Improving Water Treatment Using Reinforcement Learning

by

Puer Liu

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

Master of Science

in

Statistical Machine Learning

Department of Computing Science

University of Alberta

Abstract

We have witnessed the rising popularity of real-world applications of reinforcement learning (RL). However, most successful real-world applications of RL rely on high-fidelity simulators that enable rapid iteration of prototypes, hyperparameter selection and policy training. On the other hand, RL is not wildly used in industrial control problems where simulators are too expensive or complicated to build. In addition to the lack of simulators, industrial control problems often face other challenges including high dimensionality of the observation space, missing or noisy sensor values, and control happening on multiple time scales. These challenges combined make industrial control a unique area of research that has not been fully explored yet.

Water treatment plants (WTPs) are one of the most important infrastructures in today's world. They not only deliver clean and accessible water to households but are also responsible for wastewater treatment before returning to the water cycle. Nowadays, accurate sensors and remove control functionalities make it possible to have fully automated WTPs with minimal human intervention. Such "smart" WTPs will be valuable for providing life-giving water to areas where full-time WTP operators are inaccessible. Since the water treatment process shares many characteristics with other industrial control tasks, it is an excellent platform for developing novel algorithms that handle challenges common in industrial control tasks.

The objective of this thesis is to formulate the water treatment process as a collection of RL tasks. To achieve this, we develop the software stack for

real-time monitoring, data collection, and control on a small-scale plant that shares similarities to the main plant. We conduct a detailed survey on the characteristic of installed sensors and their behaviour in different operating modes. We determine sub-tasks that are suitable for RL. We then identify the challenges we found while experimenting with the WTP. Based on these findings, we present a case study on a sub-task: chemical dosing rate control in water pretreatment, which utilizes the offline logs and does not use a simulator.

To my family and friends.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Adam White, for his continuing support. He led me to the world of reinforcement learning when I was an undergraduate student. Adam has always been an excellent source of insights and ideas throughout my Master's program. Not only has his advice on the design of meaningful empirical experiments proven to be priceless, but also I have benefited a lot from his help with scientific writing.

I am thankful to Dr. Martha White, for her generous comfort when the project got stuck. Her vision of the project constantly helped me focus on the most important scientific question when multiple research directions can be explored.

I owe many thanks to other collaborators in the Water team of RLAI: James Bell, Matthias Stone, Marlos C. Machado, Han Wang, Kamran Janjua, Kirby Banman, Jordan Coblin, Lingwei Zhu, Erfan Miahi, Csaba Szepesvári, Shaurya Seth and Zonglun Li. It is a pleasure to work with a vivid team that is knowledgeable, open-minded and willing to accept challenges.

Finally, I would like to thank the Town of Drayton Valley, ISL Adapt, ISL Engineering, NSERC, Alberta Innovations and Alberta Machine Learning Institute for funding this research or providing in-kind donations.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a field under the broader domain of machine learning. Unlike standard supervised learning that requires the true target value directly provided to the learning model, RL is a general framework that involves a learning agent and an environment, and the goal of the agent is to learn an intelligent *policy* from trial and error while interacting with the environment, without the need of an explicit teacher (Sutton & Barto, 2018). Since we do not have to hand-craft the true target, many tasks can be formulated with the RL framework naturally, including ones that are difficult for supervised learning. RL has gained a lot of attention recently due to many successful applications, such as achieving human-level performance in Atari games (Mnih et al., 2015) and Go (Silver et al., 2017). In addition to games, we have also witnessed some successful RL applications in real-world tasks, including controlling stratospheric balloons (Bellemare et al., 2020) and tokamak plasmas for nuclear fusion (Degrave et al., 2022). In this thesis, we focus on another real-world application: water treatment. Water treatment plants (WTPs) have many characteristics that make them suitable for RL. The most important one is that their objective is clear: providing clean water. However, we also recognize that there are some challenges in applying RL. Some challenges are common to many other industrial control tasks, while some others are specific to WTPs.

1.1 RL for WTPs

Water treatment plays a vital role in today's world. As the population increases, clean and accessible water becomes increasingly important; at the same time, rapid industrialization and urbanization have increased the demand for wastewater treatment. Having fully automated WTPs is not about replacing people: operators or engineers are still needed to monitor the process and intervene if necessary. Instead, it frees people from operating at low time scales such as in minutes or hours and allows people to focus on high-level tasks with larger time scales such as choosing the operation mode for the next week, or deciding whether perform a filter replacement. Moreover, according to World Health Organization, at least 2 billion people use contaminated drinking water sources (World Health Organization, 2022). Therefore, fully automated and self-adaptive WTPs help deliver life-giving clean water to regions where water treatment operators or engineers are not accessible full-time.

A WTP is a suitable platform for RL. It is safe: they are usually equipped with built-in safety layers to prevent dangerous actions, so premature exploratory actions from an RL agent will not put the WTP into dangerous states. WTPs have clear goals: produce clean water while minimizing electricity cost, and chemical usage as well as extending the life of filters. It is sensor-rich. Many WTPs already have sensors installed to provide human operators with plant information, and it is also possible to install additional sensors to make the internal state of the plant more observable. The plant can operate in short intervals, so we can construct a continuous and dense reward function for RL agents. Furthermore, decisions happen on multiple time scales. For example, the decision on the chemical dose rate happens every three seconds, but the decision of changing the current operating mode happens hourly or daily. Therefore, a WTP is a good platform for testing RL algorithms that allow operations on multiple time scales.

Despite these advantages, there are some challenges in applying RL to water treatment tasks. Some challenges are general to many industrial control

problems, while some are specific to WTPs. Like many other industrial applications, simulators are not available for WTPs, so an RL agent cannot leverage a simulator to learn a good policy, or to do planning. Also, the input data from the WTP is high-dimensional and multimodal. Some sensor readings are noisy and sometimes inaccurate, adding more complexity to the task. Many WTPs have multiple stages in their water treatment process, and depending on the scale of the WTP and water flow rate, a change in the early stage may take a while to be observed in later stages. There are different modes of operations; special care needs to be taken to deal with mode changes. Depending on the operation time, we may have access to logs of sensory values, setpoints and modes. Effective utilization of the offline log may be an important factor for the success of an RL agent. Possible methods include initializing a policy from offline logs using offline RL algorithms such as Fitted Q Iteration (Ernst et al., 2006), and Conservative Q-learning (Kumar et al., 2020), or construct a low-granularity simulation of the environment from the offline logs (Wang et al., 2022).

1.2 Objective

The objective of this thesis is to formulate the WTP as a collection of RL tasks. To achieve this, We investigate the characteristics of a pilot WTP in detail, including understanding its operating cycles, analyzing observable sensor values and internal states such as setpoints and valve status, and subtasks that can be independently controlled. We also define open research questions found during the exploration of the WTP. We also provide a case study on the dosing rate task, one of the sub-tasks that can be independently controlled by RL.

1.3 Contributions

The main contribution of this thesis includes:

1. We helped with the formalization of the water treatment process as a

collection of RL tasks, identifying several novel research questions and opportunities for algorithmic advances.

2. We contributed to the development of a software stack for real-time monitoring, data collection, prediction and control of a real pilot WTP.

3. We provide a detailed case study on learning to control chemical dosing on the pilot WTP.

1.4 Outline

This thesis has 6 chapters. In Chapter 2 we introduce background of RL and other relevant concepts. In Chapter 3 we describe the water treatment process and the pilot water treatment plant that we have access to in detail. Chapter 4 lists some challenges in applying RL in WTP and industrial control in general. Chapter 5 formulates the WTP control as a collection of RL tasks. In Chapter 6 we provide a case study on the dosing rate task, show empirical results, and discuss directions for future work. Finally, we conclude this thesis in Chapter 7.

1.5 Summary

In this chapter, we introduced the topics of this thesis: investigation and formulation of the WTP control as a collection of RL tasks. We provided the reasons why RL is suitable for WTP and some challenges. We listed the contributions and closed with the outline of this thesis.

Chapter 2

Background

This chapter covers the important concepts used in this thesis. We describe the RL problem settings and general value functions (GVF).

2.1 Reinforcement learning

In the framework of RL, the interaction between the agent and the environment is modelled as a Markov Decision Process (MDP).A MDP can be defined as $(\mathcal{S}, \mathcal{A}, R, P, \gamma, \mu)$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defines the next state distribution as a function of current state and action, $\gamma \in [0, 1]$ is the discount factor, and μ is the distribution of starting states (Sutton and Barto, 2018). At step t, the RL agent observes a state $s_t \in \mathcal{S}$, and selects an action $a_t \in \mathcal{A}$ which is sampled from a policy $\pi(\cdot|s_t)$. The environment then takes the action a_t from the agent and return a new state $s_{t+1} \in \mathcal{S}$ according to $P(s_t, a_t)$ and a reward $r_{t+1} \in \mathbb{R}$ that is computed from R. This interaction continues until a terminal state is reached in episodic tasks. In continuing tasks, this interaction runs forever.

In control tasks, the goal of an RL agent is to learn a policy that maximizes the expected cumulative discounted reward, also called the expected *return*. At step t, the return is defined as follows:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k,$$

Where T is the termination time step. For continuing tasks, $T = \infty$.

The value function of a given state s is defined as the expected return following a given policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right],$$

Similarly, the value function of state-action pairs, also called the action-value function, is defined as:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = s \right] \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}.$$

Value functions provide an easy way to evaluate policies. We say a policy π is better than or equal to another policy policy $\pi^{'}$ if $v_\pi(s) \leq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. The optimal value function is defined as

$$v^*(s) \doteq \max_\pi v_\pi(s),$$

for all $s \in \mathcal{S}$. And consequently, the optimal policies are the policies that achieve the optimal value function:

$$\pi^* \doteq arg\max_\pi v_\pi(s).$$

Note that there could be multiple policies sharing the optimal value function. Similarly, the optimal action-value function is defined as

$$q^*(s, a) \doteq \max_\pi q_\pi(s, a) \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}.$$

2.2 Q-learning

Q-learning (Watkins, 1989) is an algorithm for approximating q^*. The update rule is defined by

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right) \right],$$

where $\alpha \in (0, 1]$ is the step size, R_{t+1} is the reward received at the current time step after taking action A_t at state S_t. It uses $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ as the estimation of the return at time t.

2.3 Partial Observability

In many tasks, storing the value or action-values in a lookup table indexed by states or (state, action) pairs is not feasible. After all, the "state" of most real-world tasks is not accessible and is only partially observable to agents. We call the values that are observable by the agent "*observations*". For example, in video games, an agent can only observe the pixels displayed on the screen as well as the previous actions it took. It does not have access to underlying game states stored in the computer memory. In a maze problem, a realistic agent only has access to its surroundings, instead of the whole map. Therefore, an agent needs to construct a *representation* of the underlying states from the history of interactions H_t: $S_t = f(H_t)$, where $H_t = A_0, O_1, A_1, O_2, \ldots, A_{t-1}, O_t$ is the history of alternating sequence of observations and actions. For computational reasons, we would like to incrementally compute the S_t using a state-update function u:

$$S_{t+1} \doteq u\left(S_t, A_t, O_{t+1}\right), \text{ for all } t \geq 0.$$

The state-update function can be manually created with domain knowledge or learned. Studies show that Recurrent Neural Networks can be used to learn a representation for RL agents (Li et al., 2015, Chen et al., 2016 and Hausknecht and Stone, 2017). As we will discuss in Section 2.5, General value functions (Sutton et al., 2011) can also be used to construct representations from observations.

2.4 Function Approximation and Deep RL

We can use function approximation to learn a representation of the underlying states from the observations and approximate the value functions. Specifically,

we denote the value function to be $v(s, \mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^d$ is the weight of the function approximator. Unlike tabular methods where the estimated value function can be equal to the true value function exactly, the best effort of function approximation methods is to minimize the *mean square value error* ($\overline{\text{VE}}$) between the true value function $v_\pi(s)$ and the approximated value function $\hat{v}(s, \mathbf{w})$:

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w})\right]^2,$$

where $\mu(s)$ is the state-visitation distribution. A stochastic gradient descent method to update this objective is as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[v_\pi\left(S_t\right) - \hat{v}\left(S_t, \mathbf{w}_t\right)\right] \nabla \hat{v}\left(S_t, \mathbf{w}_t\right),$$

where $\alpha \in (0, 1]$ is the step size and $\nabla \hat{v}\left(S_t, \mathbf{w}_t\right)$ denotes the gradient vector of $\hat{v}\left(S_t, \mathbf{w}_t\right)$.

In linear function approximation methods, the value function is approximated linearly:

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s),$$

where $x(s)$ is a function that generates a representation of the state s. Each item in $x(s)$ can be treated as a basis function: $x_i : \mathcal{S} \to \mathbb{R}$.

Deep RL, as the name suggests, uses artificial neural networks (ANNs) as the function approximator. Deep Q Network (DQN) combining ANN with Q-learning achieves human-level performance or above in many Atari games (Mnih et al., 2015). In addition to the use of ANN, DQN uses two important techniques: (1) it uses experience replay to store the most recent experiences in the form of (s_t, a_t, r_t, s_{t+1}) for time step t, and update the ANN by training with the data in the replay buffer; (2) DQN uses a target network with weights copied from the learning network every C steps, and hence it updates slower than the learning network. The objective of DQN at iteration i can therefore be expressed as:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right],$$

where $U(D)$ denotes that samples are sampled uniformly from the ER, θ_i and θ_i^- are the weight of the learning network and the weight of the target network at iteration i respectively.

2.5 General Value Functions and Nexting Prediction

General value functions (GVFs, Sutton et al., 2011) extend the standard value functions in two aspects: (1) unlike standard value functions that aim to predict the expected discounted cumulative reward ($R_t \in \mathbb{R}$), GVFs predict the expected discounted cumulative value of an arbitrary signal, which is called cumulant $C_t \in \mathbb{R}$; (2) the discounted factor γ is generalized to a discounted function $\gamma : \mathcal{S} \to [0, 1]$. Formally GVF can be written as

$$v_{\pi,\gamma,C}(s) \doteq \mathbb{E}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\gamma\left(S_i\right)\right)C_{k+1} \mid S_t = s, A_{t:\infty} \sim \pi\right].$$

GVF can be used to implement *Nexting* predictions (Modayil et al., 2012). In Psychology, "Nexting" is used to describe the behaviour of many animals that continually make short-term predictions about their input (Clark, 2013). In RL, Nexting predictions can be used as predictive auxiliary tasks along with the main RL task. The timescale of the prediction can be controlled by selecting the discounted function γ properly. A natural way to learn the Nexting predictions is to use temporal-difference methods (Sutton, 1988) like TD(λ). In the work by Modayil et al. (2012), TD(λ) is adapted to update the prediction with linear function approximation as follows:

$$\theta_{t+1}^i = \theta_t^i + \alpha\left(C_{t+1}^i + \gamma^i \phi_{t+1}^\top \theta_t^i - \phi_t^\top \theta_t^i\right)\mathbf{e}_t^i,$$

where $\theta_t \in \mathbb{R}^d$ is the weight vector, $\phi \in \mathbb{R}^d$ is the representation vector constructed from past observations or past actions (or both), $\alpha \in (0, 1]$ is the

step size parameter, C is the cumulant, and $\mathbf{e} \in \mathbb{R}^d$ is the eligibility trace vector, which is updated every step by the following update rule:

$$\mathbf{e}_t^i = \gamma^i \lambda \mathbf{e}_{t-1}^i + \phi_t,$$

where $\lambda \in [0, 1]$ is the trace decay. When $\lambda = 0$, this algorithm becomes TD(0). When $\lambda = 1$, the algorithm minimizes the mean squared value error $\hat{v}(s, \mathbf{w})$. An empirical experiment shows that it is possible to make a large number of Nexting predictions in parallel using TD(λ): a robot can learn 2160 predictions in four different timescales simultaneously (Modayil et al., 2012).

2.6 Summary

In this chapter, we introduced the formalization of reinforcement learning, and the well-known control algorithm: Q-learning. We also discussed the partial observability problem. Then we talked about function approximations and deep RL. Finally, we introduced General Value Functions and Nexting predictions.

Chapter 3

Water Treatment Process

This chapter provides an overview of the water treatment process. Then we introduce the software stack that enables data logging, control and monitoring of the pilot WTP. Finally, we take a deeper look into the pilot water treatment plant available for RL experiments.

3.1 Water Treatment Process

Depending on the characteristic of the water source and the purpose of the effluent water, technologies used in water treatment plants can be vastly different. Most WTPs use both physical methods and chemical methods. Among physical methods, filtration plays a dominant role (Cheremisinoff, 2002a). Chemical methods use chemical interactions to separate contaminants from water. Some wildly-used chemicals in water treatment include Aluminum-based and iron-based chemicals for suspended solids removal, and chlorine and iodine for disinfection.

A typical WTP has a multi-stage water treatment process: pretreatment, filtration and storage. We will describe each stage in detail.

Pretreatment

Also called primary treatment, this is the first stage of treatment after feeding the water from the water source. The main goal of this stage is to remove solids from the water. The most common method is chemical pretreatment, which involves adding chemicals such as coagulants into the flocculation tank to

remove suspended solids and pH adjusters to remove dissolved contaminants. The biggest cost factor of pretreatment is chemical usage.

Filtration

This is the stage that usually follows pretreatment immediately. Water is pushed through one or more filters to remove solids and some bacteria. A pump can be used to increase the water pressure. Usually, the filtration process requires backwash (BW): a self-cleaning mode in which a small portion of permeate water is pushed back through the filters in the reverse direction, cleaning out the dust on the filters built-up during its operation. The output of BW goes directly to the drain or goes back to the pretreatment stage. The biggest cost factor of filtration is the electricity usage and filter change.

Storage

Depending on the purpose of the water treatment plant, this stage is optional or stores the processed water in a tank or reservoir.

3.2 Pilot Water Treatment Plant

In this section, we describe the pilot water treatment plant (pilot) we have access to that is used for our RL experiments.

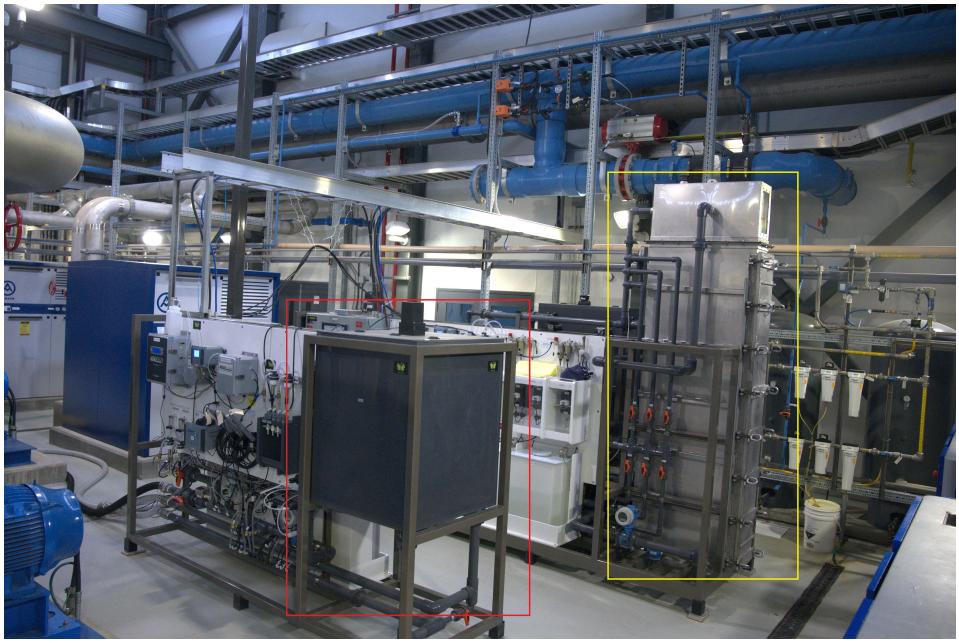

Figure 3.1: The "mini" pilot WTP. The red box shows the flocculation tank in the pretreatment stage, and the yellow box shows the filters in the filtration stage.

Located in Drayton Valley, Alberta, Canada, the pilot (Figure 3.1) is designed and built by Suez Water Technologies & Solutions, and maintained by ISL Engineering. The pilot is placed in the same building as the main production plant, thus sharing the same influent water. Figure 3.2, 3.3 and 3.4 show the simplified design diagrams of the pilot in three stages: pretreatment, process (filtration) and permeate (after filtration). The goal of the pilot is to replicate the full-scale facility, so the pilot and the main plant use the same water treatment technologies and the same water source. As an experiment platform, the pilot plant pipes the processed water to the drain directly, so the effluent water produced during experiments will not have any health risks. To help RL agents make better decisions, the pilot is equipped with more sensors than the main plant, including additional temperature, pH, pressure, turbidity, flow speed, and flocculation tank level sensors. In addition to sensors, agents have access to the internal states of the pilot via various setpoints, current plant mode and alarm timers.

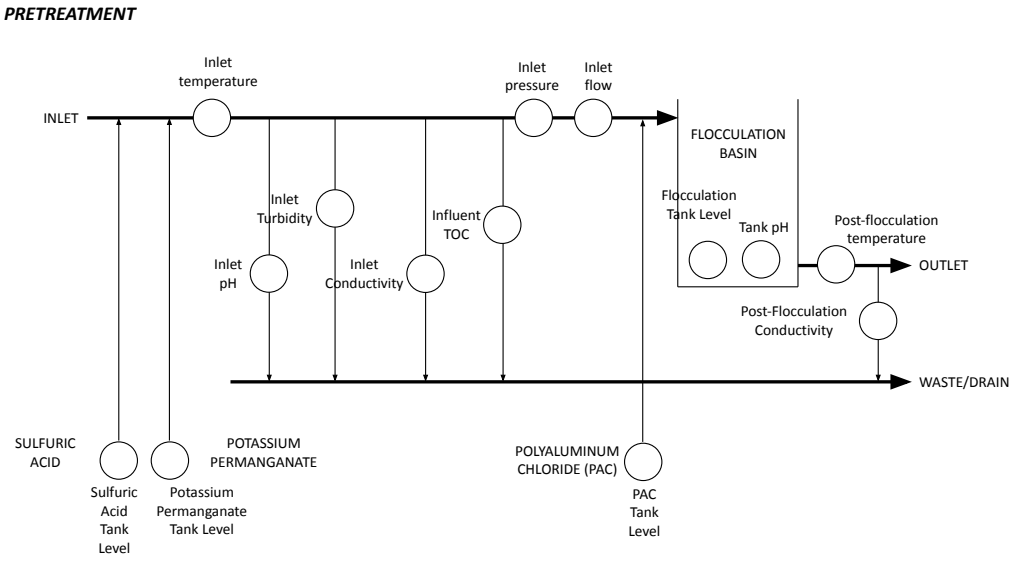

Figure 3.2: Simplified design diagram of the pretreatment stage of the pilot. Inlet water (top left) goes to the flocculation basin before being piped out of this stage (middle right). During the process, multiple characteristics of the water are measured, including temperature, pH, turbidity, conductivity, total organic carbon (TOC), pressure and flow rate. Lines represent the process lines and arrows show the flow direction. Bold lines show the major process lines, and thin lines show the minor process lines. Circles represent installed sensors. Some sensors are installed along the major process lines, while other sensors require a minor process line so that the used water flows to the drain. Figure 3.3 and 3.4 follows the same convention.

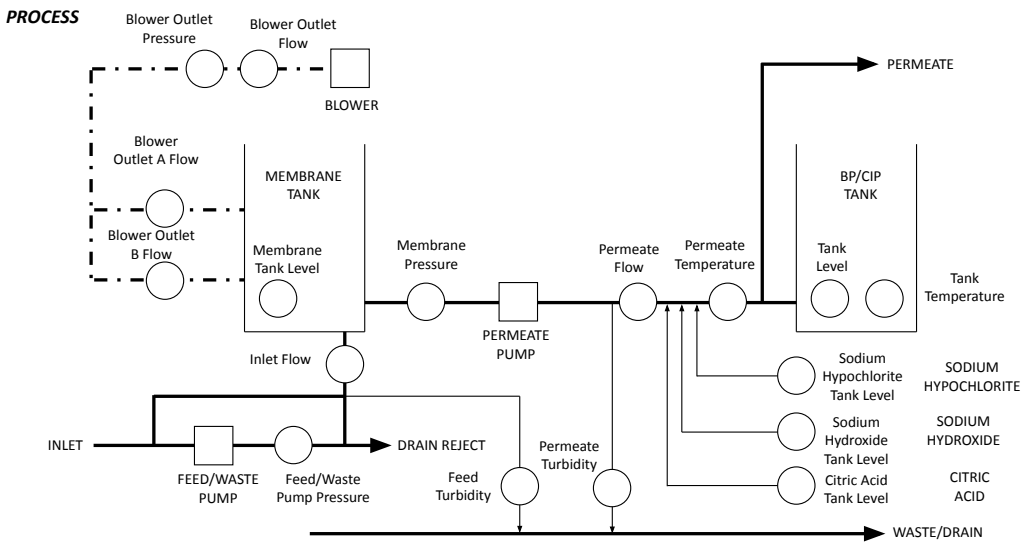

Figure 3.3: Simplified design diagram of the filtration stage of the pilot. Inlet water from the pretreatment stage is filtered in the membrane tank. The filtered water is called permeate. Dashed lines mean water surface, and sensors on the dashed lines take measurements on the surface of the water.

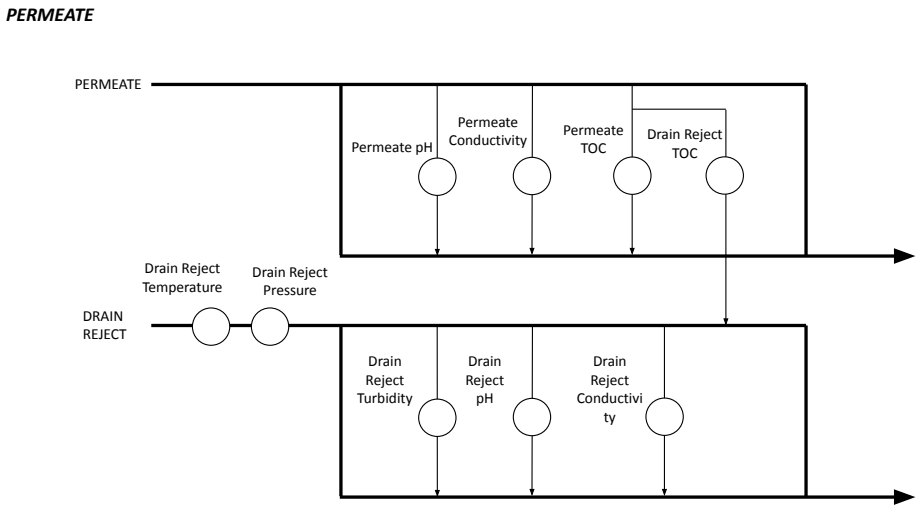

Figure 3.4: Simplified design diagram after the filtration stage. In this stage, pH, conductivity, and TOC are measured against the permeate.

3.3 Software Stack of Pilot WTP

In this section, we describe the software stack created for the pilot.

The software follows the server-client architecture: a server running on a computer physically connected to the pilot accepting incoming connections, and a client program that can be run on any computer with an internet connection and credentials to the server.

The software stack supports the following functionalities:

- Duplex communication between the pilot and the agent
- RL training loop
- Data visualization dashboard for real-time monitoring
- Data logging and backup

We will describe the high-level implementation details for each of them in the following sections.

3.3.1 Bi-directional Communication Between the Pilot and the Agent

In the RL framework, the agent and the environment interact at a fixed interval. Therefore, our software needs to establish bi-directional communication between the pilot and an agent. Our software stack uses the gRPC (Kumar et al., 2016), a Remote Procedure Call framework, to establish and transmit information between the client and the server. The transmission of the sensory values, or observations in the perspective of the agent, is implemented as the *response-streaming RPC*: when the client is ready for digesting new observations, a request is sent to the server, which then responds with a stream of observations. The client does not need to re-send the request after the initial setup, therefore saving the bandwidth. The write actions sent from the client to the server are implemented as a *simple RPC call*: the agent sends the write request to the server, which then responds with a status code that shows whether the operation is successful. If the write request fails, an error message is also provided along with the status code to explain the source of the error.

The gRPC client is implemented in the Connection module, which serves as the gateway of the client-side application, making other modules such as the Environment and the Agent independent of any type of connection.

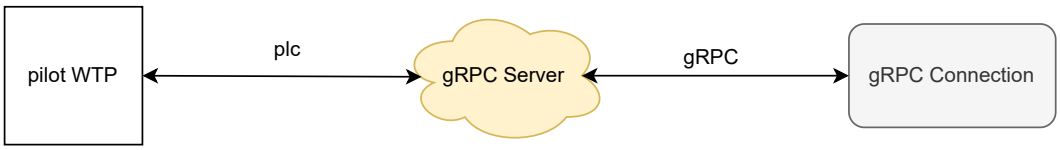

Figure 3.5: Block diagram of the connection to the pilot WTP. This implementation of Connection creates a secure duplex channel to the gRPC server, which runs on a high-performance computer physically connected to the PLC controller of the pilot WTP.

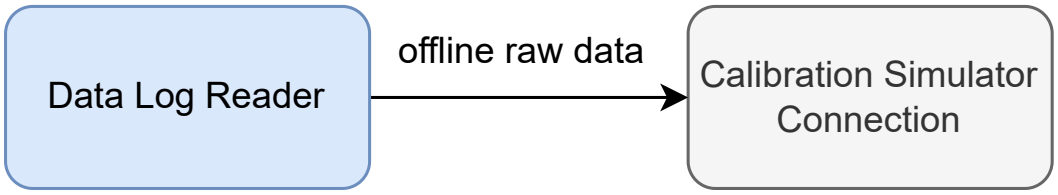

Figure 3.6: Block diagram of the calibration model. The Data Log Reader reads and processes the offline dataset for the Calibration Simulator Connection to create a calibration model.

Figure 3.5 and 3.6 shows two examples of connections: connection to the pilot and to the calibration simulator. Despite the vast difference between the data source and underlying connection technologies, the two connections follow the same interface and therefore can be plugged into the same Environment without any modification in the source code.

3.3.2 RL Training Loop

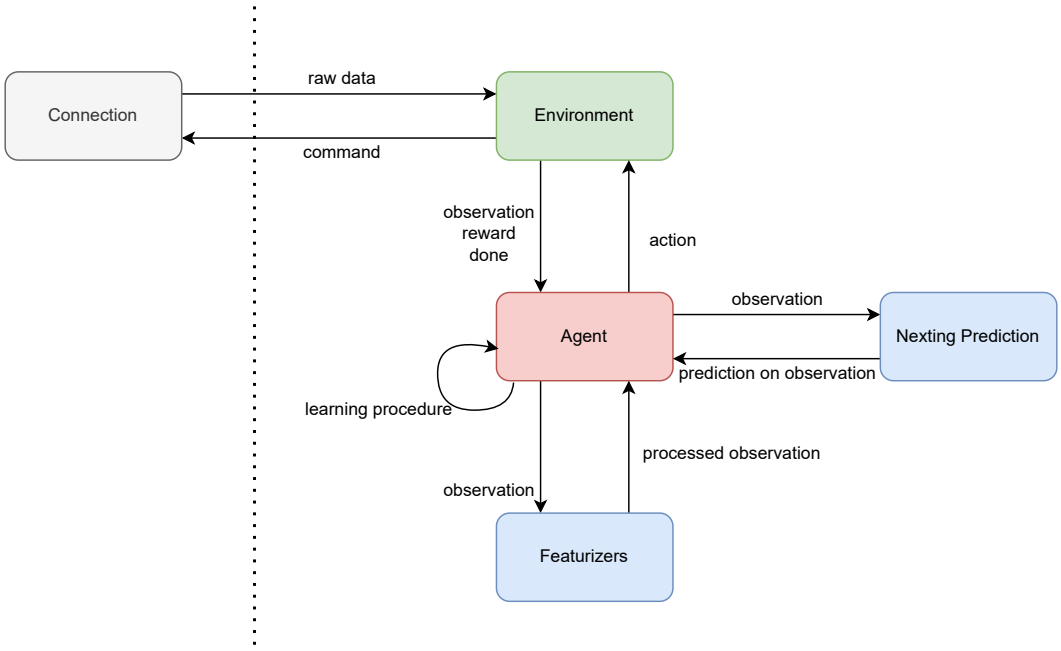

Figure 3.7: Block diagram of the client-side application.

Figure 3.7 shows the design diagram of the client application. The environment runs on the client-side, so we can implement the agent with a similar interface as other popular RL frameworks such as RL-Glue (Tanner & White, 2009) and OpenAI Gym (Brockman et al., 2016), allowing us to port existing algorithm implementations with no or only a few modifications. The Featurizers module handles observation processing, including data normalization and missing value interpolation. The Nexting Prediction module provides Nexting predictions (Clark, 2013 and Modayil et al., 2012) as extra inputs to the agent. The Nexting predictor is pre-trained with offline data and keeps learning online with TD updates. With Nexting predictions as extra inputs, the agent knows not only the current status via observations but also what is going to happen next via Nexting predictions. As a result, we believe that Nexting predictions help the agent learn a better representation of the environment.

The Environment module implements an environment under RL formulation, including defining the observation and action spaces, reward function and termination condition. The Environment module does not handle data

acquisition, but instead delegates this task to the Connection module, which provides an abstraction layer of any kind of connections, including connections to the pilot, to a dummy data source that simply replays the offline logs, or to a data-based simulator (Wang et al., 2022).

We can define multiple RL environments on the pilot plant, as well as many different RL algorithms. To avoid editing the source code manually when we want to change the environment or the agent, we use a configuration file to specify the name of the environment and the agent, as well as all the hyperparameters in the JSON format. The application loads the corresponding components specified in the configuration file dynamically at runtime. This is achieved using registries: at the start of the program, all the Environments, Agents and Connections are registered in the registries with their human-readable names. Then the program loads the corresponding components based on the names specified in the configuration file. This approach makes sharing experimentation setup easy: all the settings are fully specified in the configuration file.

3.3.3 Data Visualization Dashboard for Real-time Monitoring

It is useful to monitor the change of sensory values in the pilot during experiments. For researchers, a dashboard that monitors the learning curve provides real-time information about the performance of the agent. For water treatment engineers and operators, a dashboard that displays various sensory readings helps them determine if interventions of an experiment are necessary. We use Grafana (Grafana Labs, 2014) to build dashboards. Figure 3.8 shows an exemplary dashboard that displays useful information in real-time, including the current operation mode, number of connected agents and sensory plots. Each panel in the dashboard is individually configurable, and users can adjust the refresh rate of the real-time data stream.

Figure 3.8: The Grafana dashboard shows real-time information about the pilot, including current operation mode, number of agents connected and important sensory readings.

3.3.4 Data Logging and Backup

The logged data serves two main purposes: (1) as the historical record of the experiments for calculating metrics and plotting learning curves; (2) used by an RL agent to learn an initial policy before interacting with the pilot. To ensure the timeliness and accuracy of the records, the data logging service runs on the server. A synchronization service runs once a day to upload logs collected for the day to a cloud storage location.

3.4 Sensors

This section describes all the sensors installed on the pilot.

In this thesis, we overload the term "sensor" to describe both passive/active sensors used for measuring a physical quantity, and internal states of the pilot such as setpoint values provided by the plant system. As described in Section 3.1, water treatment consists of multiple stages. The pilot plant has a set of sensors installed in each stage.

3.4.1 Sensor Types

The pilot provides observation of dimension 473, including numerical and categorical values. These sensors can be divided into the following categories:

Pressure

Pressure sensors are installed on the inlet pipe, before the membrane, on the drain-reject pipe, on the blower outlet and at the feed/waste pumps. In water treatment, pumps are often required to increase the feed pressure, and hence pressure sensors are helpful in both providing feedback to the pump controllers, as well as monitoring the plant status for the operators.

Flowmeter

Flowmeters measure the flow rate of the fluid. which is defined as the product of the velocity of the fluid and the cross-sectional area of the pipe.

pH

pH sensors measure the acidity and alkalinity of solutions.

Temperature

Temperature sensors measure the temperature of the water.

Turbidity

Turbidity sensors measure the turbidity of the water. The turbidity sensors consist of two components: an emitter that emits lights into the water, and a receiver that measures the amount of light scattered by suspended solids.

Total organic carbon (TOC)

In the context of water treatment, TOC is a measure of carbon compounds contained in water. While carbon compounds may not be toxic, those containing nitrogen can react with chlorine, which is wildly used in municipal water treatment for disinfection, resulting in less free chlorine and therefore reduced disinfection performance (Cheremisinoff, 2002b). TOC sensors measure the TOC in the water by using the fact that organic compounds absorb a portion of the ultraviolet (UV) light. They have a similar setup to turbidity sensors, however, one significant difference is that TOC sensors emit UV light at 254

nm wavelength and the receiving component measures the percentage of the UV light passed through the water.

Setpoints

Setpoints and modes provide operators with control over the pilot. Operators can write new values to setpoints, and the actuators of the plant will adjust the quantity to the setpoint value using PID control. Some important setpoints include the dosing rate, the mixer fan speed of the flocculation tank and plant mode.

3.4.2 Pilot Operating Cycles

During operation, water treatment plants can enter multiple operation modes. Some typical modes include:

- Production (PROD)
- Standby (STBY)
- Backwash (BW)
- Drain (DRAIN)
- Membrane integration test (MIT)

Due to the small factor of the pilot plant, modes are switched more frequently than on the main plant. During its operation, the pilot loops through the following modes: STBY (10 min) $\rightarrow$ PROD (20 min) $\rightarrow$ BW (1.5 min) $\rightarrow$ DRAIN (5 min) $\rightarrow$ PROD (3 min) $\rightarrow$ STBY (10 min). This forms the normal cycle of the pilot. MIT is scheduled at around 4:30 AM every day, interrupting the normal operating cycle. The MIT mode usually takes 10 minutes. After MIT, the pilot resumes the normal cycle. Since the pilot shares the same inlet pipes with the main plant, its normal cycle may be overridden by the main plant when operators change settings on the main plant manually. Figure 3.9 shows a state diagram of the mode changes of the pilot.

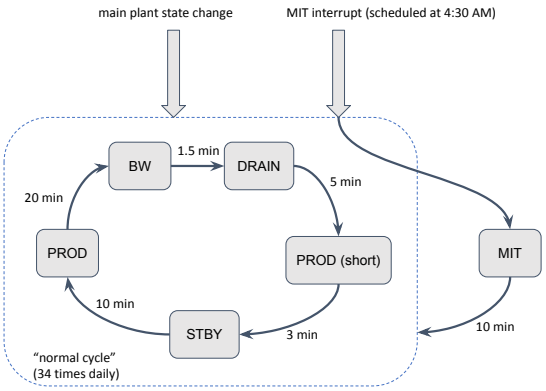

Figure 3.9: State diagram of the pilot plant.

3.4.3 Sensor Patterns in Different Operating Modes

Sensor patterns can be vastly different when the pilot operates in different modes. Figure 3.10 shows the sensor patterns in four different modes: STBY, PROD, BW, and MIT. From the plots, it is clear that some sensors exhibit mode-dependent behaviours, and the pattern and value range can be vastly different. For example, during PRD, on average the permeate temperature drops from 14 °C to 10 °C, whereas during BW we observe an increase from 10 °C to 14 °C.

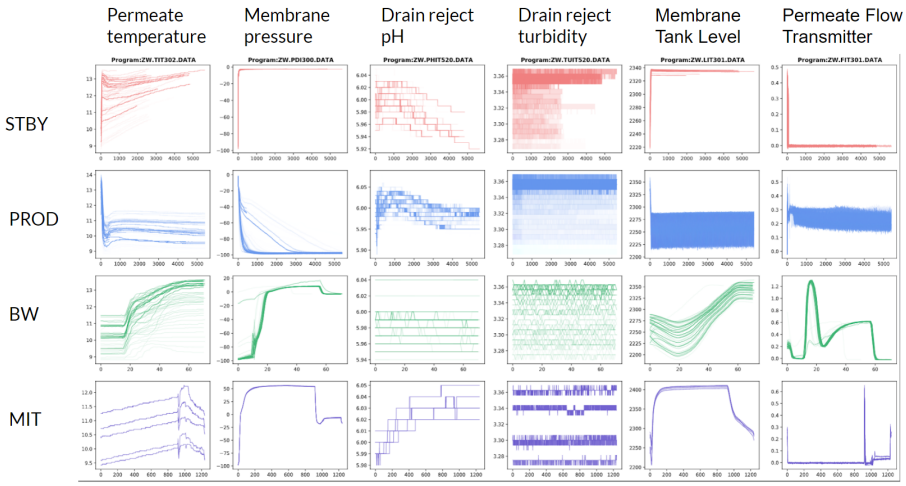

Figure 3.10: Patterns of six sensors for four different operating modes. In each sub-plot, the x-axis shows the time stamp since the start of a mode, and the y-axis shows the sensor value. Each transparent line represents one trajectory, and hence darker region indicates more common patterns. Plots are generated using data collected in 2021.

3.4.4 Temporal Patterns

Some sensory data is temporal-dependent. Among these sensory readings, the inlet temperature is the most representative reading as it has interesting intraday and seasonal patterns. Figure 3.11 illustrates the inlet temperature across a year. The red line is the smoothed value of the raw sensory reading (purple). The inlet temperature climbs up from April (3 °C) to July (24.2°C), before starting to decrease until December. It then stays flat at around 3 °C until early April of the next year. Note that the raw sensory reading contains missing ranges (i.e. around April and around July), as well as abnormal values (around October) due to system error, system maintenance or plant repairs.

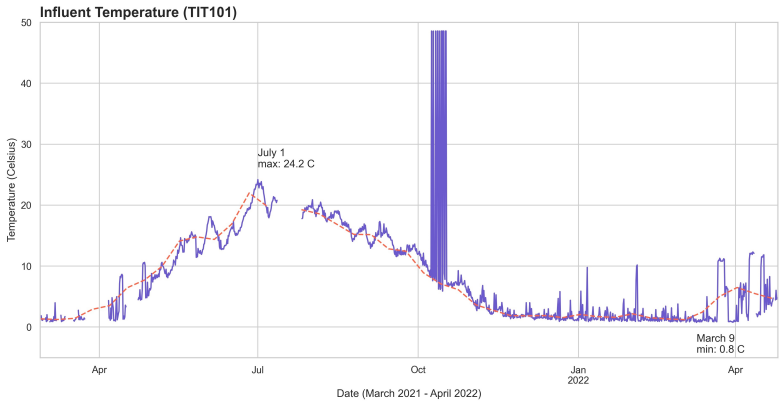

Figure 3.11: Yearly inlet temperature sensor values during 2021.

Figure 3.12 and 3.13 illustrate the difference in daily patterns of the inlet temperature in different seasons. On March 2, 2021, the inlet temperature was mostly below 1.5 °C, with a few occasional peaks that reached between 3 °C and 5 °C. In contrast, on August 21, 2021, inlet temperature was in the range between 16 °C and 18 °C, with a different intraday pattern than that on March 2, 2021.

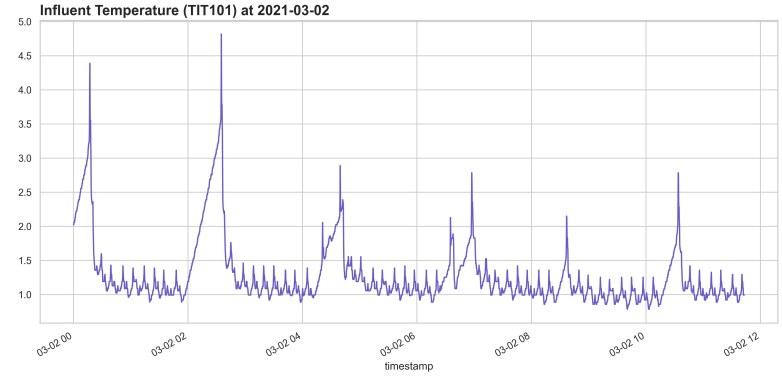

Figure 3.12: Intraday inlet temperature sensor values on March 2, 2021.

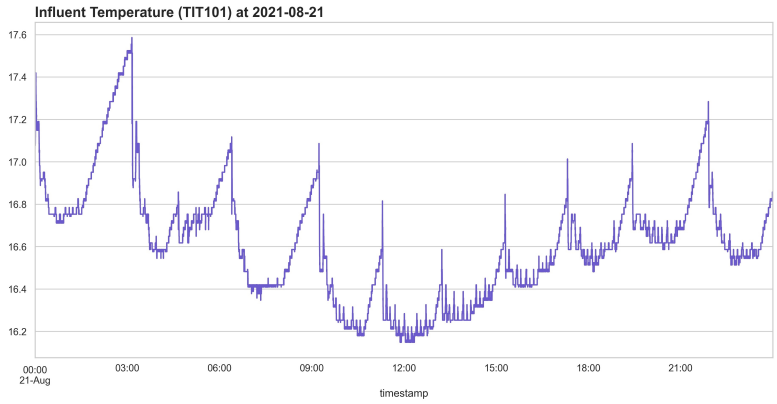

Figure 3.13: Intraday inlet temperature sensor values on August 21, 2021.

3.5 Related Work in WTP Control

There are many optimizing opportunities in different stages of water treatment. In this section, we briefly describe some related work in the control of WTP.

3.5.1 Control the Operating Modes

The main plant uses a fixed BW frequency, which is a common strategy for many WTPs. While easy to implement, fixed BW frequency is inefficient in many cases. When the filter is in good condition or the influent is clear (especially in winter time), frequent BW may lead to low productivity; in other cases, when the filter is old or the influent is very dirty (during freshet or rainy days), filters may degrade caused by BW not being carried out in time. A more efficient BW scheduler should take both the filter condition and influent characteristics into account, and schedule BW accordingly which maximizes the filter life and minimize the energy used during BW.

Dynamic programming (DP) has been used to find the optimal backwash sequence that reduces membrane fouling and hence increases membrane performance and durability (B. Zhang et al., 2020). In this research, a Markov Decision Process (MDP) involving two free variables is first designed and estimated using one year of logged data. With the MDP, the optimal policy is then selected using DP, with a cost function that takes both build-up of membrane

resistance and the cost of backwash into account. For online decision-making, they deploy the fixed optimal policy to make BW decisions. They evaluated the performance of their method against a fixed backwash interval strategy that was deployed on a pilot WTP. They showed that their method resulted in a lower long-term cost than the fixed backwash interval strategy.

3.5.2 Control in Pretreatment

There are many studies about optimizing or modelling the pretreatment stage. One way to approach this task is to train an ANN to predict the effluent turbidity using the chemical dose and water characteristics (Q. J. Zhang et al., 2007). For control, the chemical dosing resulting in the lowest turbidity by sweeping the chemical dose in a range is selected. Another approach by Han et al. (1997) optimizes the coagulant dosing process by modelling the jar test. They utilized the jar test data and trained two models: a fuzzy model and an ANN model. Depending on the water characteristics, one of the models is used to make predictions on the PAC dosing rate. To evaluate their method, they conducted a field test with their methods on a WTP for one year, compared the performance of their model against a conventional regression model, and showed that their method obtained a better accuracy in predicting the PAC dosing from the jar test. However, these approaches cannot adapt to changing conditions in the influent, weather and WTP states that are not presented in the training data.

3.6 Summary

In this chapter, we introduced the water treatment process in general, explaining two major treatment stages: pretreatment and filtration. Then we described the pilot WTP located in Drayton Valley, Alberta, Canada. We also had an overview of the software stack that supports RL experiments, monitoring and data logging. After this, we discussed operating modes and sensors installed on the pilot plant, and how their readings are affected by operating modes and time. We closed the chapter by introducing some related work

about controlling WTPs.

Chapter 4

WTP Challenges

Standard benchmarks have many benefits to the research community. We have seen many advances in RL algorithms recently, and it is common that most of the recent algorithms use standard benchmarks such as classic controls in OpenAI Gym (Brockman et al., 2016), Atari (Bellemare et al., 2013) and MuJoCo (Todorov et al., 2012) for evaluation. Those benchmark environments make the comparison of algorithms easier: environments are standardized, providing fair benchmarks to RL algorithms. Therefore, it is not surprising that many new algorithms use these benchmarks to demonstrate their performances, as well as many algorithmic advances inspired by some characteristics in the simulator environments.[1]

However, real-world industrial control problems are much harder compared to tasks in simulation. In this chapter, we will discuss some challenges of of WTP tasks that make them different from simulated tasks in the benchmarks. We divide the challenges into two groups: challenges that are general in industrial control problems, and challenges specific to WTPs.

[1] Although the environment suites provide a standard benchmark to evaluate RL agents, environments may have hyper-parameters that can be tuned by the user. Some hyper-parameters, such as the number of frames skipped in Atari (Bellemare et al., 2013) affect the performance dramatically (Braylan et al., 2015).

4.1 General challenges for RL in industrial control

In this section, we discuss four challenges faced in industrial control problems: lack of simulators, sensor noise, stable policy and actions at different time scales.

4.1.1 Lack of Simulators

High-fidelity simulators are easy to obtain in games. With advances in computation power, an RL agent has the luxury of on-policy learning with millions of trials in a simulator. For tasks with no simulators available, learning online in the actual environment from scratch is usually restricted or impossible for two reasons:

1. Unsafe actions are prohibited in safety-critical applications, including robotics and healthcare.

2. Learning can take an unacceptably long time due to the control frequency in industrial tasks. For example, the decision of changing the membrane in a WTP happens once per year. Training an online RL agent that controls such tasks from scratch is infeasible.

Therefore, most successful RL applications in industrial control use simulators to overcome the two problems.

Unfortunately, even if we have a simulator that can be used to learn a policy, it might still be challenging to deploy the learned policy in complicated systems because of the *reality gap* (Boeing & Bräunl, 2012): differences between the simulator and the real-world task. There are many sources of discrepancies, including unmodeled dynamics in the simulator, incorrect parameters and numerical errors (Tan et al., 2018). In addition, many simulators are synchronous with the agent: the simulation program pauses and waits for actions from an agent. We will discuss the asynchronous environment in more detail in Section 4.1.5. Such differences could potentially make the sim-to-

real transfer impossible because the discrepancies are accumulated throughout training, causing the agent learns in an environment that is significantly different from the real-world counterpart. Although carefully validated simulators with proper noise modelling have been found to overcome the reality gap (Jakobi et al., 1995), designing detailed and accurate simulators in large industrial projects is often infeasible.

Offline RL, a paradigm that aims to learn a policy from data (Lange et al., 2012), therefore is an important area of study as it can potentially solve the problem of lack of simulator in real-world settings. However, offline RL is empirically harder than the online setting. In the work by Ostrovski et al. (2021), three factors contribute to the difficulty of offline RL: bootstrapping, data distribution and function approximation. The three factors work together to cause and amplify the problem: Insufficient data coverage in the offline data causes incorrect generalization by function approximators, resulting in inaccurate value estimation at less-covered state-actions. The difficulty is amplified by bootstrapping, which further carries the erroneous value estimations to state-actions that are well-covered by the data distribution. In Q-learning methods like DQN, the values are usually over-estimated due to the "max" operation when estimating the target value (Ostrovski et al., 2021).

Other than offline RL, we can construct a data-based simulator extracted from offline data (Wang et al., 2022). The simulator is called a "calibration model", because it works as a low-granularity environment for hyperparameter selection, and can thus "calibrate" the hyperparameters before deployment. We can also use the policy learned in the calibration model as the starting policy for online experiments. The idea is simple: given a list of transitions (s, a, r, s'), we define a distance measure $d((s, a), (s', a'))$. Given the current state s and action a, we compute the distance between (s, a) and all other state-action pairs in the dataset using d. If the distance to the closest neighbour exceeds a threshold, terminate the episode with the default reward. Otherwise, choose a neighbour (s_i, a_i) according to an exponential distribution that puts more weights on closer state-action pairs. With (s_i, a_i), we can retrieve the transition (s_i, a_i, r_i, s'_i) from the dataset. The calibration model

returns the reward r_i and next state s'_i to the agent. This process is then repeated until the termination state is reached. Intuitively, the calibration model stitches trajectories that are close together in the dataset. In the implementation, this process can be optimized by pre-computing the distance and storing all (s, a) and the nearest neighbours within the distance threshold into a hash map before the interaction starts.

4.1.2 Sensor Noise

Sensor noise is common in industrial applications. Some sources of noise include sensors operating in sub-optimal operating conditions, installation errors, missing readings due to maintenance, or the limitation of the sensors themselves.

Noise in Sensors Used to Construct Rewards

Unlike standard RL benchmarks that provide the reward as a separated signal from observations, the reward is often constructed from one or a combination of sensor readings in real-world tasks. Therefore, noise in the sensor readings may be propagated to the reward signal if one or many components in the reward are noisy. To illustrate, Figure 4.1 shows the reward noise and sensor readings used to construct the reward in a PAC dosing control environment on the pilot. It can be seen that noise in the TOC and permeate turbidity sensors (bottom two plots) cause the noise in the reward signal (top plot).

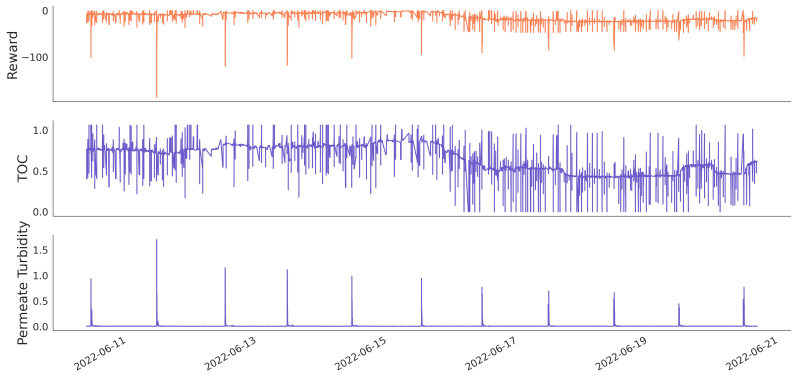

Figure 4.1: Reward and sensor readings used to construct the reward signal in the PAC dosing environment.

In the on-policy setting, noise in rewards causes slow learning in model-free algorithms, because agents need to unlearn the biased estimation with more samples. This effect is more prominent in Q-learning and its variants due to the "max" operator in the update equation (see Section 2.2). The noisy reward problem is even harder in the off-policy setting: the biased value estimate will be eventually unlearned given sufficient samples in the on-policy setting, but the biased value estimate may be kept and never unlearned in the off-policy settings (Sutton & Barto, 2018).

Double learning is a method to avoid maximization bias in Q learning algorithms (Hasselt, 2010). The idea is to separate the parameterized model for estimating the maximum of the true values and producing greedy actions. In double Q learning (van Hasselt et al., 2015), two Q networks parameterized by θ_t and θ'_t are used to construct the target value Y_t^{DoubleQ} :

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a}{\text{argmax}} Q\left(S_{t+1}, a; \boldsymbol{\theta}_t\right); \boldsymbol{\theta}'_t\right).$$

$Q(\cdot, \cdot, \theta_t)$ is used to infer greedy actions, and $Q(\cdot, \cdot, \theta'_t)$ is used to estimate the value of the greedy policy.

Another method that aims to address the noisy reward problem is to estimate the expected value of the reward directly (Romoff et al., 2018). This work suggests that instead of using the reward signal from the environment

which could be noisy or corrupted, we can learn estimation of the mean reward, and use it in the calculation of TD error. The mean reward estimator parameterized by $\theta_{\hat{R}}$ is trained by minimizing the Mean Square Error between the true rewards r_t and estimated rewards $\hat{R}(s_t; \theta_{\hat{R}})$:

$$\mathcal{L}(\theta_{\hat{R}}) = \mathbb{E}\left[\left(r_t - \hat{R}(s_t; \theta_{\hat{R}})\right)^2\right].$$

The authors demonstrated that this approach improves the performance of PPO (Schulman, Wolski, et al., 2017) in a toy MDP problem, five Atari games (Bellemare et al., 2013) and four MuJoCo tasks (Todorov et al., 2012) with artificial stochastic noise in the reward function.

Noise in Observations

A reasonable degree of noise in the input to the model is not a big problem in supervised learning. Studies show that artificially injecting noise into the input can reduce overfitting of the model (Bishop, 1996 and Goodfellow et al., 2016). On the other hand, noises may disturb the observations, causing the agent to make incorrect value estimations. A simple example is a path-finding robot in a maze with noisy localization sensors. The robot may incorrectly locate itself in a position that is on the other side of a wall due to noise, thus assigning values to incorrect states.

4.1.3 Stable Learning in Deployment

Many industrial control tasks require an agent to maintain a stable policy over a long period while being able to adapt to changes in the environment. Therefore, instead of simply deploying a fixed policy, an agent could continue learning online, but the policy should not change drastically. In value-based methods, the policy is inferred from the estimated action-values. While the action-values are updated gradually, a small change may cause the order of the action-value to be different. Since the agent selects the action with the maximum action-value, a small change in action-values could result in a sudden change in the inferred policy. Policy gradient methods, a group of methods that

parameterize the policy directly, have the advantage that ensuring the policy changes gradually (Sutton & Barto, 2018). Trust Region Policy Optimization, or TRPO, is a policy gradient method that puts a constraint on the KL-divergence between the updated policy and the old policy so that the policy does not change dramatically during training, and therefore delivering robust performance in many tasks (Schulman, Levine, et al., 2017).

Another issue that makes the never-ending problem hard with ANN is catastrophic forgetting, a phenomenon that new learning catastrophically erases previous learned information (McCloskey and Cohen, 1989 and French, 1999). Catastrophic forgetting can happen during incremental learning, where tasks are learned by an ANN sequentially. RL is a typical incremental learning task because data arrives in chronological order. Five methods that could mitigate catastrophic forgetting include 1) regularization, 2) ensembling, 3) rehearsal, 4) dual-memory models, and 5) sparsecoding (Kemker et al., 2018).

4.1.4 Handling Actions at Different Time Scales

To achieve human-level intelligence, the ability of decision-making on different time scales is essential. People make decisions on different time scales naturally: the decision of attending university happens at a much slower time scale than the decision of what to eat for lunch. Moreover, decisions at different time scales are common in many industrial tasks. In a WTP, the decision of switching from production to backwash happens every hour, while the decision of changing the fan speed in the flocculation tank happens every second.

Hierarchical Reinforcement Learning (HRL) is a framework that decomposes an RL task into sub-tasks that can be individually learned. The top-level agent can invoke policies learned in sub-tasks as if they are raw actions (Hengst, 2010). One approach to HRL is options (Sutton et al., 1999). An option is defined as a policy and a termination function: $\omega = \langle \pi_\omega, \gamma_\omega \rangle$, where π_ω is the policy of the option, and γ_ω is the termination function that determines termination of the option (Sutton & Barto, 2018). Options are the generalization of actions: a primitive action is a one-step option with a policy of selecting that action and termination at the next step immediately. Unlike a normal

RL agent that selects primitive actions at each step, an agent using options selects one of the options available at the current state, and once an option is selected, subsequent actions are drawn from the policy $A_t \sim \pi_\omega(\cdot, S_t)$, with termination possibility of $1 - \gamma_\omega(S_t)$. Therefore, options enable hierarchical learning at different time scales.

4.1.5 Asynchronous Operations

There are two sources of asynchronization in WTP tasks: asynchronous observations and asynchronous interactions. We will discuss each of them in detail.

Asynchronous Observations

Sensors installed on the WTP have different sampling frequencies. On the pilot, sensors for temperature, pressure and flow rate take measurements every second, while sensors for pH, TOC and turbidity take measurements every five minutes.

One straightforward way to handle the frequency mismatch is to select a control frequency that is the greatest common multiple of sampling frequencies of all the sensors. However, sometimes we need to control the pilot at a higher frequency so that an agent can react to sudden changes in the environment. Therefore, we need to handle delayed observations for sensors with a lower sampling frequency than the control frequency. To overcome this, we can save the last values of low-frequency sensors, and use the saved values for steps before new values are available. For sensors with a higher sampling frequency than the control frequency, we discard intermediate values between two RL steps and only use the most recent values in constructing the observations.

Asynchronous Interactions

Many RL simulation programs are synchronous, meaning that the state of an environment locks until an action is taken by an agent. While this setting is acceptable in games like Chess, Go and Poker, many real-world environments proceed while the agents are learning or making decisions. Moreover,

an action may take time to be realized. As a result, the asynchronous nature of real-world tasks could cause the agent to make decisions based on delayed information about the environment. One approach to overcome the asynchronization is to parallelize the environment interaction, batch sampling and the gradient update such that the learning process does not block the interaction with the environment, and therefore minimize the delay (Yuan & Mahmood, 2022).

4.2 Challenges for RL in WTP

In this section, we discuss some challenges specific to WTP that might not common in other industrial tasks.

4.2.1 Delayed Action Effect

One important control task on the pilot WTP is the control of the PAC dosing rate. The agent is allowed to operate at intervals as short as three seconds. However, the effect of the change in PAC dosing rate on the permeate water can only be observed at least 30 minutes later. If an agent controls with an interval of 3s, the effect of an action can only be observed 600 steps after the action is executed. Moreover, excess PAC due to the high dosing rate causes clogs on the membrane, but it could take days to be observed. Such long delayed effect of actions makes learning difficult, as it is hard for the agent to assign credit for a change in the quality of permeate water to actions 600 steps ago, not to mention assigning credit for the long-term impact of membrane degradation.

The most straightforward method is to increase the operation interval to 30 minutes. The agent is only allowed to send an action to the pilot every 30 minutes, and the dosing rate is kept the same between two actions. By doing this, an agent can observe the impact of the action in the next step immediately. However, increasing the operation interval inversely reduces the number of updates during training. In offline learning with data logs, Increasing the interval results in less data being available. We need to find the balance be-

tween the delayed action effect and training efficiency when looking for the optimal operation interval.

Another approach is that we can re-design the reward function so that it incorporates the delay. However, this can be difficult as it requires an extensive amount of domain knowledge, and reward function designing is known to be difficult and critical to the success of RL applications (Sutton & Barto, 2018).

Alternatively, we build a better representation that contains the information about the delayed effect. One approach is to provide an RL agent with Nexting predictions (Clark, 2013 and Modayil et al., 2012) about the signals used in reward construction. Using the Nexting prediction as an additional input, the agent could map actions to the expected future, thus assigning appropriate credit to its actions. However, all of this depends on the assumption that the Nexting predictions are accurate. Research has shown that an agent can make thousands of Nexting predictions in parallel on a desktop computer when following a fixed behaviour policy (Modayil et al., 2012). However, learning accurate Nexting predictions while learning a good policy is more challenging.

Other techniques that could help with the delayed action effect include: experience replay (Lin, 1992) with a buffer large enough to cover the whole period of the delay, n-step methods (Sutton & Barto, 2018) longer than or equal to the delay steps, add history to the observation of the agent, or providing a proxy for the delayed outcome (Mann et al., 2019).

4.2.2 Multi-dimensional Action Space

In some control tasks, two or more quantities need to be controlled together to achieve optimal performance. One practical task that illustrates this is the PAC dosing rate control at pretreatment of a WTP. There are two knobs: (1) the PAC dosing pump that controls the dosing rate, which consequently determines the PAC concentration in the flocculation tank, and (2) the mixer fan speed of the flocculation tank, which plays an important role in the effectiveness of the flocculation: if the fan speed is too slow, PAC and the inlet water are not evenly mixed. On the other hand, high fan speed could break

apart flocculation just formed. Therefore, an agent needs to choose a proper combination of PAC dosing rate and mixer fan speed to deliver optimal coagulation.

For value-based methods, if the actions are discrete in all dimensions, a simple method is to create a set of new actions that is the cross-product of all dimensions. For example, given a two-dimensional discrete action space, where two actions are available at each dimension $[(A_0, A_1), (B_0, B_1)]$, we can create a new one-dimensional action space with $[C_0 : (A_0, B_0), C_1 : (A_0, B_1), C_2 : (A_1, B_0), C_3 : (A_1, B_1)]$. The disadvantage of this approach is the curse of dimensionality (Bellman, 1957): the size of the new 1D action space grows exponentially with the number of dimensions of the original action space and the number of allowed actions in each dimension. Moreover, this approach only works if all dimensions are discrete.

If all dimensions of the action space are continuous, then policy gradient methods (Sutton & Barto, 2018) may be more suitable for such tasks. Unlike value-based methods that parameterize the value functions, policy-gradient methods parameterize the policy directly. Therefore, the parameterized policy function approximator can output a vector of actions (Lillicrap et al., 2019), or a vector of distribution parameters so that the vector of actions can be sampled from the outputted distribution (Haarnoja et al., 2018). Moreover, an advantage of continuous action spaces is that an agent can generalize over actions. Closer action values in a continuous action space are usually similar, and therefore an agent can not only generalize over the states but also the action space.

Alternatively, we can approach the multi-dimension action space problem using multi-agent RL (MARL). For a task with n dimensions in the action space, we can control each dimension of the action space with one agent, and solve the n-dimension action space task by the cooperation of n RL agents. MARL tasks can be divided into at least three categories: (1) fully cooperative, (2) fully competitive, and mixed. (K. Zhang et al., 2021). The PAC dosing task falls under the fully cooperative case: the agent controlling the PAC dosing rate works together with the agent controlling the mixer speed to maximize

effluent water quality. Both agents share the same reward signal. Team Q-learning (Littman, 2001), a representative MARL algorithm, can be used to solve such cooperative tasks.

4.3 Other Challenges

Other than the challenges that we have identified during our interactions with the pilot WTP, Dulac-Arnold et al. (2021) mentioned some other challenges in industrial control tasks, including following system constraints, partial observability, multi-objective reward functions, and explainable policies. In addition, they provided a suite of RL task software with intentional perturbations that simulate the aforementioned challenges. We believe that such software could help us iterate on new techniques for overcoming the challenges of real-world RL efficiently before deploying them to the pilot WTP.

4.4 Summary

In this chapter, we outlined some general challenges of RL in industrial tasks and specific challenges in WTP. For each challenge, we explain why it could occur, and some solution methods in the literature that we think could help overcome it.

Chapter 5

WTP as a Collection of RL Tasks

In this chapter, we introduce some tasks in the WTP that can be formulated into RL problems. For each task, we identify the objective, the observation and action spaces, as well as the reward function. By decoupling the water treatment process into independent RL tasks that can be solved individually, We believe that it is the starting point toward a fully RL-controlled WTP.

5.1 Controlling the PAC Dosing Rate

This task aims to control the PAC dosing rate in the pretreatment stage of the water treatment process. PAC is a wildly-used coagulant in the pretreatment stage. The addition of PAC increases the rate at which small floating particles coagulate together to form larger particles, resulting in faster settling. Moreover, PAC helps the disinfection indirectly by removing organic matter. Although some organic materials may not be harmful to humans, their ammonium ions and amino acids will react with chlorine, reducing free chlorine available for disinfection (Cheremisinoff, 2002c), which is regulated in many cities.

A study of U.S. WTPs revealed that chemical usage accounts for 16% of the operation and maintenance cost, the third-largest factor after labour and electricity (Roberts et al., 2009). As a result, an intelligent agent that uses fewer chemicals while maintaining quality effluent water has a big financial

impact. Moreover, excessive chemicals in the pretreatment stage flowing to the filtration can cause chemical build-up on the membranes, reducing the membrane lifespan. Currently, the PAC dosing rate on the pilot is selected via jar tests: the influent water is sampled and applied with different chemical doses, and the dose that results in the lowest turbidity is then selected and applied to the main plant. In practice, WTP operators tend to add more PAC on top of the selected dose as an additional buffer to ensure the effluent water meets the regulatory standards. This human-involved process is time-consuming and cannot respond to changes in the influent water rapidly. We want to automate this process by using a self-learning agent to select the proper dosing rate dynamically and online.

5.1.1 Environment Specification

In this section, we define the dosing rate environment for RL.

Observation Space

To provide sufficient observability about the pilot's current state, we analyze how human operators estimate the proper dosing rate as well as the metrics used for measuring effluent water quality. Table 5.1 shows the sensor readings used as the observations of this environment.

Name	Tag Name	Unit	Range
PAC Dosing Rate	PX730	mL/min	0.0 - 10.0
Inlet temperature	TIT101	°C	-50.0 - 50.0
Inlet flow rate	FIT101	L/s	0.0 - 2.0
Permeate flow rate	FIT301	L/s	0.0 - 1.0
Blower outlet flow rate	FIT801	L/s	0.0 - 20.0
Influent turbidity	TUIT101	NTU	0.0 - 10.0
Feed turbidity	TUIT220	NTU	0.0 - 1.0
Permeate turbidity	TUIT310	NTU	0.0 - 1.0
Influent total organic carbon (TOC)	TCIT101	cm^{-1}	0.0 - 100.0
Post-flocculation TOC	TCIT102	cm^{-1}	0.0 - 100.0
Permeate TOC	TCIT510	cm^{-1}	0.0 - 100.0
Current mode	CTRL_P	-	$\{1, \cdots, 12\}$

Table 5.1: Observation ranges of the PAC dosing environment. "Current mode" is a categorical variable that takes 12 distinct values.NTU stands for Nephelometric turbidity unit.

Action Space

Three actions are available to the agent during PROD: decrease the dosing rate, increase the dosing rate or no operation. We choose delta actions over absolute action (i.e. the action is the value of the dosing rate) because we want to have a smoother rate change due to the physical limitation of the PAC dosing pump. The allowed dosing rate range is between 0 and 10 mL/min, and the agent can change 0.5 mL/min per step.

Reward Function

The reward function implicitly defines the behaviour of the RL agents. There are three goals of this task: (1) reducing permeate turbidity to acceptable levels; (2) maintaining enough free chlorine in the permeate water specified by regulation; (3) reducing chemical usage. The reward function needs to take all of the three goals into account. The metrics for goal (1) and goal (3) are easy to obtain as the pilot has sensors that directly measure the turbidity and PAC dosing rate. However, the pilot is not equipped with sensors for permeate free chlorine, and therefore we use the readings of permeate TOC as the proxy to permeate free chlorine in the reward function. Permeate TOC measures the

total organic carbon in the permeate water, which has an inverse relationship with the concentration of free chlorine. The reward function is defined as follows:

$$R = A(\textit{permeate turbidity}) + B(\textit{permeate TOC}) + C(\textit{PAC dosing rate}),$$

where

$$A(x) = \begin{cases} -111.11(x - 0.1) - 5, & \text{if } x \geq 0.1 \\ 0, & \text{otherwise} \end{cases}$$

$$B(x) = \begin{cases} 50(x - 0.9) - 5, & \text{if } x \leq 0.9 \\ 0, & \text{otherwise} \end{cases}$$

$$C(x) = -0.1x.$$

The function of the three components is shown in Figure 5.1. We set the value to zero for the component of TOC and turbidity if they satisfy the permeate regulation requirements. We call it the "safe zone". The reward decreases linearly as the TOC or turbidity deviates away from the safe zone. Note that we explicitly add "gaps" at the edge of the safe zone so that agents are given an extra penalty when sensor values deviate from the safe zone. The maximum value of the reward function is zero when TOC is above 90%, turbidity is below 0.1 NTU and PAC dosing rate at 0 mL/min.

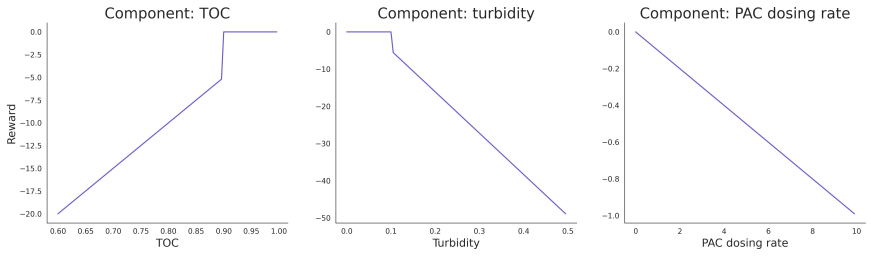

Figure 5.1: Plot of the three components of the reward function of PAC dosing environment. Note that the scale of the y-axis is different in the three plots.

5.2 Controlling Mixer Speed of the Pretreatment Tank

The fan installed at the bottom of the pretreatment tank is used to mix the inlet water with PAC. The speed of the fan is an important factor in the

effectiveness of coagulation. We want a higher fan speed to more efficiently mix the PAC and inlet water. However, high fan speed could harm coagulation, as it may break up the flocculation that has formed. Therefore, an agent needs to find a mixer fan speed that results in optimal coagulation.

5.2.1 Environment Specification

In this section, we define the flocculation mixer speed environment for RL.

Observation Space

The observation space of the mixer speed environment is the same as the observation space of the dosing rate environment, except that the current mixer speed is also included. The mixer speed can take any value within the range $[0, 7]$.

Action Space

There are three choices of action space for the mixer speed environment. We can follow a similar design as the dosing rate environment, i.e. the action space contains three discrete actions: decrease, increase, and no operation. Each step can change the mixer speed by one. Also, we could use absolute actions: create seven actions, each of which represents the desired mix speed. Such a design is possible thanks to the short response time of the mixer fan. Alternatively, we could make the action space continuous by allowing the mixer speed to be any value within $[0, 7]$.

Reward Function

The mixer speed task has three goals that are similar to the dosing rate environment: (1) reduce permeate turbidity to acceptable levels; (2) maintain enough free chlorine in the permeate water specified regulation; (3) reduce electricity usage of the mixer fan. The first two objectives are the same as the dosing rate environment and therefore share the same equations. Although we do not have a direct measure of the electricity usage of the mixer fan, the

energy consumption is proportional to the fan speed. Specifically, the reward function of the mixer speed environment is as follows:

$$R = A(\textit{permeate turbidity}) + B(\textit{permeate TOC}) + C(\textit{mixer fan speed}),$$

where

$$A(x) = \begin{cases} -111.11(x - 0.1) - 5, & \text{if } x \geq 0.1 \\ 0, & \text{otherwise} \end{cases}$$
$$B(x) = \begin{cases} 50(x - 0.9) - 5, & \text{if } x \leq 0.9 \\ 0, & \text{otherwise} \end{cases}$$
$$C(x) = -0.05x.$$

5.3 Backwash Scheduling

The purpose of BW is to flush out fouls formed during production by pushing a portion of the permeate water through the membranes in the reverse direction. The amount of foul accumulated on the membranes is affected by multiple factors, including the inlet water quality, effectiveness of coagulation during pretreatment, and wear and tear of the membranes.

Many WTPs perform BW at a fixed interval. However, this method cannot react to sudden changes in water quality. Over the lifespan of the membrane, constant manual adjustments may be required to apply the proper BW interval based on the membrane status. Moreover, the duration and frequency of BW vary between WTPs, which suggests that the optimal configuration of BW is not found yet, or the optimal configuration is case-specific (Jepsen et al., 2018). Such a finding motivates the use of RL in BW scheduling: instead of fine-tuning the configuration of different WTPs manually, we can deploy the same generic RL algorithm that learns a different configuration for each WTP. In addition, an RL agent can learn an adaptive BW schedule that dynamically changes the BW frequency and duration based on the characteristic of the inlet water.

5.3.1 Environment Specification

In this section, we define the BW scheduling task for RL.

Observation Space

Since there are many factors to be considered in determining the best BW schedule, we provide a rich list of sensory readings from all stages. Table 5.2 shows the types of sensor readings used for the observation. The first section of the table shows the set of sensors that are used in each stage: pretreatment, filtration and permeate. The second section shows the unique sensors or states of the pilot.

Name	Tag Name	Unit	Range
Temperature	TIT	°C	-50.0 - 50.0
Flow rate	FIT	L/s	0.0 - 2.0
Turbidity	TUIT	NTU	0.0 - 10.0
TOC	TCIT	cm^{-1}	0.0 - 100.0
Pressure	PIT	psi	-100.0 - 100.0
pH	PHIT		0 - 14
PAC Dosing Rate	PX730	mL/min	0.0 - 10.0
Current mode	CTRL_P	-	$\{1, \cdots, 12\}$

Table 5.2: Observation ranges of the BW scheduling environment. The first section shows the type of sensors that are read in all stages of the pilot. The PAC dosing rate and the current mode are unique to the pilot. Negative readings of pressure sensors indicate that the water flows from the reverse direction, which could happen during BW.

Action Space

At step t, the agent can take an action $a_t \in \{0, 1\}$, where $a_t = 1$ indicates the start of BW, and $a_t = 0$ indicates switching back to PROD. If the agent tries to switch to the mode that the pilot is currently in, i.e $a_t = 1$ when the current mode is BW, this action results in no change in the current mode.

Reward Function

The reward function follows the design of B. Zhang et al. (2020). The reward function consists of two components: filtration cost and backwash cost. The

filtration cost uses the trans-membrane pressure and filtration flow rate to estimate the membrane resistance, which can be used as a measure of the fouling status of the membrane. The backwash cost is the cost involved during BW. Similar to filtration cost, it is estimated with the trans-membrane pressure and filtration flow rate during BW. In addition, the backwash cost also takes the permeate water used to refill the BW module into account. The reward function is defined as follows:

$$
\begin{aligned}
R &= W_f + aW_b \\
W_f &= TMP_f \cdot FLOW_f \\
W_b &= TMP_b \cdot FLOW_b + P_{re} \cdot FLOW_{re},
\end{aligned}
$$

Where W_f is the filtration power consumption, W_b is the BW power consumption. $a \in \{0, 1\}$ is the action defined in Section 5.3.1. The component of backwash power consumption only contributes to the total reward when the pilot is in the BW mode ($a = 1$). TMP_f is the transmembrane pressure, $FLOW_f$ is the filtration flow rate, TMP_b is the transmembrane pressure during BW, $FLOW_b$ is the filtration flow rate during BW, P_{re} is refilling pressure of the permeate water used during BW, and $FLOW_{re}$ is the refilling flow rate. Among these variables, TMP_f, $FLOW_f$, TMP_b and $FLOW_b$ can be read directly from the observation, and P_{re} and $FLOW_{re}$ can be found on the specification documents of the pilot.

5.4 Summary

This chapter formulates some WTP control tasks into RL environments: PAC dosing rate control, flocculation mixer fan speed control and backwash scheduling. Each RL environment is specified with a clear observation space, action space, and reward function.

Chapter 6

Case Study: PAC Dosing Rate Control with RL

In this chapter, we present a case study on one of the tasks we defined in Chapter 5: poly-aluminum chloride (PAC) dosing rate control. We can control the rate of PAC added to the fluctuation tank, which causes big particles to coagulate so they can be removed in the filtration stage.

Traditionally, PAC dose is determined by jar tests: taking samples of the current inlet water, applying different PAC doses to each sample, and the one that results in the lowest turbidity is selected. Operators then convert the selected PAC concentration into the PAC dosing rate using software that takes the flow rate and the volume of the flocculation tank into account. Operators tend to add excessive PAC to ensure the quality of effluent water. We have reasons to believe that the current PAC rate policy is not optimal due to two reasons. First, jar tests are performed infrequently so the dose rate may not be able to handle a sudden change in the inlet turbidity. An RL agent could make decisions in a much shorter time interval, thus reacting to changes in the inlet water quality in time. Second, The excess dose on top of the result of jar tests added by operators is used as a buffer to handle sudden changes in the inlet water quality. With an RL agent that effectively handles sudden changes, the excess dose can be saved and therefore has an economic benefit.

This chapter is written in chronological order. In Section 6.1, we go through our first attempt at the PAC dosing rate control, present the empirical results, and discuss the result and aspects it fails to model in this problem setting.

In Section 6.2, we present an improved yet more complicated PAC dosing environment that takes defects in the first attempt into account, as well as the current results in this much harder environment.

6.1 Successful Learning on a Simpler Setting

Our first attempt toward the PAC dosing rate control problem only considered the PAC dosing rate and permeate turbidity in the reward function due to the lack of understanding of the TOC sensors and their effect. In this simpler setting, the reward function has two components:

$$R = A(\textit{permeate turbidity}) + C(\textit{PAC dosing rate}),$$

where

$$A(x) = \begin{cases} -114.3(x - 0.3), & \text{if } x \geq 0.3 \\ 0, & \text{otherwise} \end{cases}$$
$$C(x) = -x.$$

Note that this is our first reward function, which is different from the reward function specified in Section 5.1.1. The penalty due to high permeate turbidity only contributes to the reward function when the permeate turbidity is larger than 0.3, which is the maximum allowed turbidity according to the operators and engineers.

6.1.1 Observation Processing

In this section, we describe the processing pipeline to handle the raw observations to be suitable as the input to RL algorithms.

Data Normalization

As shown in Table 5.1, value ranges of sensors are widely different. We used linear scaling to normalize sensor values into $[0, 1]$: given a sensor value x, and the corresponding minimum and maximum values x_{min}, x_{max}, compute the normalized value $x' = (x - x_{min})/(x_{max} - x_{min})$. Some minimum and

maximum sensor values can be found in the documents of the pilot. For other values, we used the empirical maximum and minimum values found in years of logged data.

For categorical values including the current mode and the previous action taken by the agent, we applied one-hot encoding. Specifically, a categorical variable with k categories is converted into a binary vector of size k, in which only the corresponding index of the category is 1.

Time is an important input as it has a strong correlation to some properties of the inlet water such as the temperature. If we directly normalize the date-time values with linear scaling, the normalized value does not have a smooth transition across days: the normalized value starts from 0 at mid-night and goes to 1 near 23:59 of the day, and suddenly drops back to 0 at 00:00. However, the water characteristics at 23:59 may not differ much from 00:00. We solved this problem by encoding the date time s into $[t_{sin}(s), t_{cos}(s)]$ using the following equations:

$$t_{sin}(s) = sin(2\pi s/S)$$
$$t_{cos}(s) = cos(2\pi s/S),$$

where s is the second of the day, $S = 86400$ is a constant representing the total seconds in a day. Note that we did not encode the date of the year due to the scope of this case study.

Missing Sensor Values

We used zero-imputation to handle missing sensory values for three reasons: (1) zero-imputation is a wildly used technique in industrial machine learning problems; (2) zero is within the range of our normalization scheme; (3) it works better with our software implementation: we use Protocol Buffer v3 (Google, 2022) to serialize the data for client-server communication. To save bandwidth, our software stack by design does not serialize and transmit values equal to the default values. Moreover, when a sensor is not available, the corresponding field in the Protocol Buffer is left unset. When the received

Protocol Buffer structure contains an unset field, the client software cannot differentiate between zero or missing values. Therefore, it is a natural choice to use zero-imputation for this case study.

6.1.2 Experiment Setup

The task was designed to be episodic. An episode starts when the pilot enters PROD and terminates when the pilot changes from PROD to other modes. The agent only operates in PROD mode. The control interval was set to three seconds instead of the minimum control interval of one second because we discovered that the PAC pump needs more than one second to change the dose rate to the desired value. The discounted rate γ was 0.99.

6.1.3 Agent

In this experiment, we used Expected Sarsa(0) (van Seijen et al., 2009) with a linear function approximator. the processed observation produced by the procedure described in Section 6.1.1 was directly input to the linear function approximator without extra feature constructions, e.g. polynomials or tile coding (Sutton & Barto, 2018). One weight vector was initialized for each action-value, resulting in three weight vectors to be learned. Stochastic gradient descent was used to perform the gradient update on the weights with a fixed learning rate $\alpha = 0.01$. We used ϵ-greedy policy with a non-decaying $\epsilon = 0.05$. In this experiment, calibration model (Wang et al., 2022) was not used for hyperparameter selection since we have observed that the Expected Sarsa(0) agent converged quickly when running on the pilot due to the simplicity of the task, allowing us to do hyperparameter selection online.

6.1.4 Results

We conducted this experiment during the winter of 2021, a special period when the inlet water was clean because the ice and snow blocked contaminated materials from entering the water source. As a result, the permeate turbidity was always below the regulation standard. Therefore, the optimal strategy was

to never dose PAC at the pretreatment stage. Figure 6.1 shows the average learning curve of the Expected SARSA(0) agent averaged over five runs with the gray area indicating the standard error.

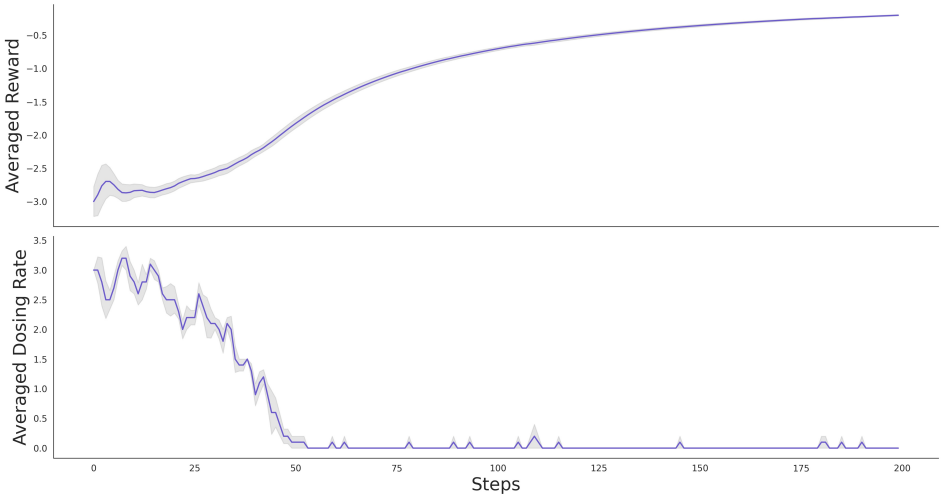

Figure 6.1: Top: learning curve of the Expected Sarsa(0) agent in our first version of the PAC dosing environment. The line represents the mean of exponential average reward over the five runs, and the shadow area shows the standard error. Bottom: The averaged dosing rate set by the Expected Sarsa(0) agent, averaged over five runs. The Shadow area shows the standard error. In five runs the agent effectively learned to decrease the dosing rate to zero, which is the optimal state during the winter time.

6.1.5 Defects of the First Attempt

Although the agent learned the optimal policy in all five runs, this environment failed to capture some important characteristics of the pilot.

Defect 1: not capturing permeate free chlorine in the reward function. In this attempt, we only considered the permeate turbidity and PAC dosing usage in our reward function. After gaining more understanding of the WTP through discussions with water treatment engineers and operators, we discovered that free chlorine is an important metric that is strictly regulated. Therefore, we should incorporate free chlorine into the reward function.

Defect 2: Information in other modes was discarded. The task in this attempt was designed to be episodic since it seemed like a natural choice

as the PAC should only be applied during PROD. However, the pilot has a small flocculation tank and membrane size, therefore the PROD duration is only around 20 minutes (Figure 3.9), whereas other modes in total take around 28 minutes. Ignoring non-PROD modes halves the data for training. In addition, data during non-PROD modes provides meaningful information about the transition dynamics, so it could be beneficial for the agent to train with non-PROD data. Moreover, the episodic setting limits us to select a shorter control interval as longer interval results in fewer data points during an episode. For example, if we set the control interval to be three minutes, a 20-minute episode only contains 6 samples. Learning a meaningful policy from such little data in each episode is nearly infeasible.

Defect 3: Incorrect setpoint being written. There was a bug in this version of the environment. In the pilot system, sensors and setpoints are categorized by sensor types, and each sensor type can have multiple sensor readings and setpoints. There are two setpoints that can be used to control the PAC dosing pump: "DOXMAX" (the maximum dosing permitted) and "AUTO_SP" (the actual applied dosing). We incorrectly wrote values to the "DOSMAX" tag on the pilot plant to change the dosing rate. However, the purpose of this tag is to specify the dosage rate's maximum threshold. While we can indeed control the dosing rate indirectly through this tag, it results in a less-straightforward control scheme.

6.2 The Current Environment

The current version of the PAC dosing rate environment overcomes the aforementioned defects in our first attempt.

We fixed the **Defect 1** by adding a component related to the permeate free chlorine concentration to the reward function. Since we do not have a sensor that directly measures the free chlorine concentration, we use the permeate TOC value as a proxy to the measure of free chlorine as a higher concentration of permeate TOC results in lower free chlorine (see Section 3.4.1). Although permeate TOC is a rough proxy to the free chlorine, it is the best sensor signal

available on the pilot to infer the concentration of free chlorine. However, the introduction of the TOC component in the reward function makes it a much harder problem to solve due to the delayed action effect on TOC (Section 4.2.1). The new reward function now has three components: permeate turbidity, PAC dosing rate and permeate TOC. We also reduced the safe region of the turbidity from [0, 0.3] to [0, 0.1]. This change resembles the practice of the operators. In addition, we added an extra penalty to the non-zero region of the TOC and turbidity component in the reward function. This extra penalty could help the agent learn to stay within the safe region (TOC and turbidity reward components are zero) to avoid the sudden big negative reward when leaving that region. The new reward function is shown in Section 5.1.1.

To overcome **Defect 2**, we switched to the continuing setting. Now the agent does not terminate when the pilot switches out of PROD, but instead continues to interact with the pilot with only one possible action: no-operation. This change allows us to utilize more data from non-PROD modes and select longer control intervals.

The bug described in **Defect 3** is fixed by changing the tag "AUTO_SP" to be writable and using it to control the PAC dosing rate. Figure 6.2 shows the effect on the actual dosing rate due to writing to DOSMAX (left) and AUTO_SP (right).

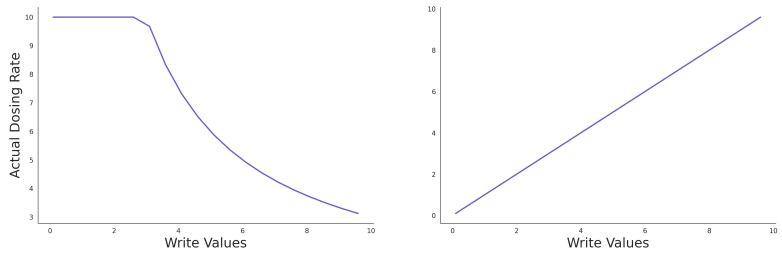

Figure 6.2: Relationship between the actual dosing rate and values written to two tags: DOSMAX (left) and AUTO_SP (right). On the pilot, the actual dosing rate is determined using the following procedure: first, compare DOSMAX and AUTO_SP. If DOSMAX is smaller than AUTO_SP, then the maximum dosing rate (10 mL/min) is applied. Otherwise, the actual dosing rate is 10 × AUTO_SP / DOSMAX mL/min.

6.2.1 Observation Processing

In addition to the procedure described in Section 6.1.1, we added three additional steps to match the increased difficulty introduced in the new problem setting.

Memory Traces on Observations

Since the PAC dosing rate task is only partially observable, i.e. the agent does not have access to the internal dynamics of the system, but only sensory readings, we use memory traces on the normalized observations to provide a compact representation of the history. At time t, the trace e_t, a vector of the same dimension as the normalized observation vectors x'_t, is updated as follows:

$$e_1 = \mathbf{0}$$
$$e_t = \tau e_{t-1} + (1 - \tau)x'_t,$$

where e_1 is the initial trace set to be a vector of zeros. τ is a hyperparameter that determines the decay rate. The traces are updated in all modes.

Action Encoding and Traces

We want to provide the history of actions executed by the RL agent as additional information in the observation, as it increases observability and could potentially help the RL agent build a better representation of the states. Instead of providing the history of actions directly, we design a compact representation of the history of actions. First, we encode the numerical action number into a one-hot vector. Then we apply the same procedure described in Section 6.2.1 to calculate the trace of the encoding of actions. Similar to memory traces on observations, the action traces are updated across all modes.

Binning

After obtaining the traces of the observations and actions, we digitize them using binning with the procedure described in Algorithm 1. Intuitively, the

binning behaves like 1D tile coding (Sutton & Barto, 2018) along each trace, allowing generalization in closer trace values. In this experiment, these bins are used: [0.14, 0.28, 0.42, 0.56, 0.7, 0.84, 0.98].

Algorithm 1 Binning

Input: *bins*: a list of bins of length K
Input: *traces*: a list of traces of observations and action encoding
Output: *output*: a list of binned traces

```
output ← []
for each value e in traces do
    o ← zero vector of length K + 1          ▷ o holds the binned output of e
    if e < bins[0] then                      ▷ e falls below the smallest bin
        o[0] ← 1
    else if e ≥ bins[K − 1] then             ▷ e falls above the largest bin
        o[K] ← 1
    else                                     ▷ e falls between the smallest and the largest
        for i ∈ {1, ..., K − 1} do
            if bins[i − 1] ≤ e < bins[i] then    ▷ Found the bin for e
                o[i] ← 1
                break
            end if
        end for
    end if
    output.append(o)
end for
```

Mode Countdown Encoding

The current mode is one of the most important signals as it affects the internal dynamic of the pilot, which determines the pattern of sensory values such as permeate temperature, trans-membrane pressure and many others (Section 3.4.3). Therefore, being able to know when the mode is about to change gives the agent an advantage in the decision-making. We use *thermometer encoding* as the countdown timer to provide such information to the agent. The procedure of the thermometer encoding is described in Algorithm 2.

Algorithm 2 Mode countdown timer with thermometer encoding

Parameters: n: encoding length, d: a mapping from mode to maximum duration of the mode
Output: *encoding*: the encoding

$m \leftarrow$ the current mode
counter $\leftarrow 0$ ▷ Counts the steps of the current mode
encoding $\leftarrow$ zero vector of length n ▷ The encoding vector
interval $\leftarrow \lceil d[m]/n \rceil$
while not terminating **do**
  if mode changed **then** ▷ Reset variables when mode changed
    $m \leftarrow$ the current mode
    counter $\leftarrow 0$
    encoding $\leftarrow$ zero vector of size n
    interval $\leftarrow \lceil d[m]/n \rceil$
  end if
  counter $\leftarrow$ *counter* $+ 1$
  if *counter*%*interval* $== 0$ **then** ▷ Set the next 1 every *interval* steps
    Set the next 0 to 1 in *encoding*
  end if
end while

Sensor Smoothing

As shown in Figure 4.1, The TOC and turbidity signals are noisy. To reduce the noise in the reward signal, we smooth the TOC and turbidity signals using exponential moving average, and trim off values that are two standard deviations away from the exponential average. The smoothing process is shown in Algorithm 3. For the TOC signal, the decay factor τ is set to 0.9 and *std* is set to 0.062. For the turbidity signal, the decay factor τ is set to 0.8 and *std* is set to 0.027. The decay factor of the TOC signal is larger than that of the turbidity signal because we would like to apply more smoothing to the TOC signal as it is noisier than the turbidity signal. The *std* of both TOC and turbidity are calculated from offline logs.

Algorithm 3 Sensor Smoothing

Parameter: τ: the decay parameter of the exponential average
Input: *std*: the standard deviation calculated from offline logs
Output: $\bar{v}$: the exponential moving average

$\bar{v} \leftarrow$ None
for value v of the sensor signal **do**
  if $\bar{v}$ is None **then**
    $\bar{v} \leftarrow v$ ▷ Initialize $\bar{v}$
  else if $|v - \bar{v}| \leq 2std$ **then** ▷ Check whether value is noise
    $\bar{v} \leftarrow \tau\bar{v} + (1 - \tau)v$ ▷ Exponential moving average
  end if
end for

6.2.2 Agent

We used Double DQN (van Hasselt et al., 2015) as the RL algorithm in the updated PAC dosing task. The reason why this algorithm was selected is that Double DQN is known to reduce the maximization bias, which is prominent in environments with noisy rewards (see Section 4.1.2), plus the team members are familiar with DQN (Mnih et al., 2015) and its variants.

The Double DQN agent utilizes a fully-connected network of two hidden layers; each hidden layer has 64 nodes. All the layers are separated by ReLU activation functions. The output layer maps to the Q values of the three actions allowed during PROD. It does not learn the action-value for the no-operation action in non-PROD modes. Adam optimizer (Kingma & Ba, 2017) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ is used for adaptive gradient update. Updates are performed at every step of the interaction with the environment. We use a replay buffer of size 256. On every step, all the data in the experience buffer is used for the gradient update, i.e. minibatch size of 256, because we want to have more gradient updates between actions to make the most use of the computation power in the online setting. The learning rate is selected from the sweep of $\{0.1, 0.03, 0.01, 0.003, 0.001, 0.0001, 0.00001\}$ using a calibration model (Wang et al., 2022) made from a full year of logged data to select the best learning rate before deployment (for more details about the calibration model, see Section 4.1.1). The selected learning rate $\alpha = 0.0001$ is used during

deployment in the real-world environment.

6.2.3 Current Progress

In this section, we describe the current progress of the PAC dosing task. We first present the result of the experiment with a control interval of three seconds. Then we present the result with a three-minute control interval.

We first applied the Double DQN agent with the aforementioned observation processing procedure on a 3-second control interval. Due to time constraints, we could only collect results from one run of the experiment. Figure 6.3 shows the average reward, PAC dosing rate, TOC and turbidity over this run. As shown in the first figure of the plot, no improvement can be found in over 12k steps, which is equivalent to 10 hours of online training.

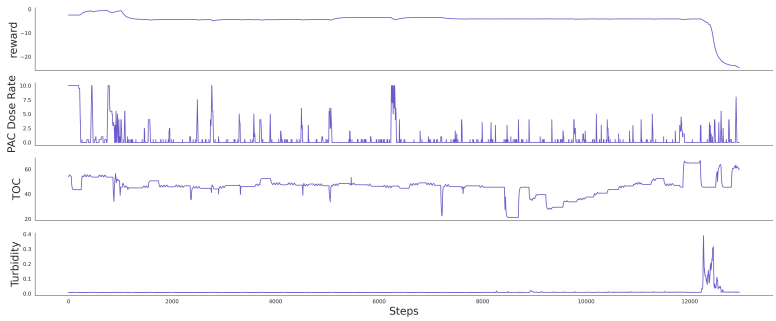

Figure 6.3: Averaged reward curve of the Double DQN agent in the new PAC dosing task (top) and the three sensor readings used to construct the reward function.

We argue that there are two reasons for the unsatisfactory performance: delayed action effect and no pre-training.

Delayed Action Effect on Permeate TOC

The effect of a change in the dosing rate could take a while to be fully observed on the permeate TOC concentration. To demonstrate this, we conducted an experiment that set the PAC dosing rate to a fixed value and observed the change of permeate TOC. Figure 6.4 shows the sensor reading of the PAC dosing rate and the permeate TOC during the experiment. It can be seen that

the full effect of the PAC dosing rate could be fully observed after 3 hours on the permeate TOC, where the curve starts to flatten out. However, the initial change in the permeate TOC can be observed as early as 30 minutes after the change of PAC dosing rate.

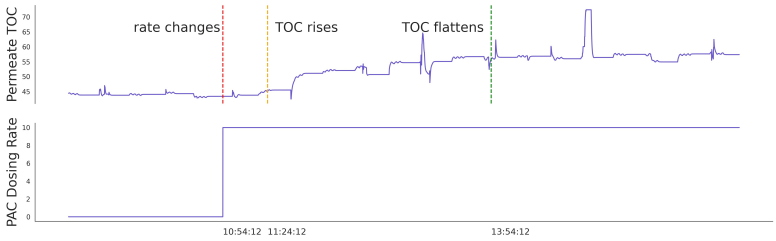

Figure 6.4: Response of permeate TOC to a change of PAC dosing from 0 to 10 mL/min. The red dashed line indicates the time when the PAC dosing rate changes. The orange dashed line shows the time that a rise of permeate TOC can be observed around 30 minutes after the dosing rate change. The green line shows the time when the rise of permeate TOC flattens out, which happens around three hours after the dosing rate change.

When controlling at the three-second interval, the effect of an action is delayed for $30 \times 60 \ / \ 3 = 600$ steps. Such a big delay makes the credit assignment difficult.

No Pre-training

Although we use the offline logs to construct calibration models for hyperparameter selection, we did not use them to help with the learning. Currently, the Double DQN agent is trained from scratch on the pilot directly with random weight initialization, and the replay buffer is initialized empty. Since the data collected during online learning is limited due to time constraints, it is possible that the agent simply required more data to learn a good policy. Pre-training with offline logs may help overcome the limitation of online learning.

Result with Longer Control Interval

To investigate the first hypothesis: the delayed action effect on permeate TOC, we experimented with a three-minute interval. In this setting, the earliest sign

of action effect can be observed after 30 / 3 = 10 steps, which is a reasonable delay for RL. However, the downside of this change is fewer data available for learning. Therefore, we decrease the minibatch size from 256 down to 128 so that the agent can start doing gradient updates earlier. The first subplot of Figure 6.5 shows the learning curve of the Double DQN agent running with a three-minute control interval. Although data available for training is much less than that with three-second intervals, we see improved performance from the learning curve. However, we have to take it with a grain of salt: the agent may be lucky on this run due to the limited running time of the experiment, or the reward increase may be driven by external factors, e.g. the inlet TOC improves due to less rainy days. Longer experiments are required to verify the behaviour of the agent in the future. Allowing an RL agent to distinguish the source of reward between its actions and external factors requires more research.

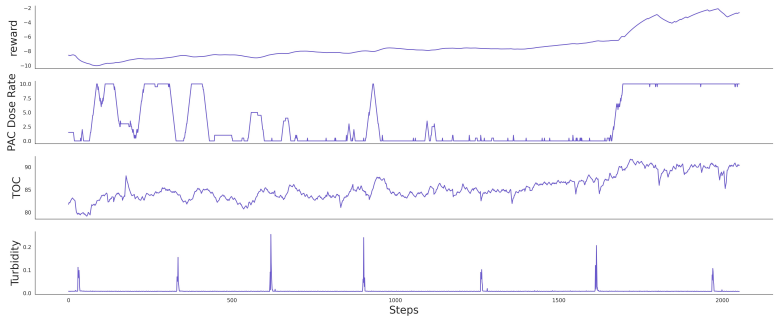

Figure 6.5: The top plot shows the exponential average reward of the Double DQN agent with a control interval of three minutes. The bottom three plots show the sensor values used for constructing the reward function.

6.3 Future Work

We have found some interesting future directions beyond the scope of this case study.

First, we can explore various pre-training methods that make use of the offline logs to obtain an initial policy, or at least make online learning easier by providing a better representation. Here we list two possible pre-training

directions to be explored in the future:

Offline RL. Train a policy offline with the logged data using offline RL algorithms such as Constrained Q learning (Kalweit et al., 2020).

Nexting predictions. Train a network for Nexting predictions from the offline logs. We can either use the second-to-last layer's output as a representation of the observations that are inputted to the action-value network, or we can concatenate the raw observations with Nexting predictions.

Second, we can explore techniques that help with delays of the reward function. We have experimented with longer control intervals to overcome the delay effect at the cost of data efficiency. Methods that can handle the delay effect without sacrificing data efficiency will be beneficial in solving this task more efficiently.

Finally, we should conduct more experiments to verify the result of the three-minute interval experiment. We should run longer experiments or more runs to ensure that the performance is not due to luck. We should also design some baseline policies. Comparing the result three-minute interval experiment with baseline results could help us get a better understanding of the performance. We have proposed two approaches for constructing baselines: (1) consulting operators and creating a procedure that mimics their behaviours; (2) cloning the operations on the main plant controlled by operators to the pilot. We should also find methods to separate average reward gains due to improvements in the agent's policy from the average reward gains due to external factors such as improved inlet water quality. After we can distinguish the source of rewards, we should design algorithms that improve the policy in the presence of external factors that could change the rewards.

6.4 Summary

In this Chapter, we present a case study on one of the RL problems listed in Chapter 5: PAC dosing rate control. We first showed our approach to our initial version of the PAC dosing environment. Although we observed learning, the environment missed some important characteristics that have

to be considered. We then iterated based on these defects and present the current version of the environment, as well as the current progress. For each version, we described the design decisions in both the task specification and the solution methods. Finally, we described some future research directions based on the result of the case study.

Chapter 7

Conclusion

In this thesis, we formulated the water treatment plant control as a collection of RL tasks. We first conducted a deep dive into a pilot WTP that we have access to, describing its operation modes, sensor value patterns and the software architecture that enables duplex communication between the pilot and client programs. We then identified challenges that could make the application of RL on the WTP difficult, including the lack of simulators, sensor noise, actions in different time scales, the stable learning in deployment, delayed action effect and multi-dimensional action space. With the understanding of the pilot WTP and the potential challenges, we decoupled the the whole water treatment process of the pilot into multiple RL tasks that can be solved independently, including controlling the PAC dosing rate, flocculation mixer fan speed, and backwash. For each RL task, we provided the goal, the observation and action spaces and the reward function.

We closed this document with a case study on one of the identified tasks: PAC dosing rate control during pretreatment. We described our progress incrementally. First, we demonstrated a learning agent that effectively solved our first but limited environment. After gaining more insights about the PAC dosing process and its effects on the permeate water, we enriched our environment to overcome the defects, and changed to a more powerful RL algorithm and applied traces on observations and actions to provide a compact representation of the history to the agent. We changed from the episodic setting to the continuing setting to make use of data collected during non-production mode.

We also switched the control interval from three seconds to three minutes to mitigate the delayed action effect.

Finally we pointed out some directions of future development: (1) make use of the offline logs to initialize a policy, or learn a better representation to make online learning easier; (2) explore methods that deal with delayed reward without impairing data efficiency; (3) discover ways to distinguish the sources of reward: improved policy and external factors, and design algorithms that improve the policy in the presence of external factors that may alter the rewards.

References

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., & Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, *588*(7836), 77–82. https://doi.org/10.1038/s41586-020-2939-8. 1

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents [arXiv:1207.4708 [cs]]. *Journal of Artificial Intelligence Research*, *47*, 253–279. https://doi.org/10.1613/jair.3912. 29, 34

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press. Retrieved July 22, 2022, from https://press.princeton.edu/books/paperback/9780691146683/dynamic-programming. 39

Bishop, C. M. (1996). *Neural Networks for Pattern Recognition* (1st edition). Oxford University Press, USA. 34

Boeing, A., & Bräunl, T. (2012). Leveraging multiple simulators for crossing the reality gap. *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, 1113–1119. https://doi.org/10.1109/ICARCV.2012.6485313. 30

Braylan, A., Hollenbeck, M., Meyerson, E., & Miikkulainen, R. (2015). Frame Skip Is a Powerful Parameter for Learning to Play Atari, 2. 29

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym [Number: arXiv:1606.01540 arXiv:1606.01540 [cs]]. Retrieved June 22, 2022, from http://arxiv.org/abs/1606.01540. 18, 29

Chen, C., Ying, V., & Laird, D. (2016). Deep Q-Learning with Recurrent Neural Networks, 6. 7

Cheremisinoff, N. P. (2002a). Chapter 1 - An Overview of Water and Wastewater Treatment. In N. P. Cheremisinoff (Ed.), *Handbook of Water and Wastewater Treatment Technologies* (pp. 1–61). Butterworth-Heinemann. https://doi.org/10.1016/B978-075067498-0/50004-8. 11

Cheremisinoff, N. P. (2002b). Chapter 12 - Treating the Sludge. In N. P. Cheremisinoff (Ed.), *Handbook of Water and Wastewater Treatment Technologies* (pp. 496–600). Butterworth-Heinemann. https://doi.org/10.1016/B978-075067498-0/50015-2. 21

Cheremisinoff, N. P. (2002c). Chapter 2 - What Filtration is All About. In N. P. Cheremisinoff (Ed.), *Handbook of Water and Wastewater Treatment*

Technologies (pp. 62–90). Butterworth-Heinemann. https://doi.org/10.1016/B978-075067498-0/50005-X. 41

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, *36*(3), 181–204. https://doi.org/10.1017/S0140525X12000477. 9, 18, 38

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., ... Riedmiller, M. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning [Number: 7897 Publisher: Nature Publishing Group]. *Nature*, *602*(7897), 414–419. https://doi.org/10.1038/s41586-021-04301-9. 1

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., & Hester, T. (2021). Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis. *Machine Learning*, *110*(9), 2419–2468. https://doi.org/10.1007/s10994-021-05961-4. 40

Ernst, D., Geurts, P., & Wehenkel, L. (2006). Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 54. 3

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, *3*(4), 128–135. https://doi.org/10.1016/S1364-6613(99)01294-2. 35

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* (Illustrated edition). The MIT Press. 34

Google. (2022). Protocol Buffers - Google's data interchange format [original-date: 2014-08-26T15:52:15Z]. Retrieved August 5, 2022, from https://github.com/protocolbuffers/protobuf. 51

Grafana Labs. (2014). Grafana/grafana [original-date: 2013-12-11T15:59:56Z]. Retrieved July 14, 2022, from https://github.com/grafana/grafana. 19

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor [arXiv:1801.01290 [cs, stat]]. Retrieved July 26, 2022, from http://arxiv.org/abs/1801.01290. 39

Han, T.-H., Nahm, E.-S., Woo, K.-B., Kim, C., & Ryu, J.-W. (1997). Optimization of coagulant dosing process in water purification system. *Proceedings of the 36th SICE Annual Conference. International Session Papers*, 1105–1109. https://doi.org/10.1109/SICE.1997.624942. 27

Hasselt, H. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*, *23*. Retrieved August 3, 2022, from https://proceedings.neurips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html. 33

Hausknecht, M., & Stone, P. (2017). Deep Recurrent Q-Learning for Partially Observable MDPs [arXiv:1507.06527 [cs]]. https://doi.org/10.48550/arXiv.1507.06527. 7

Hengst, B. (2010). Hierarchical Reinforcement Learning. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 495–502). Springer US. https://doi.org/10.1007/978-0-387-30164-8_363. 35

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Morán, A. Moreno, J. J. Merelo, & P. Chacón (Eds.), *Advances in Artificial Life* (pp. 704–720). Springer. https://doi.org/10.1007/3-540-59496-5_337. 31

Jepsen, K. L., Bram, M. V., Pedersen, S., & Yang, Z. (2018). Membrane Fouling for Produced Water Treatment: A Review Study From a Process Control Perspective [Number: 7 Publisher: Multidisciplinary Digital Publishing Institute]. *Water*, *10*(7), 847. https://doi.org/10.3390/w10070847. 46

Kalweit, G., Huegle, M., Werling, M., & Boedecker, J. (2020). Deep Constrained Q-learning [arXiv:2003.09398 [cs, stat]]. Retrieved August 17, 2022, from http://arxiv.org/abs/2003.09398. 63

Kemker, R., McClure, M., Abitino, A., Hayes, T., & Kanan, C. (2018). Measuring Catastrophic Forgetting in Neural Networks [Number: 1]. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://doi.org/10.1609/aaai.v32i1.11651. 35

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization [arXiv:1412.6980 [cs]]. https://doi.org/10.48550/arXiv.1412.6980. 59

Kumar, A., Kolhe, J., Ghemawat, S., & Ryan, L. (2016). *gRPC Protocol* (Internet Draft draft-kumar-rtgwg-grpc-protocol-00) [Num Pages: 10]. Internet Engineering Task Force. Retrieved September 9, 2022, from https://datatracker.ietf.org/doc/draft-kumar-rtgwg-grpc-protocol-00. 16

Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. *Advances in Neural Information Processing Systems*, *33*, 1179–1191. Retrieved June 30, 2022, from https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html. 3

Lange, S., Gabel, T., & Riedmiller, M. (2012). Batch Reinforcement Learning. In M. Wiering & M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art* (pp. 45–73). Springer. https://doi.org/10.1007/978-3-642-27645-3_2. 31

Li, X., Li, L., Gao, J., He, X., Chen, J., Deng, L., & He, J. (2015). Recurrent Reinforcement Learning: A Hybrid Approach [arXiv:1509.03044 [cs]]. Retrieved July 19, 2022, from http://arxiv.org/abs/1509.03044. 7

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2019). Continuous control with deep reinforcement learning [arXiv:1509.02971 [cs, stat]]. https://doi.org/10.48550/arXiv.1509.02971. 39

Lin, L.-J. (1992). *Reinforcement learning for robots using neural networks* (phd) [UMI Order No. GAX93-22750]. Carnegie Mellon University. USA. 38

Littman, M. L. (2001). Value-function reinforcement learning in Markov games. 40

Mann, T. A., Gowal, S., György, A., Jiang, R., Hu, H., Lakshminarayanan, B., & Srinivasan, P. (2019). Learning from Delayed Outcomes via Proxies with Applications to Recommender Systems [arXiv:1807.09387 [cs, stat]]. Retrieved August 15, 2022, from http://arxiv.org/abs/1807.09387. 38

McCloskey, M., & Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In G. H. Bower (Ed.), *Psychology of Learning and Motivation* (pp. 109–165). Academic Press. https://doi.org/10.1016/S0079-7421(08)60536-8. 35

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning [Number: 7540 Publisher: Nature Publishing Group]. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236. 1, 8, 59

Modayil, J., White, A., & Sutton, R. (2012). Multi-timescale Nexting in a Reinforcement Learning Robot, 11. 9, 10, 18, 38

Ostrovski, G., Castro, P. S., & Dabney, W. (2021). The Difficulty of Passive Learning in Deep Reinforcement Learning. *Advances in Neural Information Processing Systems*, *34*, 23283–23295. Retrieved July 15, 2022, from https://proceedings.neurips.cc/paper/2021/hash/c3e0c62ee91db8dc7382bde7419bb573-Abstract.html. 31

Roberts, D., Kubel, D., & Carrie, A. (2009). *Costs and Benefits of Complete Water Treatment Plant Automation*. AwwaRF. 41

Romoff, J., Henderson, P., Piché, A., Francois-Lavet, V., & Pineau, J. (2018). Reward Estimation for Variance Reduction in Deep Reinforcement Learning [arXiv:1805.03359 [cs, stat]]. Retrieved July 19, 2022, from http://arxiv.org/abs/1805.03359. 33

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). Trust Region Policy Optimization [arXiv:1502.05477 [cs]]. Retrieved August 3, 2022, from http://arxiv.org/abs/1502.05477. 35

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms [arXiv:1707.06347 [cs]]. Retrieved July 19, 2022, from http://arxiv.org/abs/1707.06347. 34

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge [Number: 7676 Publisher: Nature Publishing Group]. *Nature*, *550*(7676), 354–359. https://doi.org/10.1038/nature24270. 1

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44. https://doi.org/10.1007/BF00115009. 9

Sutton, R., & Barto, A. (2018). *Reinforcement learning: An introduction* (Second edition). The MIT Press. 1, 5, 33, 35, 38, 39, 52, 5

Sutton, R., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., & Precup, D. (2011). Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction, 8. 7, 9

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1-2), 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1. 35

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., & Vanhoucke, V. (2018). Sim-to-Real: Learning Agile Locomotion For Quadruped Robots [arXiv:1804.10332 [cs]]. Retrieved August 3, 2022, from http://arxiv.org/abs/1804.10332. 30

Tanner, B., & White, A. (2009). RL-Glue: Language-Independent Software for Reinforcement-Learning Experiments, 4. 18

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control [ISSN: 2153-0866]. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109. 29, 34

van Hasselt, H., Guez, A., & Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning [arXiv:1509.06461 [cs] version: 3]. Retrieved June 29, 2022, from http://arxiv.org/abs/1509.06461. 33, 59

van Seijen, H., van Hasselt, H., Whiteson, S., & Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa [ISSN: 2325-1867]. *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 177–184. https://doi.org/10.1109/ADPRL.2009.4927542. 52

Wang, H., Sakhadeo, A., White, A. M., Bell, J. M., Liu, V., Zhao, X., Liu, P., Kozuno, T., Fyshe, A., & White, M. (2022). No More Pesky Hyperparameters: Offline Hyperparameter Tuning for RL. Retrieved August 12, 2022, from https://openreview.net/forum?id=AiOUi3440V. 3, 19, 31, 52, 59

Watkins, C. (1989). *Learning From Delayed Rewards* (Doctoral dissertation). 6

World Health Organization. (2022). Drinking-water. Retrieved June 21, 2022, from https://www.who.int/news-room/fact-sheets/detail/drinking-water. 2

Yuan, Y., & Mahmood, A. R. (2022). Asynchronous Reinforcement Learning for Real-Time Control of Physical Robots [arXiv:2203.12759 [cs]]. Retrieved August 3, 2022, from http://arxiv.org/abs/2203.12759. 37

Zhang, B., Kotsalis, G., Khan, J., Xiong, Z., Igou, T., Lan, G., & Chen, Y. (2020). Backwash sequence optimization of a pilot-scale ultrafiltration membrane system using data-driven modeling for parameter forecasting. *Journal of Membrane Science*, *612*, 118464. https://doi.org/10.1016/j.memsci.2020.118464. 26, 47

Zhang, K., Yang, Z., & Başar, T. (2021). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In K. G. Vamvoudakis,

Y. Wan, F. L. Lewis, & D. Cansever (Eds.), *Handbook of Reinforcement Learning and Control* (pp. 321–384). Springer International Publishing. https://doi.org/10.1007/978-3-030-60990-0_12. 39

Zhang, Q. J., Shariff, R., Smith, D. W., Cudrak, A., & Stanley, S. J. (2007). Artificial neural network real-time process control system for small utilities [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1551-8833.2007.tb07961.x]. *Journal AWWA*, *99*(6), 132–144. https://doi.org/10.1002/j.1551-8833.2007.tb07961.x. 27