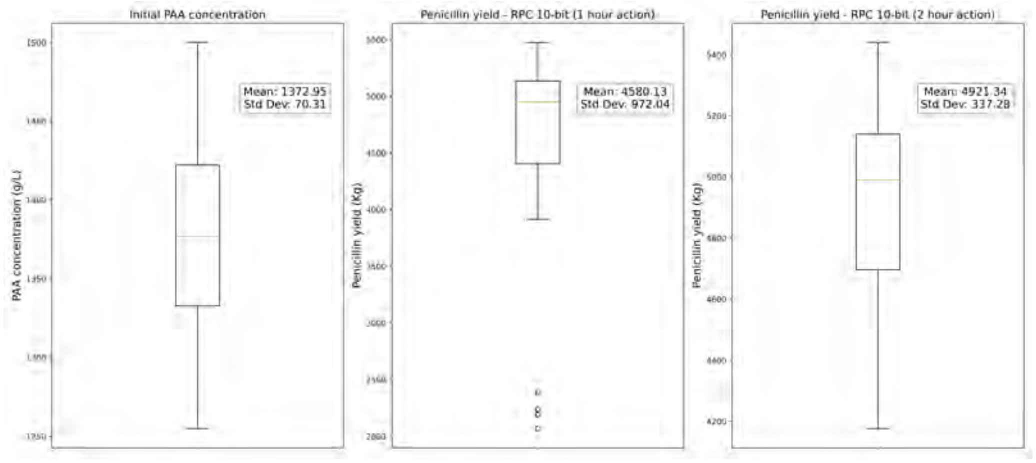

Figure 25: Effect of different initial PAA concentration on the penicillin yield for 10-bit RPC

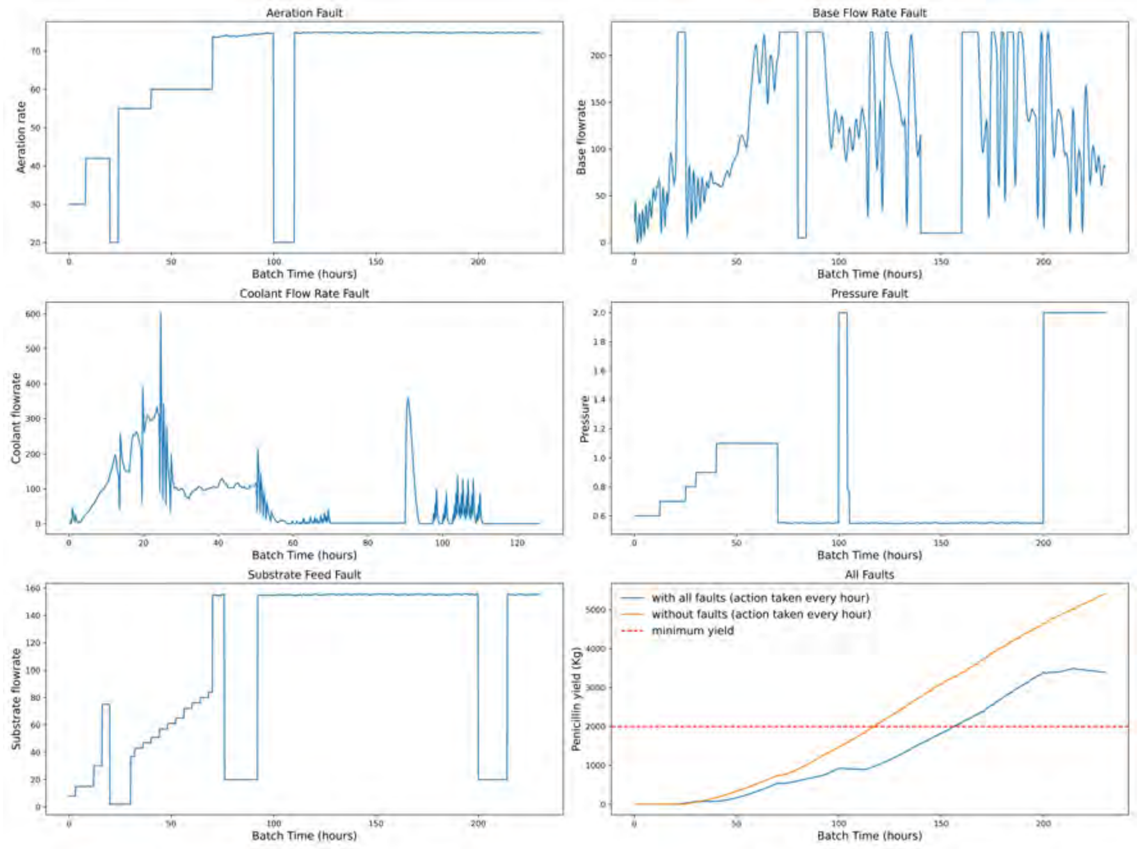

Figure 26: Individual process faults and its impact on the performance of the 10-bit RPC agent taking actions every hour.

Figure 26 shows individual faults in the process and the performance of the agent when all faults are activated simultaneously. Despite the drop on the yield, it still meets the minimum yield requirement. To see if taking an action every 2 hours instead of 1 hour makes a difference in dealing with these faults, Figure 27 shows the comparison in terms of yields.

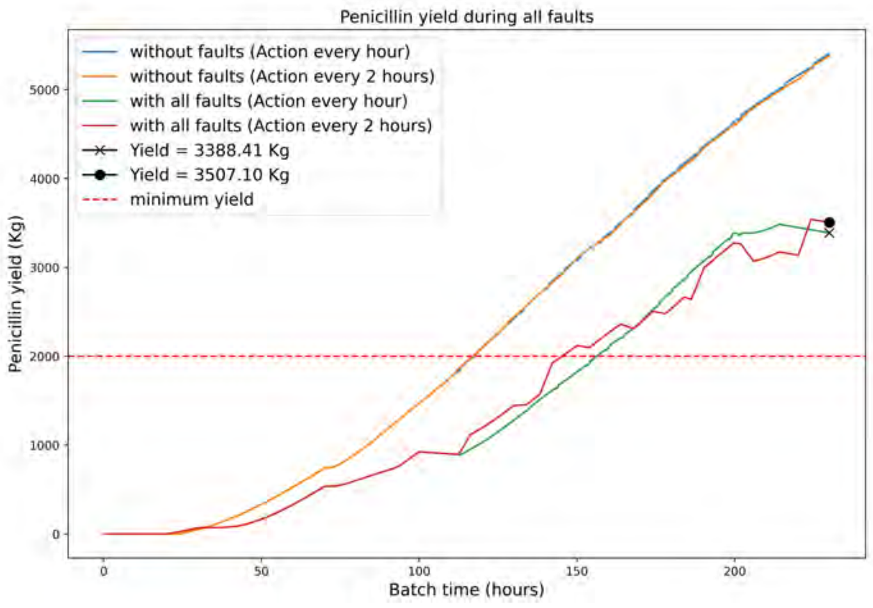

Figure 27: Effect of different action intervals by 10-bit RPC agent on the process with and without faults.

Even with all the faults activated, the 10-bit RPC agent can complete the entire batch without any violation of the constraints and still achieve yields much higher than the minimum requirement. Its ability to reject sensor noises such as pH and temperature sensor noises is depicted in Figure 28. The sensor noises does not affect the performance of the agent.

The 10-bit RPC agent was trained with the nominal vessel dimensions and evaluated against increasing and decreasing dimensions. Results from Table 6 shows the agent is able to achieve good yields despite not being trained on the modified vessel dimensions therefore showing its adaptability to modified process dynamics.

As discussed in section 2.2.3, the radius of the vessel and the impeller radius can affect the rate of dissolved oxygen which therefore can alter the dynamics of the process. The 10-bit RPC is evaluated against a decrease and increase in vessel radius by 10% as well as a decrease and increase in impeller radius by 10%.

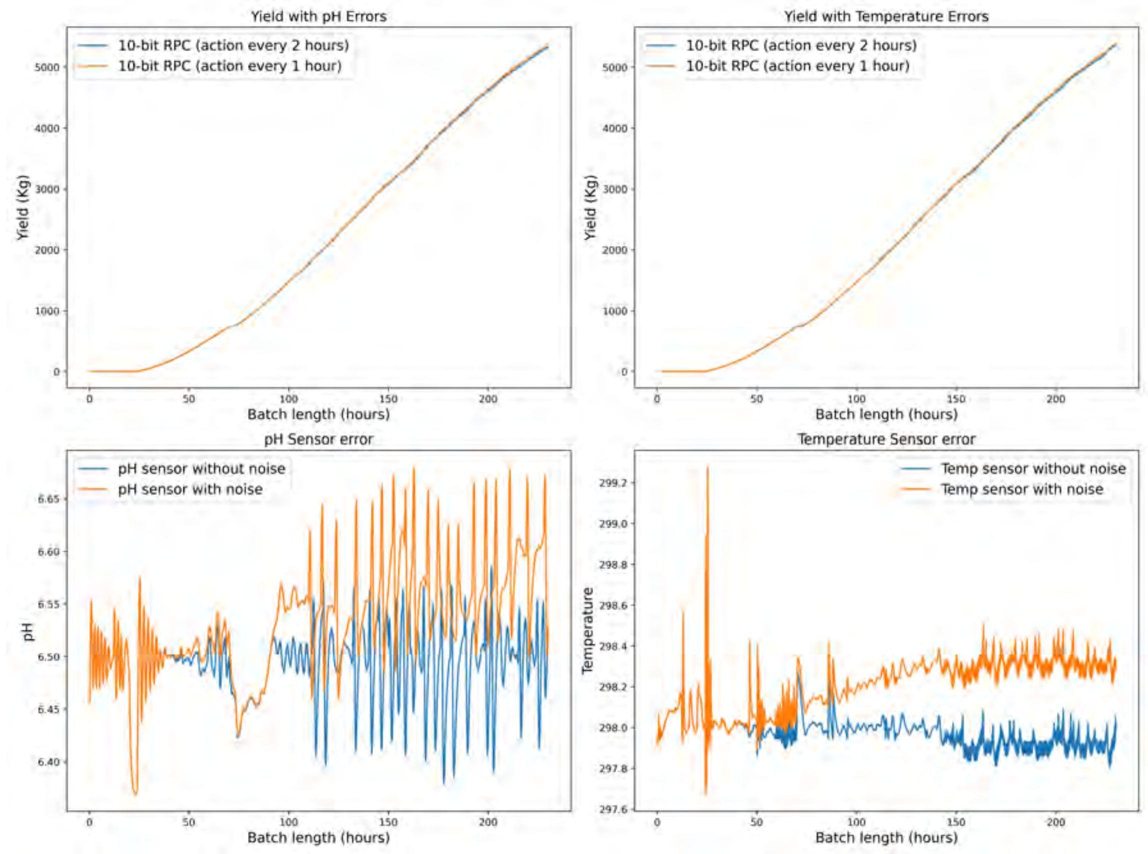

Figure 28: Effect of different action intervals by 10-bit RPC agent on the process with sensor errors.

	Penicillin yield (Kg)	**Penicillin yield (Kg)**
Nominal vessel and impeller radius	5405.59 (1 hour action)	5379.77 (2 hour action)
Vessel radius (10% increase)	5287.74 (1 hour action)	5275.61 (2 hour action)
Vessel radius (10% decrease)	5474.15 (1 hour action)	5439.87 (2 hour action)
Impeller radius (10% increase)	5467.96 (1 hour action)	5439.15 (2 hour action)
Impeller radius (10% increase)	5003.22 (1 hour action)	5138.25 (2 hour action)

Table 6: Effect of vessel dimensions on the performance of 10-bit RPC agent

4.4 Training with other RL algorithms

Training the environment with PPO exhibited greater instability compared to TD3. This is an anticipated outcome with an on-policy algorithm, as it relies on the current policy for exploration. The absence of a replay buffer means it does not recycle experiences, which is crucial in environments with slow dynamics, like this chemical process. Consequently, a significant number of experiences need to be reused before the algorithm can converge on an optimal policy.

PPO combined with Neuro-symbolic frameworks such as Logic Tensor Networks (LTN) [67] was also used for training. However, the algorithm breaks down quite early during the training process due to exploding gradients in the neural network.

5 Discussion

In this section we discuss the results from section 4 and understand the behaviour of the RL agents training performance and its evaluation against different test cases.

Among the RL agents trained in this work for 50000 steps, agents that use RPC showed better performance in terms of maximizing the penicillin yield compared to TD3 and SAC. Three different versions: 1-bit, 10-bit and 15-bit RPC agents were trained with 2 types of reward functions. The 15-bit showed better performance and was trained further for another 50000 steps (10000 steps in total) as the favourite RL agent.

The choice of reward function is critical when training the RL agent. In this work, 2 reward functions were used - penicillin concentration and penicillin yield, both of which are related to each other as seen in Equation 16. The performance of the RL agent was better when penicillin concentration was used as the reward function. To understand why this is the case, the plot of the penicillin yields and penicillin concentration must be analyzed.

From Figure 29, the profile of the penicillin yield has small dips and looks slightly noisy compared to the penicillin concentration. These small dips are caused by the vessel discharges to avoid vessel overflow. The vessel discharge does not affect the penicillin concentration and hence is insensitive to these vessel discharges. In conclusion, smoother the reward function better is the training performance.

When evaluated under various initial conditions and process faults, the 10-bit RPC agent, trained to take actions every 1 hour, exhibited inferior performance compared to when the same agent was configured to take actions every 2 hours. This discrepancy in performance can be elucidated by analyzing Figure 30. When actions are taken every hour, the batch fails to complete due to a violation of the viscosity constraint. Conversely, with actions taken every 2 hours under an identical agent and process conditions, significantly higher penicillin yields are achieved without violating any constraints.

The underlying reason for this disparity can be discerned by examining the discharge rates depicted in Figure 30. With longer action intervals, the vessel discharge is sustained for a prolonged duration. This allows for a larger volume available for biomass growth and extends the residence time in the vessel. Given the slow dynamics inherent in biochemical processes, it takes time to observe the effects of actions taken by the agent. With a longer residence time, these effects become more pronounced, enabling the agent to make informed decisions based on observed outcomes. Consequently, the viscosity could not be effectively controlled with actions taken every hour, as subsequent actions were initiated before the full effects of previous actions had manifested.

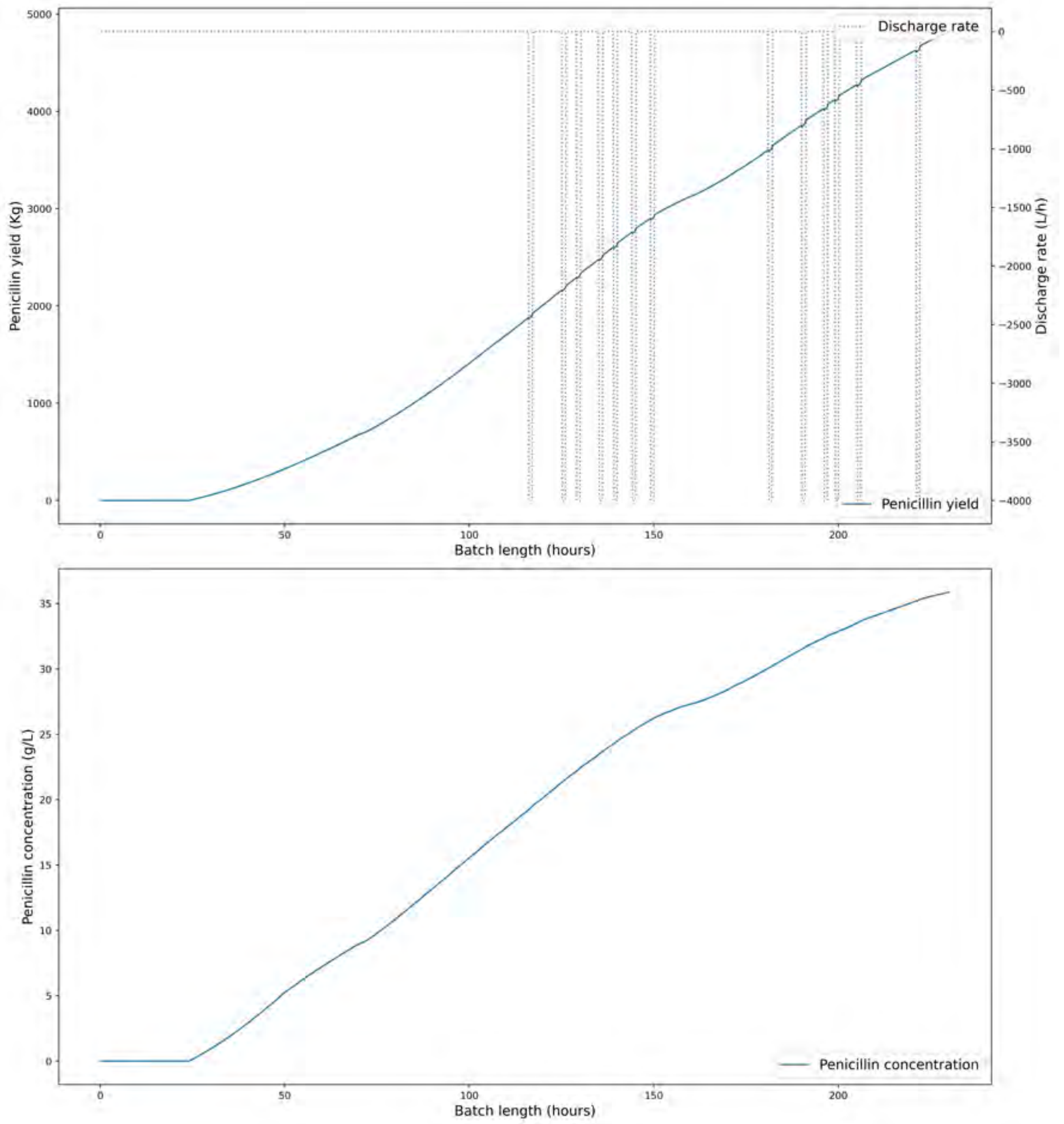

Figure 29: Comparison of penicillin yield and penicillin concentration profile for 10-bit RPC agent.

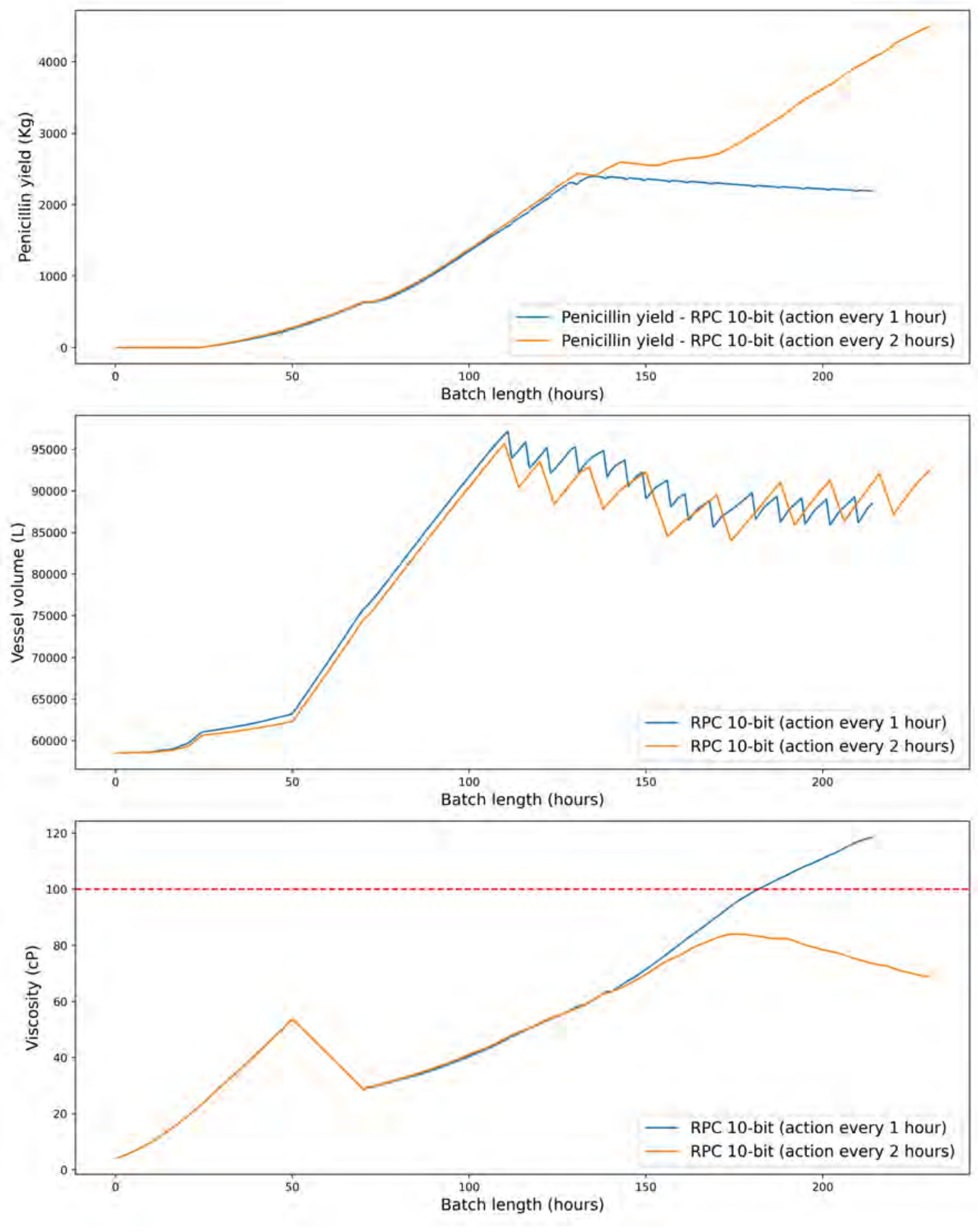

Figure 30: Impact of 10-bit RPC agent's actions every 1 hour and 2 hour on the process.

	Avg. batch yield (Kg)	**Num. of batches**	**Num. of batches below target**	**Training steps**	**Control action interval**
Operator controlled [9]	2882 ± 745	26	2	-	Not fixed (in hours)
Sequential Batch Control [9]	2912 ± 786	26	2	-	Not fixed (in hours)
Raman Spectroscopy with PI control on PAA [9]	3517 ± 315	26	0	-	PAA controlled every 12 minutes
Nonlinear MPC [38]	3987 ± 265	30	Not specified	-	Every 12 minutes
Soft Actor Critic (RL) [38]	3944 ± 272	30	Not specified	1 million	Every 12 minutes
10-bit RPC (This work)	4921 ± 337	26	0	100,000	Every 2 hours

Table 7: Comparison of different control strategies on the Indpensim

As a result, the performance of the 10-bit RPC agent acting every 2 hours under varying input conditions achieved high penicillin yields with a small variation compared to the same agent acting every 1 hour (see Figure 25). Furthermore, under different process faults (all acting in the same batch) and varying vessel dimensions, none of the batch yields were below target. This shows the generalization capabilities of the 10-bit RPC agent to take actions at different intervals, reject disturbances, achieve high penicillin yields and operate safely (no violation of constraints) without the need for retraining.

Finally, Table 7 compares the best RL agent from this work with other implementations. When operated manually or with an optimal predefined recipe using Sequential Batch Control (SBC), there were batches below target. With the help of Raman Spectroscopy for online measurements and PI controller to control the PAA concentration which is a critical parameter, the performance was improved as no batches were below target and the overall penicillin yield increased by 20% in comparison to SBC. Nonlinear MPC implemented by [38] was able to increase the yield by 37% (with respect to SBC). However, it assumed a full state feedback which means all 35 observations were recorded and used to get the next optimal control action. Soft Actor Critic was use as a RL controller similar to this work, but was trained with different environment configurations. For example 9 observations was used for the SAC training in this work whereas 16 observations were used in [38]. Moreover, training was done for 1 million steps. The SAC implementation was able to achieve yields 36% higher than SBC. Finally, the 10-bit RPC implementation in this study was able to achieve the highest increase of 57% in penicillin yield compared to SBC which only required the least number of training steps and used the least number of observations (9 observations).

6 Conclusion

In this study, various RL agents were deployed as high-level controllers for a penicillin process simulation, aiming to optimize penicillin yield, ensure safe operation, and generalize effectively to unseen scenarios. Among all the trained agents, the 10-bit RPC demonstrated superior performance.

It's adaptability and robust performance across various unseen input conditions underscore its potential for real-world applicability. It demonstrated remarkable flexibility by effectively executing actions even at intervals it hadn't been explicitly trained for, showcasing its ability to generalize learned behaviors to new situations. Despite encountering disturbances, the agent consistently met its control objectives and attained the desired target yields, albeit with minimal impact on performance. Notably, its ability to maintain control in the presence of errors in temperature and pressure sensors highlights its resilience to noisy or imperfect sensor data, a critical attribute in real-world environments where sensor accuracy may be compromised.

Moreover, the agent's commendable response to shifted process dynamics, such as alterations in vessel dimensions, further validates its versatility and adaptability. These observations indicate that the agent can effectively adapt to changes in the underlying system, a crucial capability for ensuring robust performance in dynamic industrial settings.

Remarkably, these achievements were attained with fewer training steps and observations compared to other implementations cited, making the approach more practical for real-world deployment. This efficiency is particularly promising considering the challenges often associated with collecting large amounts of training data in industrial settings. In many real processes, it may be impractical or cost-prohibitive to measure all process variables accurately. Therefore, the agent's ability to achieve commendable performance with limited data inputs holds significant potential for streamlining the deployment of AI-driven control systems in industrial applications.

7 Future research

Future research for the trained Reinforcement Learning Process Control (RPC) agent encompasses several key directions aimed at enhancing its real-world applicability and reliability. Integration into actual industrial processes is paramount to validate its effectiveness and behavior under real-time conditions. Concurrently, robustness testing is crucial to evaluate the agent's adaptability and resilience against extreme scenarios, ensuring its reliability in practical applications.

Generalization efforts should focus on extending the agent's capabilities to control diverse process types, enhancing its versatility and widening its potential applications. Additionally, improving data efficiency and exploring transfer learning techniques can expedite deployment in new environments with limited data availability. Integration with human operators presents an opportunity to develop collaborative control systems, leveraging the strengths of both humans and AI to enhance system performance and safety.

8 Acknowledgement

I would like to express my sincere gratitude to Dr. Fabian Buelow and Prof. Dr. Margret Bauer for providing me with the invaluable opportunity to conduct my thesis at ABB. Their support and encouragement throughout this journey have been instrumental in shaping my research experience and professional growth.

I am also grateful to the entire team at ABB for their collaboration, encouragement, and valuable input throughout this endeavor. Their contributions have enriched my learning experience and enhanced the quality of my research.

Lastly, I would like to extend my heartfelt appreciation to my family and friends for their unwavering support, encouragement, and understanding throughout this journey. Their love, encouragement, and belief in my abilities have been a constant source of motivation and strength.

References

[1] Afrânio Melo, Maurício M Câmara, Nayher Clavijo, and José Carlos Pinto. Open benchmarks for assessment of process monitoring and fault diagnosis techniques: A review and critical analysis. *Computers & Chemical Engineering*, 165:107964, 2022.

[2] Stephen Goldrick, Andrei Ştefan, David Lovett, Gary Montague, and Barry Lennox. The development of an industrial-scale fed-batch fermentation simulation. *Journal of biotechnology*, 193:70–82, 2015.

[3] R.S. Sutton and A.G. Barto. Reinforcement Learning: An Introduction. 9(5):1054–1054.

[4] Spinning up in deep rl. `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`, 2018.

[5] Mark L. Darby and Michael Nikolaou. MPC: Current practice and challenges. *Control Engineering Practice*, 20(4):328–342, April 2012.

[6] Steven Spielberg P, Aditya Tulsyan, Nathan Lawrence, Philip Loewen, and Bhushan Gopaluni. Towards self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal*, 65, 06 2019.

[7] Kalpesh Patel. Safe, fast and explainable online reinforcement learning for continuous process control. pages 54–60, 08 2022.

[8] Stephen Goldrick, Carlos Duran-Villalobos, Karolis Jankauskas, David Lovett, Suzanne Farid, and Barry Lennox. Modern day monitoring and control challenges outlined on an industrial-scale benchmark fermentation process. *Computers & Chemical Engineering*, 130, 05 2019.

[9] Stephen Goldrick, Carlos A Duran-Villalobos, Karolis Jankauskas, David Lovett, Suzanne S Farid, and Barry Lennox. Modern day monitoring and control challenges outlined on an industrial-scale benchmark fermentation process. *Computers & Chemical Engineering*, 130:106471, 2019.

[10] Ruan de Rezende Faria, Bruno Didier Olivier Capron, Argimiro Resende Secchi, and Maurício B. de Souza. Where Reinforcement Learning Meets Process Control: Review and Guidelines. 10(11):2311.

[11] Dominique Bonvin. Optimal operation of batch reactors—a personal view. *Journal of process control*, 8(5-6):355–368, 1998.

[12] Dominique Bonvin, Bala Srinivasan, and David Ruppen. Dynamic optimization in the batch chemical industry. *Chemical Process Control-VI*, 2001.

[13] A Arpornwichanop, P Kittisupakorn, and IM Mujtaba. On-line dynamic optimization and control strategy for improving the performance of batch reactors. *Chemical Engineering and Processing: Process Intensification*, 44(1):101–114, 2005.

[14] Omar Santander, Vidyashankar Kuppuraj, Christopher A Harrison, and Michael Baldea. An open source fluid catalytic cracker-fractionator model to support the development and benchmarking of process control, machine learning and operation strategies. *Computers & Chemical Engineering*, 164:107900, 2022.

[15] James J Downs and Ernest F Vogel. A plant-wide industrial process control problem. *Computers & chemical engineering*, 17(3):245–255, 1993.

[16] Andreas Bathelt, N Lawrence Ricker, and Mohieddine Jelali. Revision of the tennessee eastman process model. *IFAC-PapersOnLine*, 48(8):309–314, 2015.

[17] Carla Martin-Villalba, Alfonso Urquia, and Guodong Shao. Implementations of the tennessee eastman process in modelica. *IFAC-PapersOnLine*, 51(2):619–624, 2018.

[18] RK Bajpai and M Reuss. A mechanistic model for penicillin production. *Journal of Chemical Technology and Biotechnology*, 30(1):332–344, 1980.

[19] Gülnur Birol, Cenk Ündey, and Ali Cinar. A modular simulation package for fed-batch fermentation: penicillin production. *Computers & chemical engineering*, 26(11):1553–1565, 2002.

[20] Jan Van Impe and Geert Gins. An extensive reference dataset for fault detection and identification in batch processes. *Chemometrics and Intelligent Laboratory Systems*, 148:20–31, 2015.

[21] Geert Gins, Jef Vanlaer, Pieter Van den Kerkhof, and Jan FM Van Impe. The raymond simulation package—generating raypresentative monitoring data to design advanced process monitoring and control algorithms. *Computers & chemical engineering*, 69:108–118, 2014.

[22] Michał Bartyś, Ron Patton, Michał Syfert, Salvador de las Heras, and Joseba Quevedo. Introduction to the damadics actuator fdi benchmark study. *Control engineering practice*, 14(6):577–596, 2006.

[23] GC Paul and CR Thomas. A structured model for hyphal differentiation and penicillin production using penicillium chrysogenum. *Biotechnology and bioengineering*, 51(5):558–572, 1996.

[24] Fazilet Vardar and MD Lilly. Effect of cycling dissolved oxygen concentrations on product formation in penicillin fermentations. *European journal of applied microbiology and biotechnology*, 14:203–211, 1982.

[25] GN Rolinson. Respiration of penicillium chrysogenum in penicillin fermentations. *Microbiology*, 6(3-4):336–343, 1952.

[26] Ryon Frick and Beth Junker. Indirect methods for characterization of carbon dioxide levels in fermentation broth. *Journal of bioscience and bioengineering*, 87(3):344–351, 1999.

[27] Patrick N Royce. Effect of changes in the ph and carbon dioxide evolution rate on the measured respiratory quotient of fermentations. *Biotechnology and bioengineering*, 40(10):1129–1138, 1992.

[28] Celeste M Todaro and Henry C Vogel. *Fermentation and biochemical engineering handbook*. William Andrew, 2014.

[29] Mhairi McIntyre, David R Berry, and Brian McNeil. Response of penicillium chrysogenum to oxygen starvation in glucose-and nitrogen-limited chemostat cultures. *Enzyme and Microbial Technology*, 25(3-5):447–454, 1999.

[30] Dirk J Hillenga, H Versantvoort, S Van der Molen, A Driessen, and Wil N Konings. Penicillium chrysogenum takes up the penicillin g precursor phenylacetic acid by passive diffusion. *Applied and environmental microbiology*, 61(7):2589–2595, 1995.

[31] B Metz and NWF Kossen. The growth of molds in the form of pellets–a literature review. *Biotechnology and bioengineering*, 19(6):781–799, 1977.

[32] Jianqiang Lin, Sang-Mok Lee, Ho-Joon Lee, and Yoon-Mo Koo. Modeling of typical microbial cell growth in batch culture. *Biotechnology and Bioprocess Engineering*, 5:382–385, 2000.

[33] L Shuler Michael and Fikret Kargi. Bioprocess engineering: basic concepts, 2002.

[34] RD Megee III, S Kinoshita, AG Fredrickson, and HM Tsuchiya. Differentiation and product formation in molds. *Biotechnology and Bioengineering*, 12(5):771–801, 1970.

[35] Volker Tiller, Juliane Meyerhoff, Ditmar Sziele, Karl Schügerl, and Karl-Heinz Bellgardt. Segregated mathematical model for the fed-batch cultivation of a high-producing strain of penicillium chrysogenum. *Journal of biotechnology*, 34(2):119–131, 1994.

[36] José C Menezes, Sebastião S Alves, João M Lemos, and Sebastião Feyo de Azevedo. Mathematical modelling of industrial pilot-plant penicillin-g fed-batch fermentations. *Journal of Chemical Technology & Biotechnology: International Research in Process, Environmental AND Clean Technology*, 61(2):123–138, 1994.

[37] Stephen Goldrick, Ewan Mercer, Gary Montague, David Lovett, and Barry Lennox. Control of an industrial scale bioreactor using a pat analyser. *IFAC Proceedings Volumes*, 47(3):6222–6227, 2014.

[38] Haoran Li, Tong Qiu, and Fengqi You. Ai-based optimal control of fed-batch biopharmaceutical process leveraging deep reinforcement learning. *Chemical Engineering Science*, 292:119990, 2024.

[39] Maciej Ławryńczuk, Piotr M Marusak, and Piotr Tatjewski. Cooperation of model predictive control with steady-state economic optimisation. *Control and Cybernetics*, 37(1):133–158, 2008.

[40] Haeun Yoo, Boeun Kim, Jong Woo Kim, and Jay H. Lee. Reinforcement learning based optimal control of batch processes using monte-carlo deep deterministic policy gradient with phase segmentation. *Computers & Chemical Engineering*, 144:107133, 2021.

[41] Sigurd Skogestad. Control structure design for complete chemical plants. *Computers & Chemical Engineering*, 28(1-2):219–234, 2004.

[42] Ton Backx, Okko Bosgra, and Wolfgang Marquardt. Integration of model predictive control and optimization of processes: Enabling technology for market driven process operation. *IFAC Proceedings Volumes*, 33(10):249–260, 2000.

[43] Veronica Adetola and Martin Guay. Integration of real-time optimization and model predictive control. *Journal of Process Control*, 20(2):125–133, 2010.

[44] Rui Nian, Jinfeng Liu, and Biao Huang. A Review on Reinforcement Learning: Introduction and Applications in Industrial Process Control. *Computers & Chemical Engineering*, 139:106886, April 2020.

[45] Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. Mutual Alignment Transfer Learning. pages 281–290. PMLR.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[47] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

[48] Richard Bellman. Dynamic programming and stochastic control processes. *Information and control*, 1(3):228–239, 1958.

[49] Martijn RK Mes and Arturo Pérez Rivera. *Approximate dynamic programming by practical examples*. Springer, 2017.

[50] Markov Decision Processes: Discrete Stochastic Dynamic Programming | Wiley.

[51] Christopher JCH Watkins and Peter Dayan. \cal q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[52] Oguzhan Dogru, Nathan Wieczorek, Kirubakaran Velswamy, Fadi Ibrahim, and Biao Huang. Online reinforcement learning for a continuous space system with experimental validation. *Journal of Process Control*, 104:86–100, 2021.

[53] Andrew Y Ng. *Shaping and policy search in reinforcement learning*. University of California, Berkeley, 2003.

[54] S Kumar Chenna, Yogesh Kr Jain, Himanshu Kapoor, Raju S Bapi, Narri Yadaiah, Atul Negi, V Seshagiri Rao, and Bulusu Lakshmana Deekshatulu. State estimation and tracking problems: A comparison between kalman filter and recurrent neural networks. In *Neural Information Processing: 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25, 2004. Proceedings 11*, pages 275–281. Springer, 2004.

[55] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[56] Steven J Bradtke, B Erik Ydstie, and Andrew G Barto. Adaptive linear quadratic control using policy iteration. In *Proceedings of 1994 American Control Conference-ACC'94*, volume 3, pages 3475–3479. IEEE, 1994.

[57] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[58] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[59] Ian H Witten. An adaptive optimal controller for discrete-time markov environments. *Information and control*, 34(4):286–295, 1977.

[60] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. In *Neurocomputing: foundations of research*, pages 535–549. 1988.

[61] Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. *Advances in neural information processing systems*, 11, 1998.

[62] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Robust predictable control. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27813–27825. Curran Associates, Inc., 2021.

[63] Wenjiao Zai, Junjie Wang, and Guohui Li. A drone scheduling method for emergency power material transportation based on deep reinforcement learning optimized pso algorithm. *Sustainability*, 15(17):13127, 2023.

[64] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Robust predictable control. *Advances in Neural Information Processing Systems*, 34:27813–27825, 2021.

[65] Gymnasium documentation. `https://gymnasium.farama.org/`.

[66] Mohan Zhang, Xiaozhou Wang, Benjamin Decardi-Nelson, Bo Song, An Zhang, Jinfeng Liu, Sile Tao, Jiayi Cheng, Xiaohong Liu, DengDeng Yu, et al. Smpl: Simulated industrial manufacturing and process control learning environments. *Advances in Neural Information Processing Systems*, 35:26631–26644, 2022.

[67] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.