Decentralized Operation of Urban Water Distribution Networks Using Multi-Agent Reinforcement Learning

Dezentraler Betrieb urbaner Wasserverteilungsnetzwerke mittels Multi-Agent Reinforcement Learning
Bachelor thesis in the department of Computer Science by Jan Sammet
Date of submission: April 12, 2025

1. Review: Prof. Jan Peters, Ph.D.
2. Review: Prof. Dr.-Ing. Peter Pelz
3. Review: Aryaman Reddi, M.Eng.
4. Review: Katharina Henn, M.Sc.
5. Review: Julius Breuer, M.Sc.
6. Review: Ning Xia, M.Sc.

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Jan Sammet, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 12. April 2025

J. Sammet

Abstract

This thesis explores the use of Multi-Agent Reinforcement Learning (MARL) for the efficient operation of urban Water Distribution Networks (WDNs). While previous work focused on industrial networks with hierarchical topologies, we extend the application of MARL to urban WDNs with cyclic structures and more complex flow dynamics. Following the Centralized Training for Decentralized Execution (CTDE) paradigm, we investigate the performance of two deep MARL algorithms, IDDPG and FACMAC, across a range of control tasks.

We evaluate the impact of different reward structures on energy efficiency and demonstrate that a multifaceted reward function, combining demand deviation, power penalties, and a target for maximum valve openings, leads to the most robust and efficient agent behavior. Our results further highlight the importance of observability: fully observable agents achieve the highest efficiency and generalize better to unseen scenarios, while partially observable agents face a trade-off between efficiency and robustness depending on training duration.

Finally, we assess the importance of centralized learning in cooperative tasks with temporal planning. FACMAC, leveraging centralized training, enables agents to coordinate pump and tank operations to preserve water for peak demand hours, outperforming IDDPG's decentralized learning approach. These findings demonstrate the potential of MARL to enable smart and sustainable control of urban water infrastructure.

Zusammenfassung

Diese Arbeit untersucht den Einsatz von Multi-Agent Reinforcement Learning (MARL) zur effizienten Steuerung urbaner Wasserversorgungsnetze (Water Distribution Networks, WDNs). Während bisherige Studien sich hauptsächlich auf industrielle Netze mit hierarchischen Topologien konzentrierten, erweitern wir den Anwendungsbereich von MARL auf städtische WDNs mit zyklischen Strukturen und komplexeren Strömungsdynamiken. Unter dem Paradigma Centralized Training for Decentralized Execution (CTDE) vergleichen wir die Leistung zweier MARL-Algorithmen, IDDPG und FACMAC, in verschiedenen Steuerungsaufgaben.

Wir bewerten den Einfluss unterschiedlicher Belohnungsstrukturen auf die Energieeffizienz und zeigen, dass eine vielseitige Belohnungsfunktion, bestehend aus der Differenz zwischen Nachfrage und Versorgung, Leistungskosten und einem Zielwert für maximale Ventilöffnungen, zu besonders robustem und effizientem Verhalten der Agenten führt. Unsere Ergebnisse verdeutlichen zudem die Relevanz der Sichtbarkeit: Agenten mit vollständiger Sicht erzielen die höchste Effizienz und generalisieren besser auf unbekannte Szenarien, während Agenten mit eingeschränkter Sicht in Abhängigkeit der Trainingsdauer einen Kompromiss zwischen Effizienz und Robustheit treffen müssen.

Abschließend untersuchen wir die Bedeutung zentralisierten Lernens bei Aufgaben mit einem Fokus auf langfristiger Planung und Kooperation. FACMAC ermöglicht es den Agenten durch zentrales Training, Pumpen- und Tankbetrieb so zu koordinieren, dass Wasser für Zeiten hoher Nachfrage gespeichert wird, eine Fähigkeit, die IDDPG aufgrund seines dezentralen Lernansatzes nicht zuverlässig entwickelt. Diese Ergebnisse zeigen das Potenzial von MARL zur Umsetzung intelligenter und nachhaltiger Strategien im Betrieb urbaner Wasserinfrastrukturen.

Contents

1. Introduction

Water distribution networks (WDNs) are an important part of urban infrastructure. Especially in the context of climate change, with droughts and periods of heavy rainfall, an effective operation is more crucial than ever. In addition, older networks need to be adapted to an increasing demand, as some cities have significantly outgrown the networks' original designs. While demand fulfillment is a priority for the operation of WDNs, other factors have to be considered as well. Operation strategies should be energy efficient and adaptable to a multitude of different network architectures. They should be robust, ensuring demand fulfillment even under adverse conditions. Finally, the operation should be cost-efficient, considering not only maintenance costs, but also the costs associated with the development and implementation of a strategy.

Conventional strategies for controlling WDNs are based on simple heuristics on a local level of the individual components. In our work we consider mainly the operation of pumps and valves. Typically, the pumps hold a certain pressure level in the system, ensuring that any demand can be met. Global and network-specific topological information is often not considered. This can lead to increased power consumption of the pumps, as they can be under high load in scenarios where lesser pressure levels throughout the network could be sufficient as well.

In contrast, optimal control strategies designed for particular network architectures can significantly improve the efficiency, by leveraging information from the whole network to choose an optimal pump speed. However, their flexibility is limited, as they are designed around a specific model of a network. When that model is not applicable anymore, the strategy becomes inefficient. Furthermore, they require communication between components, which can be difficult and expensive to implement in real-world applications.

Multi-agent reinforcement learning (MARL) could form a balance between these approaches. Following the Centralized Training for Decentralized Execution (CTDE) paradigm [1], cooperation between agents can be improved, while still ensuring a decentralized execution for each agent, removing the necessity for component communication. In this thesis we aim to demonstrate that MARL has the ability to control WDNs effectively by achieving high energy efficiency, due to its abilities to learn the key characteristics of different tasks and networks and to develop dynamic and effective strategies. Specifically, the performance of the two deep multi-agent reinforcement learning algorithms IDDPG [2] and FACMAC [3] are investigated and tested on suitable tasks.

MARL, among other multi-agent control strategies, has previously been investigated as a method to manage hierarchical fluid networks typically found in industrial WDNs [4]. These earlier studies primarily focused on hierarchical topologies with straightforward flow patterns and limited inter-dependencies. In this work, we extend the application of MARL to more complex network structures that include network cycles and tanks, which are more representative of urban WDNs. These networks introduce additional challenges in terms of flow dynamics and pressure management.

We evaluate the performance of MARL regarding its efficiency, as well as its ability to coordinate cooperative tasks. The evaluation is structured across progressively complex scenarios, beginning with basic tasks focused on satisfying water demand — a fundamental requirement of any WDN. These initial experiments serve as a starting point and test the agents' ability to learn stable policies under simple conditions. We then introduce energy metrics and train the agents to aim for high efficiency. In this context, different approaches for reward signals are discussed and evaluated.

A key research question of this study is the investigation of observability conditions and their impact on efficiency. To be exact, we compare fully observable settings, where each agent has global knowledge of the environment, with partially observable scenarios, where agents rely only on local information.

Finally, we present the agents with the advanced task of nightly water preservation to ensure sufficient supply during the following day. This scenario introduces a temporal dimension to the problem and tests the agents' ability to plan ahead as well as their ability to coordinate pumps and tanks and cooperate effectively.

In the following chapters, we begin with the foundations of water distribution networks and reinforcement learning in chapter 2. We then present our methodology, including the simulation, the training environment and our Neural Networks in chapter 3. After that, we describe the experimental setup, covering the models of the water distribution networks used (chap. 4). We then analyze the results of our experiments (chap. 5). Finally, we conclude with a summary (chap. 6) and an outlook on future work (chap. 7).

2. Foundations

2.1. Water Distribution Networks

A water supply system can be generically divided into the water extraction and transport, the water treatment and storage, and into clear water transport and distribution [5]. Abiding by that definition urban WDNs are critical infrastructure systems that deliver clean water from sources, such as reservoirs, wells, or treatment plants, to end-users in urban areas. They are designed to maintain continuous water supply under varying demand conditions.

A typical WDN consists of a range of interconnected components, including pipes, junctions, pumps, valves, and storage tanks. Pipes and junctions form the structural backbone of the network, directing water through the topology. Pumps and valves, as active components, play a key role in regulating the flow and availability of water. Pumps provide hydraulic power to increase pressure and flow rate, ensuring water reaches all areas of the network. Valves, on the other hand, control the direction, pressure levels, and flow rates by adjusting their opening and closing states, thereby dynamically managing distribution.

Tanks serve as storage units and play a critical role in pressure management. When positioned at elevations higher than the rest of the network, tanks can utilize gravitational potential energy to maintain or boost system pressure, supplementing or temporarily replacing the need for pump operation. This strategic use of elevation not only supports energy efficiency but also enhances the resilience of the system during peak demand or emergency conditions.

For simulation purposes, non-linear models are employed and solved using iterative solvers. Even in steady-state simulations, which already neglect transient dynamics, the computational load can be significant. To address this, techniques such as skeletonization are applied to reduce network complexity [6]. This is particularly important in urban WDNs, where the number of individual consumers can easily reach into the tens of thousands. In such cases, consumers with similar demand patterns are aggregated into representative nodes to simplify the model.

2.2. Reinforcement Learning

Reinforcement learning (RL) is a fundamental paradigm within machine learning that focuses on the problem of learning control strategies through trial-and-error interactions between an agent and its environment, which is typically formalized as a Markov Decision Process (MDP) [7].

An MDP is formally defined as a 4-tuple (S, A, T, R), where S denotes the state space, and A denotes the action space. Both can be either discrete or continuous. The state transition dynamics are given by $T_a(s, s')$, the probability of transitioning from state s to state s' under action a. Each transition is associated with an immediate reward, defined by the function $R(s, s')$.

In an MDP, a policy π can be specified, that explicitly maps each state s to an action $a = \pi(s)$. The goal in reinforcement learning is to find a optimal policy that maximizes a cumulative function of the rewards gained for the chosen transitions.

In order to find an optimal strategy, a combination of exploration and exploitation is used. Exploration involves selecting actions that may deviate from the current policy in order to discover potentially better strategies. In contrast, exploitation leverages the agent's existing knowledge to select actions that are expected to yield high cumulative rewards based on past experience. Evidently, the two approaches pose a trade-off.

The implementation of exploration strategies is generally uncomplicated. In discrete action spaces, exploration is often implemented by occasionally selecting random actions. In continuous action spaces, a common approach is to add noise, such as Gaussian noise, to the actions given by the policy. The trade-off of exploration and exploitation can be controlled by adjusting the magnitude of this noise, for example, by scaling the standard deviation of the Gaussian distribution. In discrete action spaces, the exploration can be controlled by varying the frequency of selecting random actions, as seen in ε-greedy policies [7].

The process by which agents learn from their experience is not as straightforward. Numerous algorithms exist, each employing different approaches to the aforementioned problem. Given the extensive range of methods, a comprehensive summary of all algorithms is beyond the scope of this thesis. Therefore, the focus will be on Actor-Critic methods, as they are most relevant to our study.

2.2.1. Actor-Critic Algorithms

Actor-Critic methods address the problem of learning with a split approach. They combine policy-based techniques with value-based techniques with their two components, the actor and the critic. As the name suggests, the actor is responsible for learning the policy, and thereby which actions to take. The critic assesses the actions taken by the actor, so that the latter can learn from the feedback. Both components train together.

To assess the actions of the actor, the critic approximates the action-value function $Q(s, a)$, mapping each state-action pair (s, a) to a q-value q. The q-value represents the discounted cumulative reward given by the Bellman Equation, defining the optimal action-value function as:

$$Q_{\text{optimal}}(s,a) = \mathop{\mathbb{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q_{\text{optimal}}(s',a')\right] \tag{2.1}$$

where $r(s, a)$ represents the immediate reward for choosing action a in state s and with the next state s' being drawn from a distribution P of the environment.

DDPG

Deep deterministic policy gradient (DDPG) [2, 8] is an algorithm extended from deterministic policy gradient (DPG) [9] for the deep-RL setting. It uses Neural Networks to approximate the policy and the action-value function. For the critic, the loss function

$$L(\phi, B) = \left(Q_\phi(s,a) - \left(r + \gamma(1-t)Q_{\phi\text{target}}\left(s', \mu_{\theta\text{target}}(s')\right)\right)\right)^2 \tag{2.2}$$

can be defined, extending equation 2.1 with a mean squared error, the neural networks for the actor μ_θ, $\mu_{\theta\text{target}}$ and the critic Q_ϕ and $Q_{\phi\text{target}}$ and a batch of transitions $B = (s, a, r, s', t)$ taken from the replay buffer. The replay buffer contains transitions from past experience, including the original state s, the action a taken in that state, the immediate reward r and the resulting next state s'. Additionally, it contains a boolean flag (interpreted as 1 or 0) indicating terminal states, which is used to determine when no additional rewards have to be considered.

Furthermore, the action maximizing the action-value function is now chosen by the actor target network. The target networks are initialized to be an exact copy of their corresponding neural network. Over the training they are updated with the soft-update rule

$$\phi_\text{target} \leftarrow \tau\phi + (1-\tau)\phi_\text{target} \tag{2.3}$$

to stabilize training.

The actor can then be trained on the q-values provided by the critic. As the q-value represents the quality of a state-action pair, the actor should be optimized to maximize the q-values, resulting in the loss

$$L(\theta, B) = -Q_{\phi\text{target}}\left(s, \mu_\theta\left(s\right)\right). \tag{2.4}$$

IDDPG

As we consider only multi-agent systems[10], we primarily use indipendent DDGP (IDDPG), which is the natural extension of DDPG to multiple agents. In theory, any RL algorithm can be used for multi-agent problems. Each agent then trains independently on their own observations, completely unaware of the other agents in the system. This is the case for IDDPG as well, where each agent uses DDPG with its own seperate networks. Independent learning introduces several issues. As the agents are not aware of each other, the environment becomes non-stationary from their point of view. As a result, the Markov assumption, that the future of the system is entirely dependent on the state, is violated. The agent's assume the other agents to be part of the environment and have no way of predicting their change in action caused by individual policy updates. This can cause instability and convergence issues. The latter is aided by the exceedingly large joint action space, as the agents can struggle to explore effectively, if each agent learns by themselves. Finally, a problem of multi-agent systems is credit-assignment. Agents can never be sure, whether their actions were aided or sabotaged by the others.

FACMAC

To address the issues of multi-agent systems, central learning techniques can be applied. In multi-agent DDPG (MADDPG) [11] the critics are supplied with not just the observations of the individual agent but with global system knowledge, meaning all observations and the actions of all agents. Central learning techniques benefit from the actor-critic method, as the critic can be supplied with arbitrary information during learning, while the actor will always be able to act independently based on its observations. While MADDPG compensates the problem of non-stationarity using the global state and actions, the problem of challenging exploration and credit-assignment remains.

Factored multi-agent centralized policy gradient (FACMAC) [3] extends the critics even further. Instead of a centralized critic per agent, the agents are supplied with a central mixing network. The q-values of the individual independent critics are then passed to the mixing network which has knowledge about the global state. It then mixes the q-values together, weighting each q-value based on its global knowledge. The resulting total q-value is then used as a loss for backpropagation, similarly to DDPG. The decisive difference lies in the central mixing network which can generate more meaningful gradients. As the individual q-values can be weighted, it solves the credit assignment problem and as the central mixing network is able to generate meaningful gradients for the individual critics, even if the agents did not successfully cooperate, exploration is more effective. In conclusion the loss for the factored shared critic, consisting of the individual critics and the mixing network, is defined as:

$$L(\phi, \psi, B) = \Big((Q^{\text{tot}}_{\phi,\psi}(s, \mathbf{o}, \mathbf{a}) - r + \gamma(1-t)Q^{\text{tot}}_{\phi_{\text{target}},\psi_{\text{target}}}(s', \mathbf{o'}, \boldsymbol{\mu}_{\boldsymbol{\theta}_{\text{target}}}(\mathbf{o'}))\Big)^2 . \tag{2.5}$$

In the equation above, the observations o_i of all agents are concatenated to $\mathbf{o}$ and the taken actions are concatenated to the joint action $\mathbf{a}$. Similarly, the individual target networks of the agents are concatenated to $\boldsymbol{\mu}_{\boldsymbol{\theta}_{\text{target}}}$. The centralized critic Q^{tot} is described as the non-linear mixing function

$$Q^{\text{tot}}(s, \mathbf{o}, \mathbf{a}) = g_\psi\left(s, \{Q_i(o_i, a_i)\}_{i=1}^n\right) \tag{2.6}$$

where $Q_i(o_i, a_i)$ represents the individual critic supplied with the observations o_i and the action a_i of agent i.

The update of the actors is performed analogously to DDPG. The loss of the negated total q-value is backpropagated back through the individual critics to the actors.

3. Methodology

The methodology is separated into two main components: the simulation of the water distribution networks and the reinforcement learning framework for training the agents.

3.1. Simulation

In order to simulate our water distribution networks, we use a Functional Mock-up Unit (FMU) modeled in Modelica [12]. The Functional Mock-up Interface (FMI) [13] is a free standard developed by the Modelica Association to exchange and co-simulate dynamic simulation models. It aims for simple creation, storage and usage of such models. FMUs are built to support the FMI standard. Each FMU is a zipped archive containing XML files, binaries and C code to accurately describe the model, including the model's variables, parameters, essential metadata and their implementation.

To integrate the FMU with our reinforcement learning environment, we use SOFIRpy[14]. SOFIRpy is a Python package that supports the export of Dymola and OpenModelica[15] models as FMUs, the co-simulation of these FMUs with custom Python models, and the management of simulation results.

However, in our use case, the simulation must be extended to allow communication between the Python simulation models and the RL agents. SOFIRpy is designed such that the Python models, inheriting from the base class `SimulationEntity`, contain all necessary code to control the inputs and outputs of the FMU. These models are executed in each simulation step, setting the FMU's inputs according to their internal state, their logic and the FMU's output. In order to integrate the RL agents, this structure needs to be modified. Specifically, the handling of inputs and outputs needs to be decoupled from the decision-making process to allow the agents to make decisions independently.

We implement a custom class, `SimulationEntityWithAction`, which extends the base `SimulationEntity` to include an additional method for receiving external actions. This allows the RL agents to handle the decision-making process, while the `SimulationEntityWithAction` serves as an interface that passes the selected actions to the simulation.

To support this setup, the control over the individual time steps of the simulation must be shifted as well. Rather than running a complete episode, we simulate only the consequences of an action for a specific time frame. Therefore, we break down the simulation loop into individual steps, which can be executed externally by the MARL environment.

We also separate the code for both the creation and initialization of the FMU from the simulation itself. That way the FMU can be created at the beginning of a training and then be stored in a `ManualStepSimulator` object. As a result, unzipping the FMU does not affect performance, as it is just done once per training.

Separating the initialization also enables fast and efficient resets of the FMU and thereby the simulation, at any point during the training.

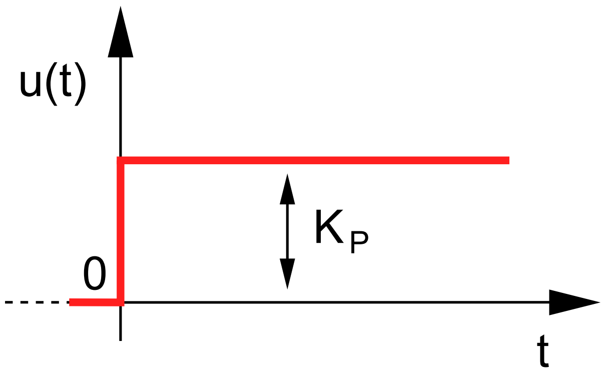
In addition, we implement an interface to read out the simulation data at any time step, enabling the creation of states and observations for the agents.

With the simulation interface in place, we can now define the main components of the water network: pumps, tanks, valves and consumers. Pumps serve as the network's source. They pump water into the network or increase its pressure level, depending on their set pump speed, which is set relative to the maximum power as a percentage. Valves can be opened and closed analogously using a percentage relatively to the maximum opening. The tanks cannot be controlled themselves in any way, but their valve can be controlled just like any other valve. That way, the tank can be opened to either let water out or in, depending on the pressure in the network. Alternatively, it can be closed to conserve water for a later use. The consumers are the sinks in the network. They are modeled as such. To control how much water the consumers consume, they are fitted with a valve as well. Though, in this case the valve cannot be controlled by the agents.

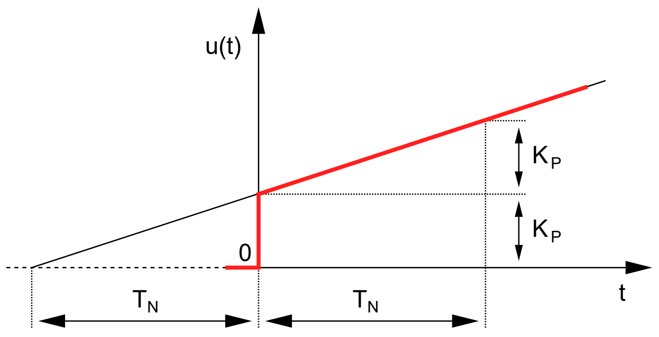
Instead, the valve is controlled by a PI-Controller. The PI-Controller receives a certain volume flow as a demand for the consumer and adjusts it's opening accordingly. It does so by calculating the error between the desired flow and the actual measured flow, and then applying a correction based on both the magnitude of the error (proportional term) and the accumulated past errors over time (integral term). To be exact, the resulting output is calculated following the equation

$$u(t) = K_p e(t) + K_i \int e(t)dt, \tag{3.1}$$

where $e(t)$ denotes the error function and K_p and K_i denote constants for tuning the individual terms. A simple proportional compensation of the error can lead to steady-state errors, where the controllers response levels out with a small, persistent error that the controller cannot eliminate. For that reason, an integral term is included to consider the error over time. If the system remains in a state with error, the integral term accumulates it, thereby increasing the correction consistently until the error is corrected. Figure 3.1 shows the difference in response to a step in the error function, showing the additional correction of the integral term.



(a) Step-response of a P-controller, reprinted from [16]



(b) Step-response of a PI-controller, reprinted from [17]

Figure 3.1.: The step response, i.e. the output generated for the error function being the step-function, for a P-controller with only the proportional term, and a PI-Controller with an additional integral term.

3.2. MushroomRL

For reinforcement learning, we use the Python library MushroomRL[18]. MushroomRL provides an interface to run a variety of RL algorithms in traditional RL and deep RL experiments. However, MushroomRL is designed primarily for single-agent reinforcement learning. Therefore, we adapted the multi agent extension of MushroomRL from [19].

3.2.1. MAMDPInfo

To create environments for multiple agents, the default `MDPInfo` is extended. While basic information, such as the discount factor and the horizon, can be reused, the new `MAMDPInfo` object requires support for partial observability and distinct action spaces for different agents. A state space attribute is introduced to describe the global state space, which is used in fully observable systems or in partially observable systems with global information during training, as in a MADDPG setting. For partially observable systems the observation spaces attribute is added, replacing the general observation space and specifying the observation space for each agent individually. Finally, the action spaces attribute is created following the same principle, as a list of action spaces for each agent.

3.2.2. MAEnvironment

The base class `MAEnvironment` is functionally identical to the base class `Environment` that comes with MushroomRL, with the exception for its info. This class serves solely to distinguish between environments that support multi-agent systems, containing a `MAMDPInfo` object, and environments that do not.

3.2.3. Abstract Fluid Environment

With the environments supporting multi-agent systems, the RL code can be integrated with the simulation. We create the class `AbstractFluidEnvironment`. While implementing the `MAEnvironment`, this class contains a ManualStepSimulator instance. That way, any interaction with the environment can be translated into the corresponding actions within the simulation.

When the environment is reset, the reset of the simulation is also triggered. Afterward, the initial state of the simulator is read out and is used to define the state and the observations of the agents.

The `step` method is left without implementation in the abstract class. Nevertheless, the general functionality remains consistent throughout all fluid environments. The actions supplied by the agents are typically clipped. Then, the simulation step method is executed with the provided actions. Since pumps are generally controlled on a larger interval than a single second and the simulation time steps can be quite small, multiple simulation steps are executed in one single step of the `AbstractFluidEnvironment`. The number of simulation steps run by the simulator with the step size δt between each step call on the `MAEnvironment` is controlled by the `CONTROL_STEP_INTERVAL` constant, which varies between different environments.

This abstract environment is then overridden for each WDN presented in the next chapter.

3.2.4. Neural Networks

In all our experiments, we exclusively use a consistent Neural Networks architecture consisting exclusively of linear layers. The input shape for the first layer is specified using the `MAMDPInfo` object. Then, a hidden layer follows with its size determined by the hyper-parameter set used for the training. Finally, the hidden layer is connected to the output layer to generate the network's output.

For an actor network, the output shape is also defined in the `MAMDPInfo` object. In the case of a critic network, the output is always the q-value.

The activations for the linear layers are set to RELU, while the output layer uses a linear activation for the q-values. For an actor network, the output layer uses a specified activation for the actions. By default, this activation function is the sigmoid function, but it also supports $\tanh$ and linear activations.

3.2.5. Training and Hyperparameters

Each training run follows the same general procedure. Regardless of the choice of algorithm or the specific WDN used, we begin by setting up the agents and the environment. Afterwards, the warmup phase is started, during which the replay buffer is filled with experiences, without any learning or fitting steps. Once the `initial_replay_size` is reached, training begins. For `n_epochs` the following steps are repeated:

1. Learning step: The agents complete `n_episodes_train` episodes, and after each step in the environment, the weights are updated with a `fit` call.
2. Evaluation step: The agents complete `n_episodes_eval` episodes and the results are evaluated. During this evaluation, the policy noise is paused to only consider the actions of the agents. The scores of the episodes are evaluated and logged, along with any debug information.
3. Epoch-End: This step marks the end of an epoch. The policy noise is resumed and updated if a decay was specified in the parameters. Additionally the weights are saved, if the epoch count matches the checkpoint interval `n_epochs_per_checkpoint`.

After the training is completed, a final evaluation is performed on a much larger scale, with a significantly higher number of episodes, with the results being saved in a json file. Like the periodic evaluations during training, this evaluation uses the same environment as the training. Finally, the agent's final weights are stored as well.

For a complete explanation on each of the parameters that can be supplied to the training, please refer to appendix A. For the complete code base please refer to `https://github.com/Crypter44/WateRL`.

4. Network Configuration

In our experiments we consider two models of water distribution networks, with tasks specifically designed around them. The following sections explain each of them individually.

4.1. Circular Network

The topology of our circular network consists of two pumps and four consumers. The pumps p_1 and p_2 are located on the top and bottom of the network, each pumping towards the middle. The 4 consumers c_2, c_3, c_5 and c_6 are arranged in a circle or rather a square. Each consumer has a certain demand of water, that needs to be fulfilled. The demands are considered over a time frame of less than a minute. Due to the scale of a WDN, we assume this demand to be constant over this short period of time.

In this network the IDDPG agents control the two pumps, setting their rotational speed, thereby affecting the amount of water available to the consumers. Their task is simple: supply enough water that each individual demand is settled. At the same time, they are supposed to be as energy-efficient as possible, aiming for the lowest power consumption during their operation, without violating the demand.

An agent is supplied with each of the four demands from the consumers, therefore creating a fully observable setting. Additionally, they are passed the actual flow rates from the consumers, allowing them to analyze the difference in supply and demand. The 8-dimensional observation space is then mapped to a single action for each of the agents, supplying the network with two individually chosen pump speeds.

4.1.1. Demand Generation

To train the agents on this network, appropriate training data is required. Since the focus of this task is on efficiency, we aim to generate training load cases with varying demand patterns to evaluate the agents under different conditions. As each consumer represents a group of households, we assume their minimum and maximum demands to be identical across all consumers. Additionally, we assume that there is always a certain level of demand present at all times. We therefore define the range of possible demands as $[0.4, 1.4]$.

The naive approach to generating training data would be to sample the demand of each consumer from a distribution, such as a uniform distribution. However, this method has a significant drawback. The relevant quantity for the pumps is the total demand in the network, not the individual demands. When sampling individual demands from a uniform distribution, the distribution of their sum converges towards a Gaussian, according to the central limit theorem. As a result, load cases with particularly high or low total demand become increasingly unlikely, while scenarios with average total demand dominate the training data.

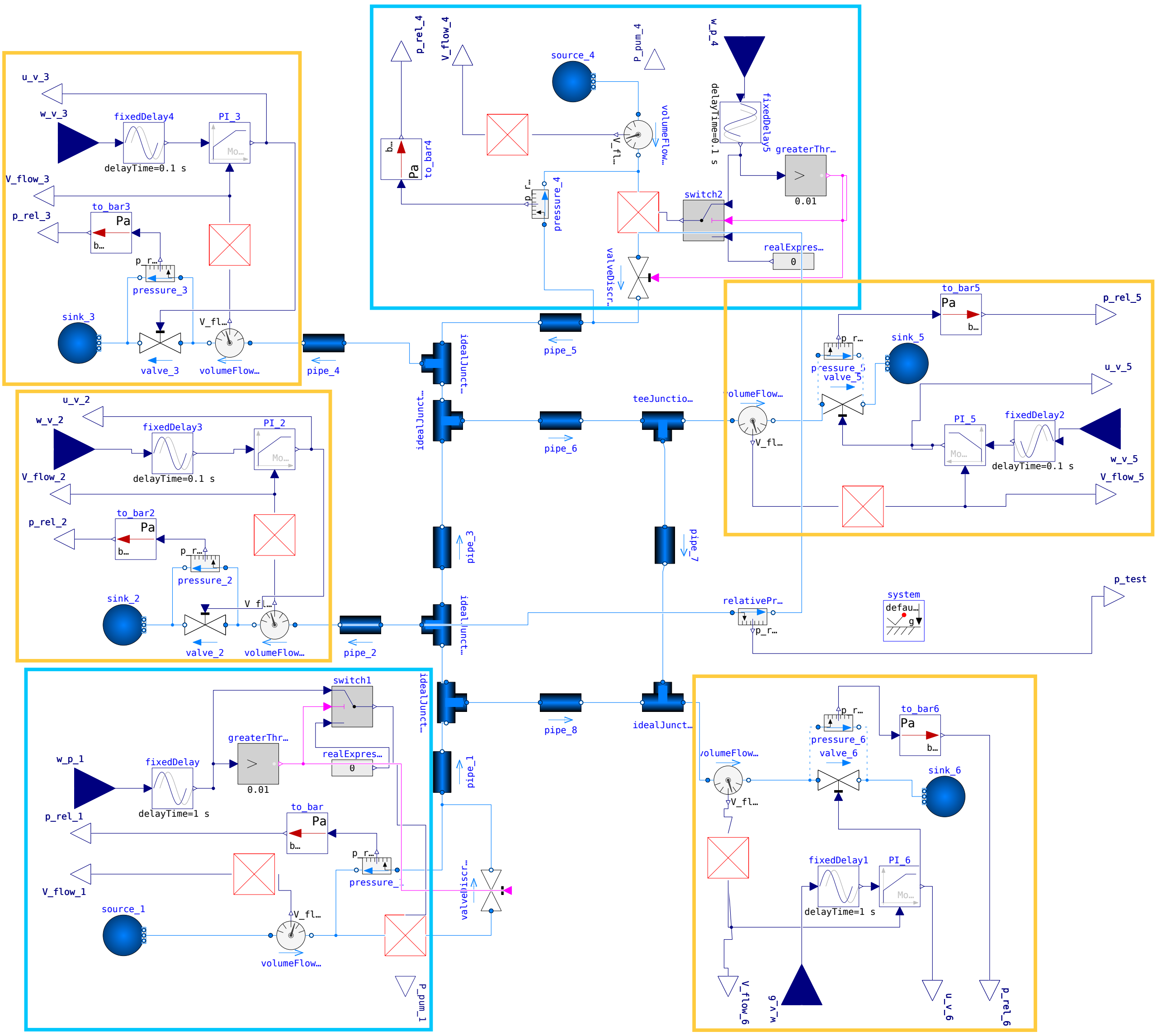


Figure 4.1.: The topology of the circular network, with the pumps (blue) located in the lower left and in the middle at the top and with the four consumers (yellow) splitting of from the circular network at the T-Junctions.

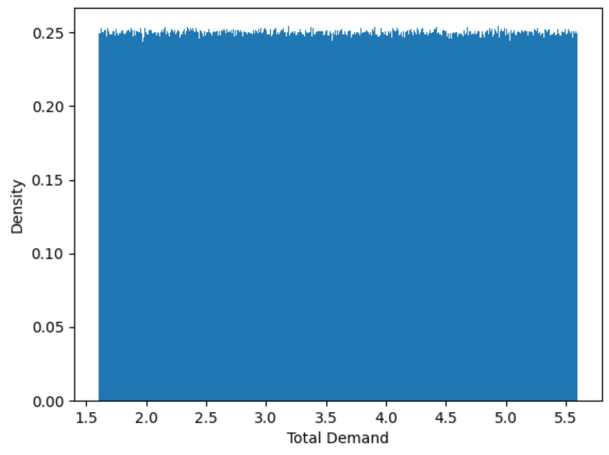


Figure 4.2.: Histogram of the total demand for 10 million load cases generated with our demand generation.

It is not possible for both the individual samples and their sum to follow a uniform distribution simultaneously. Therefore, we choose to distribute the total demand uniformly instead, as this is the more relevant quantity for our application, as explained above. This leads to the challenge of generating individual demands that sum to the sampled total demand.

To address this, we use a Dirichlet distribution [20] to sample the relative fractions of the total volume flow for each consumer. However, while the Dirichlet distribution ensures that the sampled fractions sum to one, it does not account for the additional constraint that each individual demand must lie within a specified range. So, instead of multiplying the fractions by the total sum, we multiply them by the remainder of the sum after subtracting the minimum volume flow for each consumer. This ensures that the resulting individual volume flows meet the minimum requirements. To handle the maximum, we apply excess shifting. Any additional volume flow exceeding the maximum for a consumer is removed and equally redistributed to the remaining consumers. This process is repeated until no consumer exceeds the maximum, or until the excess has been shifted up to ten times, to prevent infinite loops.

The resulting demand distribution clearly follows a uniform distribution for the total demand, as shown in Figure 4.2. As expected, the individual demands cannot be uniformly distributed simultaneously (Fig. 4.3). Instead, the distribution shows a distinct tendency towards the maximum volume flow. This is intuitive, as the excess shifting process reduces any demand exceeding the maximum back to the maximum, with the excess being redistributed to the other consumers.

The resulting distribution has one additional feature that we want to explicitly mention, as it will be relevant in an upcoming experiment. The density of the standard deviation across the load cases, generated as described above, is shown in Figure 4.4. It is evident that in our training data, the individual demands tend to remain relatively close to one another, avoiding the most extreme differences between individual consumers.

4.1.2. Visualization

We visualize the behavior of the agents in the water network, as well as most of the data supplied by the simulation, with a series of plots (Fig 4.5). The 5 plots in the upper half visualize the consumers and their

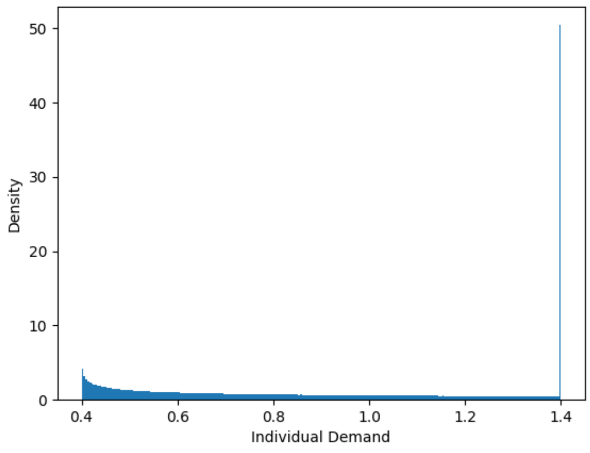


Figure 4.3.: Histogram of the individual demand for 10 million load cases generated with our demand generation.

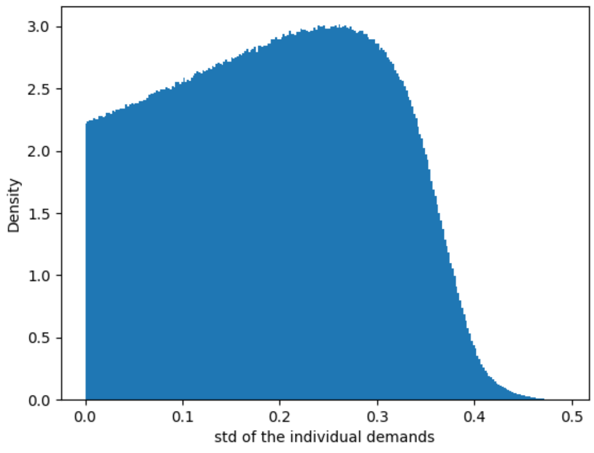


Figure 4.4.: Histogram for 10 million load cases, showing the standard deviation of our individual demands per load case.

Table 4.1.: Set of symbols for the data provided by the circular network

Symbol	Description
f_i	Volume flow at the component with index i
d_i	Demand of the consumer with index i
o_i	Opening of the valve with index i
s_i	Rotational speed of the pump with index i
P_i	Power consumption of the pump with index i
p_i	Pressure measured at the pump with index i

demands. Each consumer's demand is shown as a dotted line, to visualize, what volume flow they require. The actual volume flow arriving at or flowing through the valve is plotted additionally as a solid line of the same color. In the best case these plots would overlap perfectly. Finally each consumer's plot contains the opening of it's valve on a separate y-axis as well. The bottom plots show the behavior of the pumps which are controlled by the agents. For each of the two pumps the rotational speed, i.e. the action of the agent, is displayed. Since the power consumption is closely related to the speed, we include it in the same plot on a separate y-axis. Finally, we visualize the volume flow provided for each pump.

Some plots will additionally contain two summary plots in the right column. One showing the total demand split into the 4 individual demands and the other illustrating the total supplied volume flow split by the pump providing it.

The complete set of data provided by the circular network and their symbols can be seen in Table 4.1

4.2. Tank Network

The tank network has fewer components. It consists of just one pump and one consumer. Additionally, the network contains a tank. Both the pump and the valve of the tank are controlled by agents. The network topology is quite simple. The pump transfers water into the system and straight towards the consumer. The connecting pipe continues towards the tank, which is elevated at a height h. In contrast to the circular network, this network is simulated over the course of an entire day.

The task of this network focuses on more advanced temporal planning. The demand is chosen to resemble a typical daily demand in a network, where the growing number of consumers has reached the maximum capacity of the pumps, leading to water shortages in peak hours. These peak hours are situated in the morning and in the evening. A solution for shortages aside from upgrading or replacing the pumps, can be a tank to conserve water in off-peak hours. This network simulates such a scenario. Both demand peaks exceed the maximum capacity of the pump used in this network. This requires the pumps to provide more pressure during off-peak hours to pump water into the tank, which can then be used in the peak hours to help meet the demand, when the pumps cannot supply enough on their own.

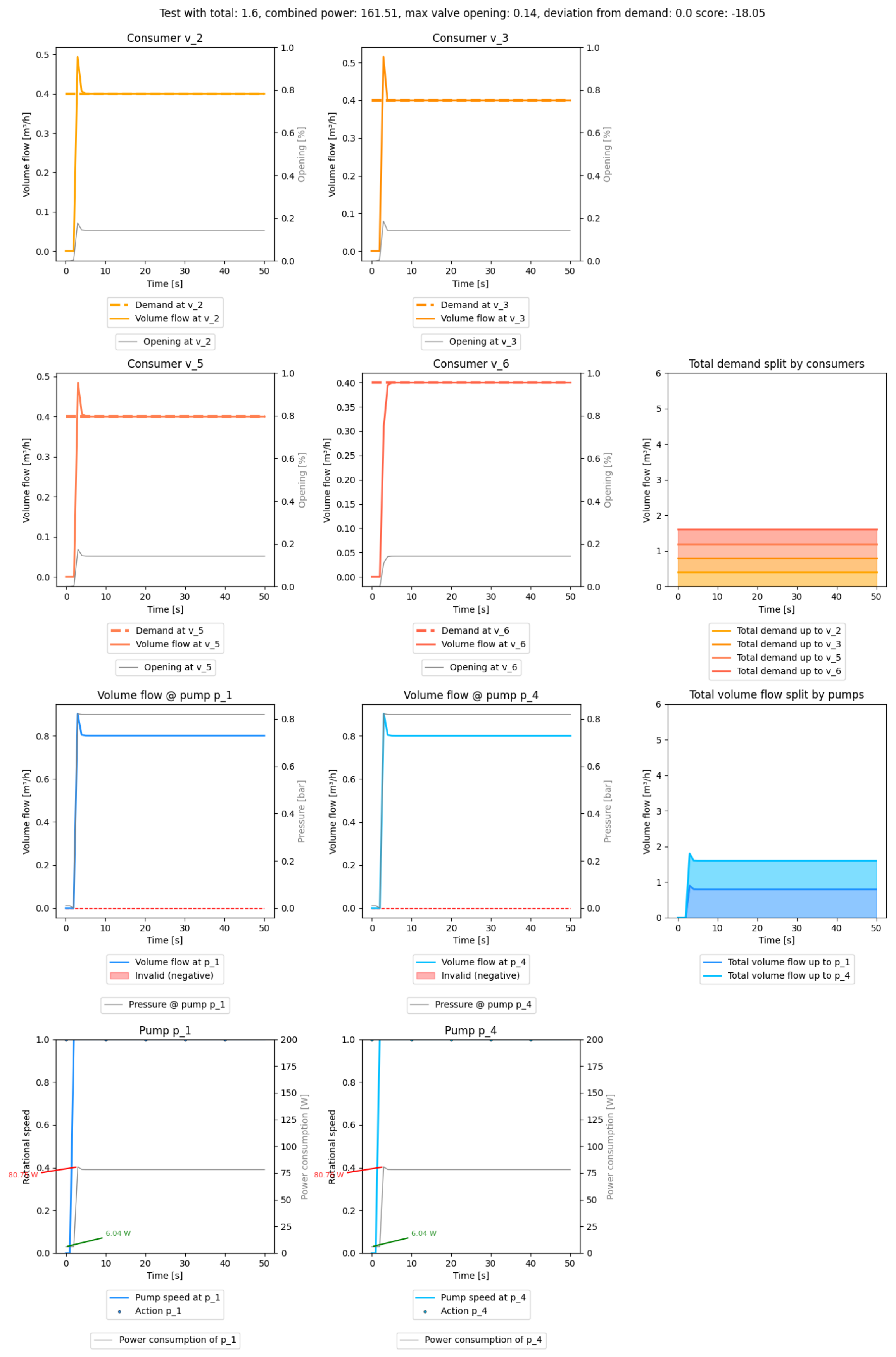


Figure 4.5.: An exemplary plot for the circular network.

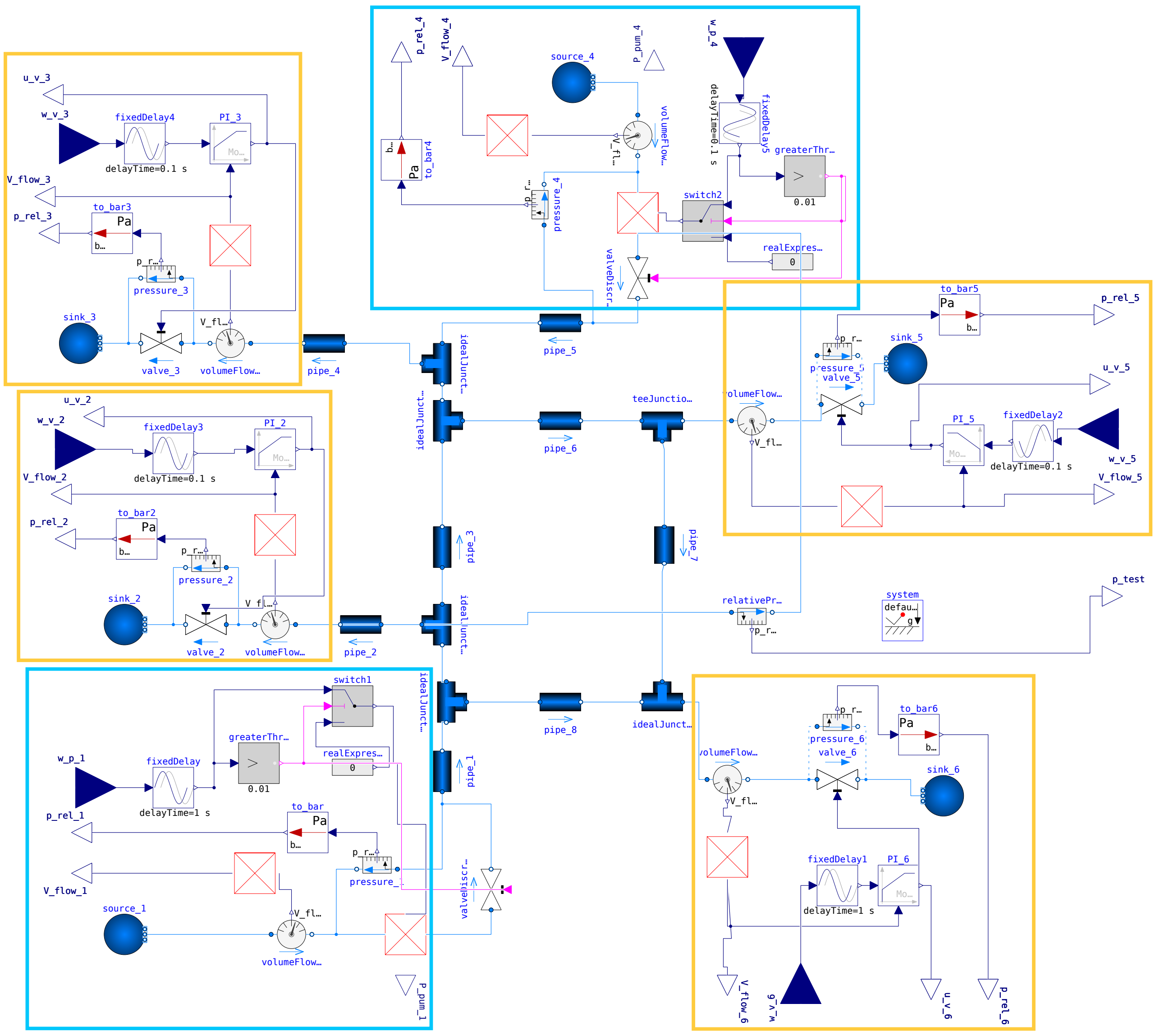


Figure 4.6.: The topology of the tank network.

4.2.1. Visualization

Again, the behavior of the agents is visualized with a series of plots (Fig- 4.7). The plot for the consumer is equivalent to the one in the circular network. The plot displays the demand, the actual volume flow and the opening of the corresponding valve. The plots for the pump are also equivalent: one plot for the pump's rotational speed and power consumption, one for the volume flow it is providing. The additional plots show information about the tank. The bottom left diagram displays the level of the water in the tank in m, to show how much water is left in the tank at any time during the day. Also, the flow through the valve of the tank is presented in the middle plot of the second row. Positive flow represents a volume flow into the tank, while negative flow represents an outflow. Finally, the last plot contains the valve opening of the tank, and thereby the action of the agent controlling the tank.

Figure 4.7.: An examplary plot from the tank network

5. Experiments

In the following we present the experiments performed and their results.

5.1. Reward Metrics for Enhancing Agent Efficiency

In the first group of experiments we investigate, which reward signals can be used to train agents that are efficient in their power consumption, while still ensuring the satisfaction of the demand. In the following, we test and evaluate different signals individually and finally combine them into one reward function that can be used to train agents effectively.

5.1.1. Demand Deviation as a Core Metric

The most important aspect of any strategy in the WDNs is the functionality, meaning the ability to provide enough water to fulfill the demand. To train the agents to do that,we define an exponential reward term r_{demand} on the deviation from the demand d. To make the combination of different reward functions simple and the individual reward signals more modular, we add the subsequently described parameters. For the reward signal we introduce a weight w_{demand}. To map the exponential from the range of $[0, 1]$ linearly to an arbitrary different range, we use a minimum l and a maximum value h. The shape of the exponential is parameterized with a smoothness factor s for the peak of the reward as well as a boundary point $p = (p_x, p_y)$, which specifies a reward value before the mapping takes place for a specific deviation. Using this point, that the exponential function is defined on, one can intuitively change the wideness of the curve, making it narrower or wider, depending on the intention.

The resulting reward signal is defined as follows:

$$r_{\text{demand}}(\delta_{\max}) = w_{\text{demand}} \left((h - l)\, a \exp\left(-b\sqrt{\delta_{\max}^2 + s} \right) + l \right) \tag{5.1}$$

with a, b defined by the boundary p_x, the value of the reward at the boundary p_y and the smoothness s:

$$a = \exp\left(b\sqrt{s} \right) \tag{5.2}$$

$$b = \frac{\ln(p_y)}{\sqrt{s} - \sqrt{s + p_x^2}} \tag{5.3}$$

and with the deviation defined as the maximum deviation between the actual flow and the demand across all consumers, ensuring that a deviation from the demand cannot be compensated by averaging effects:

$$\delta_{\text{max}} = \max\left(|f_2 - d_2|, |f_3 - d_3|, |f_5 - d_5|, |f_6 - d_6|\right). \tag{5.4}$$

Since this reward only assesses the deviation, the agents are unlikely to learn an efficient strategy, but a purely functional one instead. The results from a training match this expectation. The agents use a high rotational speed, even in scenarios with low load across all consumers, assuring that the demand is met in every case tested. It can be safely concluded, that the exponential reward signal works well to guide the agents towards functional strategies.

5.1.2. Raw Power Consumption as an Efficiency Metric

While meeting the demand is crucial, the agents are supposed to consume only as little power P_i as necessary. Hence, we introduce the following reward signal, weighted by $w_{\text{raw-power}}$

$$r_{\text{raw-power}}(P_1, P_4) = -w_{\text{raw-power}}\left(P_1 + P_4\right), \tag{5.5}$$

which can be combined into a sum with the exponential reward term for the demand (Eq. 5.1):

$$r = w_{\text{demand}} r_{\text{demand}} + w_{\text{raw}-\text{power}} r_{\text{raw}-\text{power}}. \tag{5.6}$$

The weights are set to $w_{\text{demand}} = 10$ and $w_{\text{raw}-\text{power}} = 0.1$ for the following subsections.

Power Consumption Issues in the Simulation Environment

However, the very first training reveals several issues (Fig. 5.1). Instead of having the intended effect, that the pump speeds are reduced and the demand is met, the results show deviations from the demand in numerous load cases. The pumps are turning off, even when this causes the consumers to deviate from their demand. This would normally be easily fixed by tuning the weights of the individual signals, as the importance of the demand could be scaled up that way. Unfortunately, the problem is caused by the simulation itself, which can be confirmed by running specific test cases. The pumps turn off, their power consumption drops to zero and they stop providing water, even when the pumps' agents are supplying non-zero actions.

Another case shows a power consumption of less than $6\,\text{W}$ for the pumps, which according to their specification is below their minimum idle power consumption.

Further investigation of the issues establishes a pattern. The simulation of the pumps fails after spikes in the actions of the agents. A direct quick transition from a high rotational speed to a low rotational speed and back to the high one, breaks the pump for an unspecified amount of time. This is exploited by the agents utilizing a strategy of oscillating quickly between maximum and minimum power to break the pumps' simulation. Thereby, saving power, whenever they deemed the power consumption more important than the deviation from the demand. Another noteworthy observation is a correlation between the violation of the minimum idle power consumption and backwards flow through the pump. In a regular network this is prohibited by one-way valves, contrary to our simulation, where no similar restrictions apply.

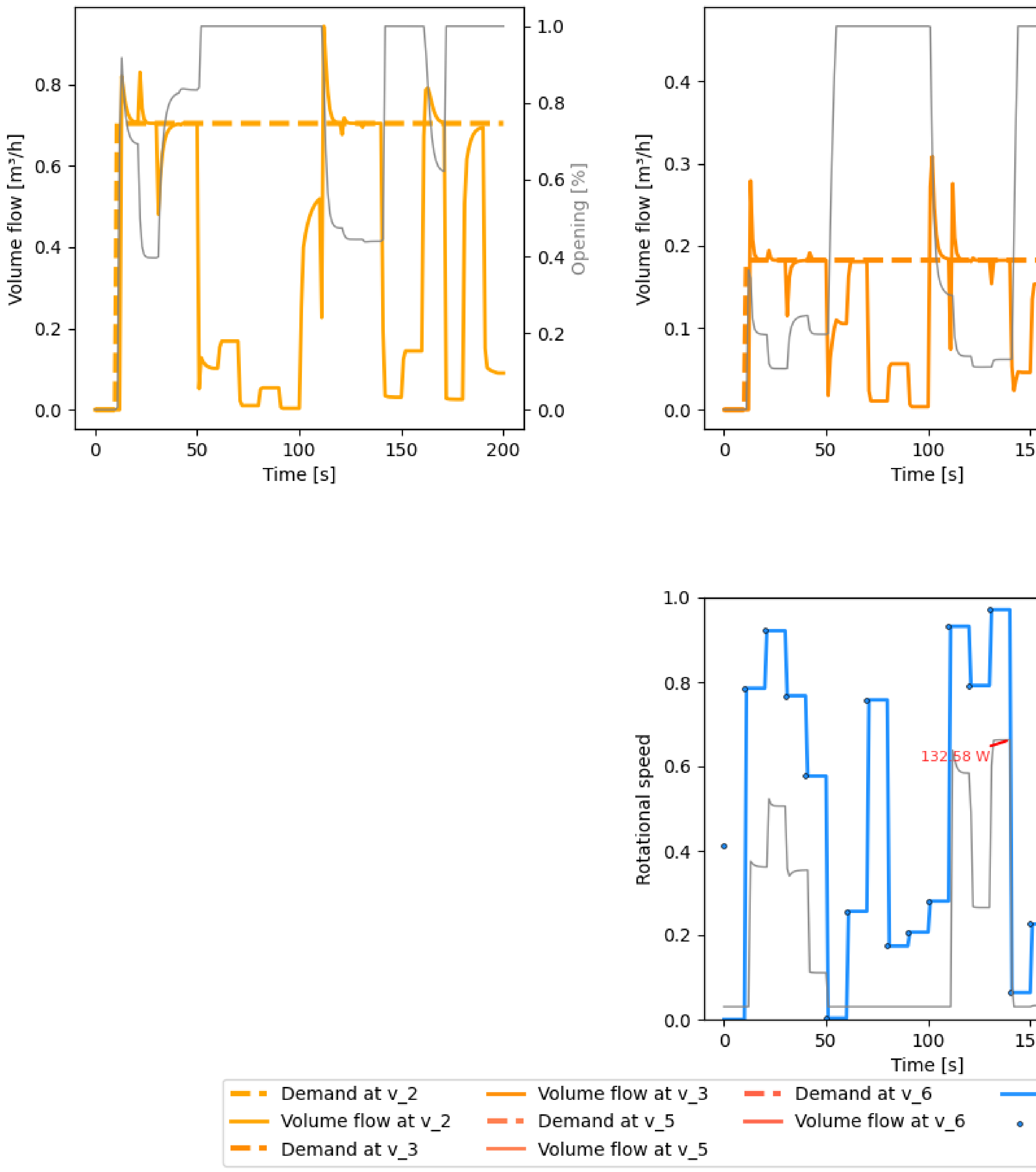


Figure 5.1.: Plot of an episode during early training, showing issues with the simulation from time-step 50 to 110.

Fixing Oscillations

The observations described above show, that the agents need to be prohibited from choosing oscillating actions for the pump's rotational speed until the issues in the simulation are fixed. By just supplying the demand as the state for the agents as opposed to the additional flow rates (compare Chap. 4.1) the task becomes static and turns essentially into a single state problem, since the demand is assumed constant in this network. This forces the agents to learn one action entirely specified by the demand, making oscillations and thereby broken pumps impossible. As a constant demand typically warrants a static solution, this simplification is both effective and reasonable.

Reducing the Influence of Raw Power Consumption

Another takeaway from the training is to reduce the weight of the power consumption to avoid deviations from the demand. The weight of the power penalty is decreased to $w_{\text{power}} = 0.05$, while keeping the w_{demand} at 10. In addition, a more narrow the demand signal is chosen by moving the boundary point further in, specifying a reward of only 0.001 at a deviation from the demand by 0.1. With that the agents' accuracy, regarding the demand-fulfillment should improve, since even small deviations are punished heavily.

With these insights implemented the results show the first indication of improved efficiency. In scenarios with medium to high load the pumps run at high speeds and the demand is met. This behavior can be observed in the high-load scenario, where the pumps operate at full power (Fig. 5.2). In scenarios with low load, the pump speeds are reduced (Fig. 5.3). Still, the pressure is very high in the system. This can also be seen in the openings of the valve, as they are only opened half way in most medium to low load cases, indicating that the pressure is so high, that the desired flow is already achieved with that opening. In a system where the pressure is tuned perfectly, at least one valve opening must be close to fully open, meaning that a further reduction of the pumps' speeds would be beneficial.

Analyzing the maximum load cases reveals further inefficiencies. Here, the agents choose to make use of a trade-off between reducing the power consumption and and meeting the demand. This results in the behavior of turning off one pump completely to conserve energy independent of the demand or load in the system, and one pump running at full speed accepting the heavy penalty for not completely fulfilling the demand (Fig. 5.4). This poses a conundrum for the reward defined in this chapter. In order to suppress the afore mentioned trade-off one might lower the weight of the power penalty, while at the same time the pumps are still by far not as efficient as they could be. But the latter suggests to adopt a higher weight, to give efficiency more importance in lower load cases.

In addition, the manner in which the pump speeds are reduced is suboptimal. The individual agents show two distinct strategies. One pump runs at full speed, independent of the demand or load in the system, while the other lowers its speed to conserve energy. This large imbalance of speeds is typically inefficient, as we create an uneven supply in the symmetric system. Furthermore, it introduces large amounts of backwards flow through the pump running at a lower speed, affecting the power calculation in the simulation.

To counteract the backwards flow, we include a simple penalty in the reward term, penalizing any negative flow at the pumps beneath a threshold with a constant value. This is more robust than making changes in the simulation, to model one-way valves. We define the negative flow reward signal as

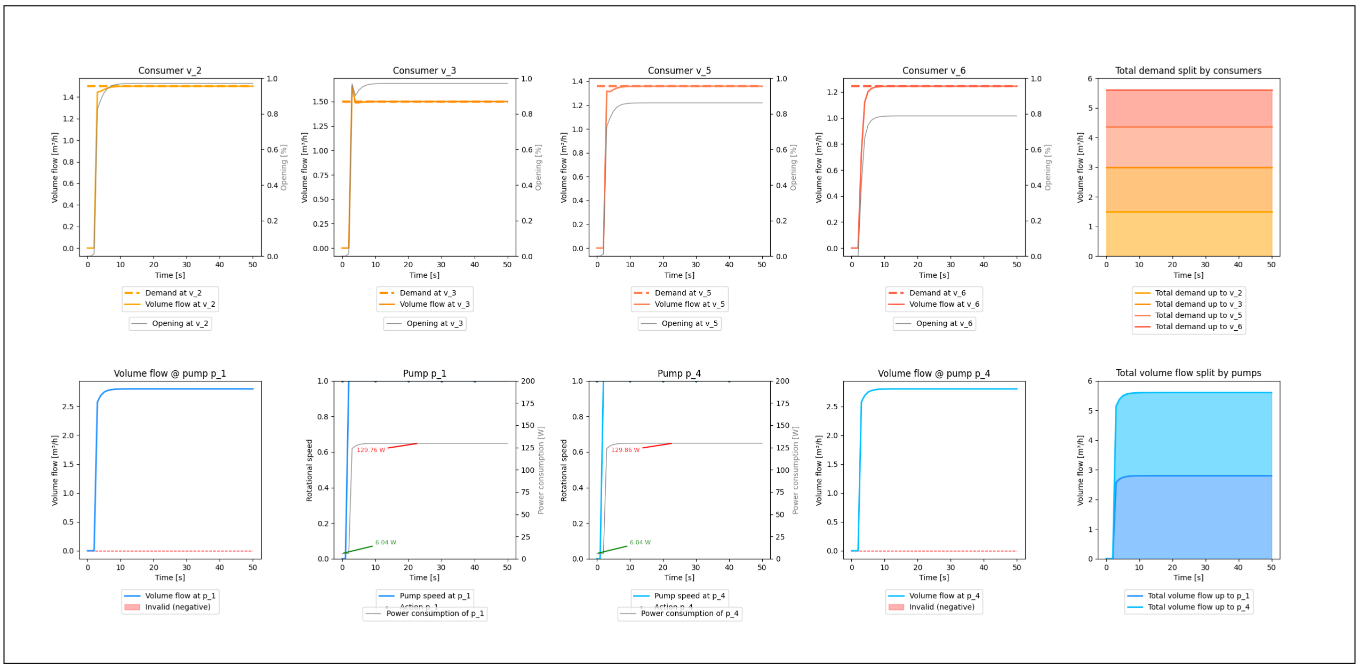


Figure 5.2.: Scenario with high load, showing pumps running at full power.

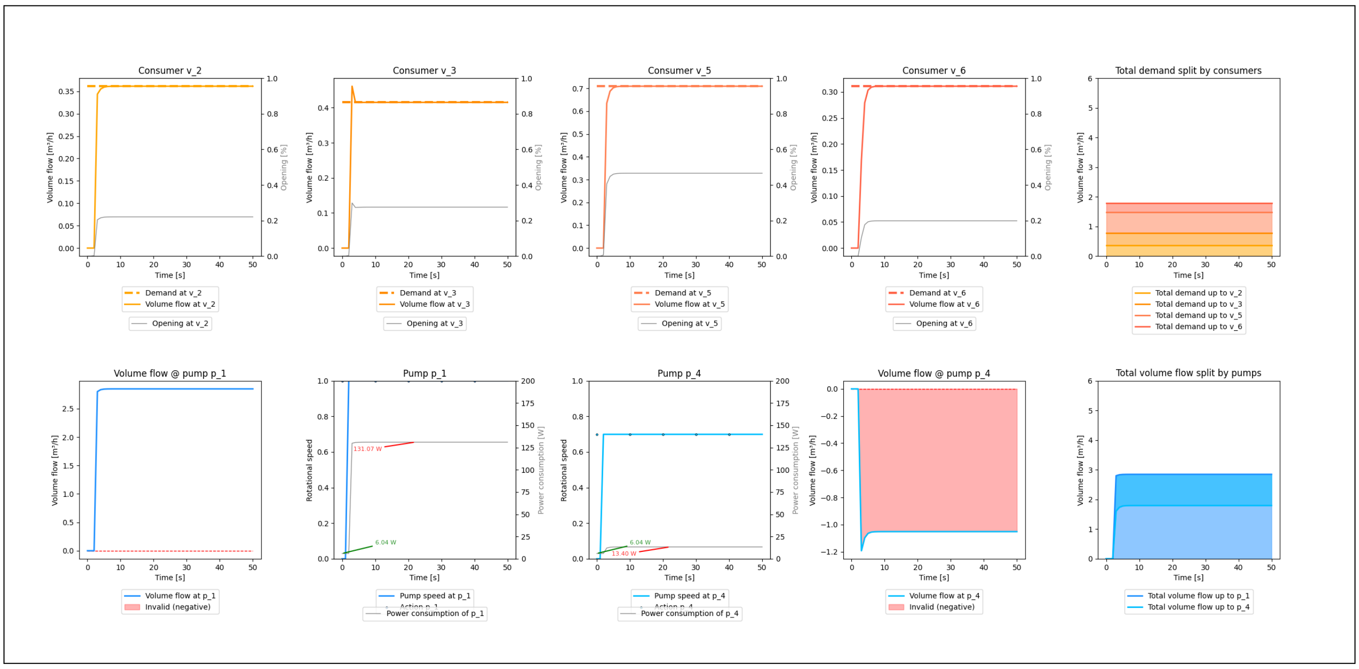


Figure 5.3.: Scenario with very low load, showing reduced pump speeds in one pump leading to backwards flow. While saving power through the reduction, the pressure in the system is high, leading to low valve openings.

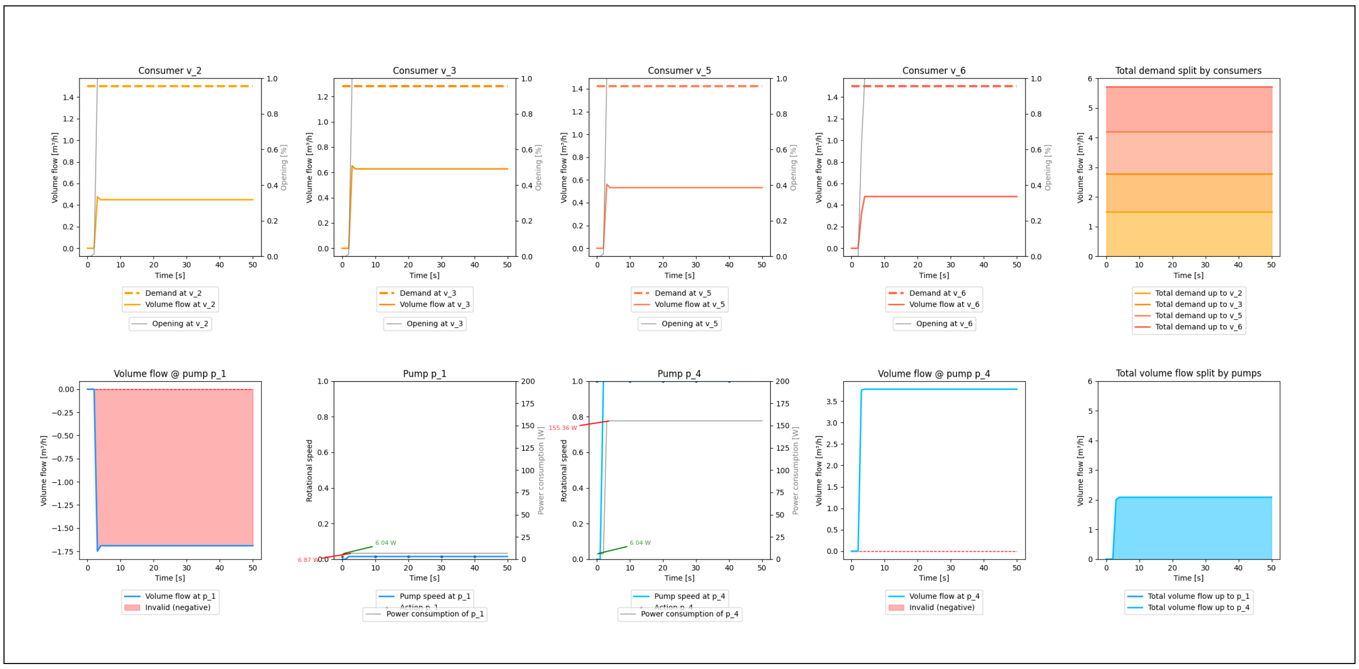


Figure 5.4.: Scenario with maximum load, showing significant deviations from the demand, as well as negative flow caused by the imbalance of the pumps' speeds.

$$r_{\text{neg-flow}}(f_1, f_4) = \begin{cases} -w_{\text{neg-flow}}, & f_1 < t \vee f_4 < t \\ 0, & \text{else} \end{cases} \tag{5.7}$$

with t being the threshold at which we start penalizing the flow. As we are using a numerical simulation and we want to make sure, that a pump speed of 0 will always be allowed, we accept very low values, both negative and positive, around zero and set the threshold to $t = -1 \times 10^{-6}$ per default.

5.1.3. Improving the Trade-Off Between Efficiency and Demand-Fulfillment

Given that all reward signals must be combined into a single metric, trade-offs will be inevitable between two conflicting signals, such that optimizing one leads to a decrease in the other. Such a trade-off exists between the power consumption and the deviation from the demand, as we have seen previously. This trade-off can be mitigated by adjusting the weights of the signals, effectively shifting the threshold to ensure that meeting the volume flow requirement remains the dominant objective, making it unlikely — if not impossible — for power minimization to take precedence. This assigns one of the two signals with much lower importance. Since we need the agents to show more efficient behavior, lowering the importance of the power penalty is not an option. For this reason, we needed a different approach to improve efficiency as well as the robustness of the demand-fulfillment. Two approaches will be discussed in the next sections.

Power Consumption per Flow Rate

The previous experiments consider the raw power consumption only, disregarding the load of the network. A combined power consumption of $10\,\mathrm{W}$ gets a penalty of $-10w_{\text{raw-power}}$ in any scenario. Therefore, we move the reward towards a definition of efficiency. An efficient system is still allowed to consume large amounts of power, if a large majority of that power is used for its intended purpose and converted to useful output, i.e. providing a sufficient volume flow. Likewise, a system, that consumes less power in total but only converts a small fraction of its power into useful output, is still considered less efficient. Thus, we want to punish high power consumption especially in lower load cases, and allow the pumps to consume much more power in higher load cases, where this larger amount of power is of use. As a consequence we define a reward signal

$$r_{\text{pow-per-flow}}(f_1, f_4, P_1, P_4) = -w_{\text{pow-per-flow}} \frac{P_1 + P_4}{f_1 + f_4 + 1}, \tag{5.8}$$

that assesses the power consumption in relation to the volume flow provided in turn.

We run a grid search for the value of $w_{\text{pow-per-flow}}$. The weight for the demand-deviation signal is kept at $w_{\text{demand}} = 10$. The results show that even with the volume-flow-based scaling of power consumption in place, assigning a weight above $w_{\text{pow-per-flow}} = 0.2$ still leads to deviations from the demand under high-load conditions. Nevertheless, the trade-off is improved considerably. Lower weights such as $w_{\text{pow-per-flow}} = 0.1$ show power consumptions that are superior to any prior results. Under high load and even maximum load the pumps are kept at full speed, thereby meeting the demand at all times (Fig. 5.5). Simultaneously, the pumps' speeds are reduced under low load, under minimum load even as far as 60% (Fig. 5.6), which marks a tremendous improvement compared to a single pump, with its power reduced to 80%, as seen previously (Fig. 5.3). The effect of the negative flow penalty is also visible, as the pumps mostly coordinate their speeds to match, only deviating slightly from each other and never generating any backwards flow through the pumps.

Still, the results can be further improved. The absolute difference in power consumption, when considering an increase of the pump speed is based on the pump's power characteristic curve. In our case, increasing the speed when it is already high results in a substantially larger increase in power consumption than increasing it by the same amount at low speeds (Fig. 5.7). This implies that high load cases still receive a stronger reward signal towards efficient pump speeds compared to low load cases. Considering the minimal load case in Figure 5.6 again, it is evident that the openings are still not optimized fully. Even though no water is wasted on backwards flow through the pump, as seen in the training on the raw power consumption (Fig 5.3, the maximum opening only reaches slightly more than 50%. Apparently, the pressure in the system is still higher than necessary, though the effect on the raw power consumption is admittedly small.

Targeting Maximum Valve Openings

Following the findings from the previous chapter the idea of optimizing the valve openings can be investigated further. As the valves of the consumers are controlled by PI-Controllers, they have innate logic that makes the opening of the consumer a valuable source of information regarding the state of the network. The PI-Controllers are set up to simulate greedy consumers, which are only satisfied once they receive the volume flow that they require. The resulting behavior of the valves can be observed to gather information about both key criteria, the deviation from the demand and the efficiency in terms of wasted pressure in the system.

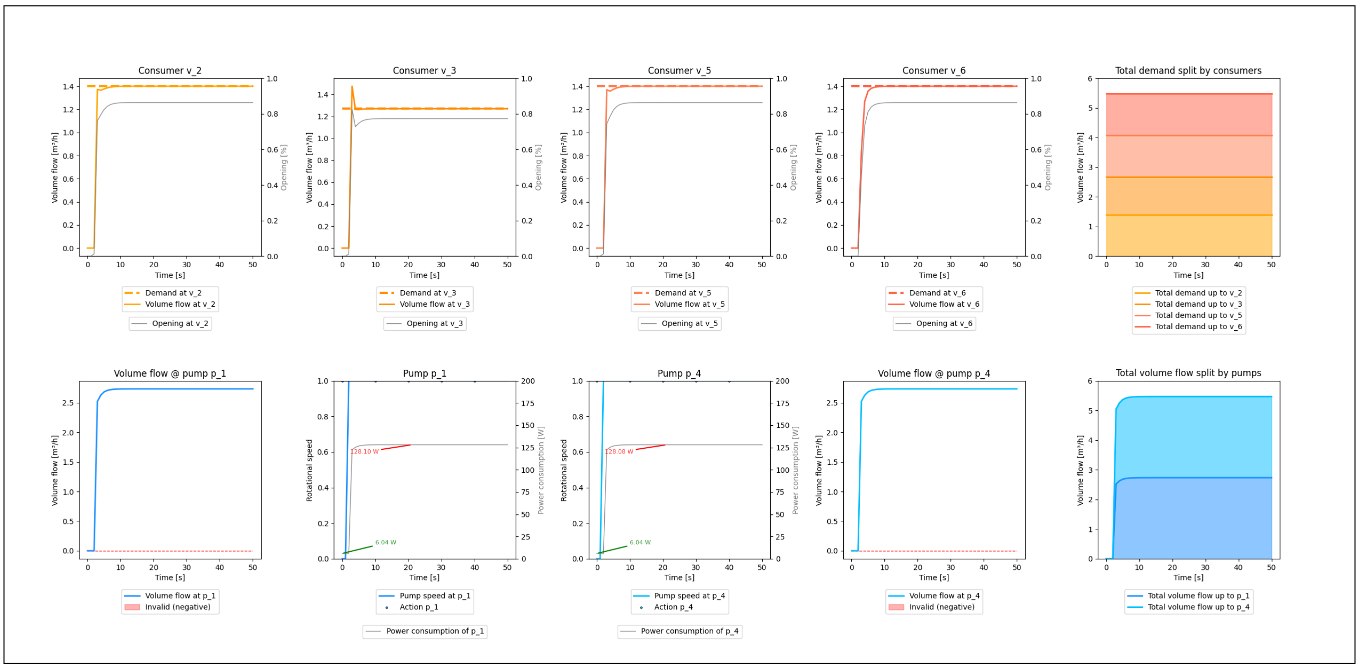


Figure 5.5.: Scenario with close to maximum load, showing high pump speeds and no deviation from the demand.

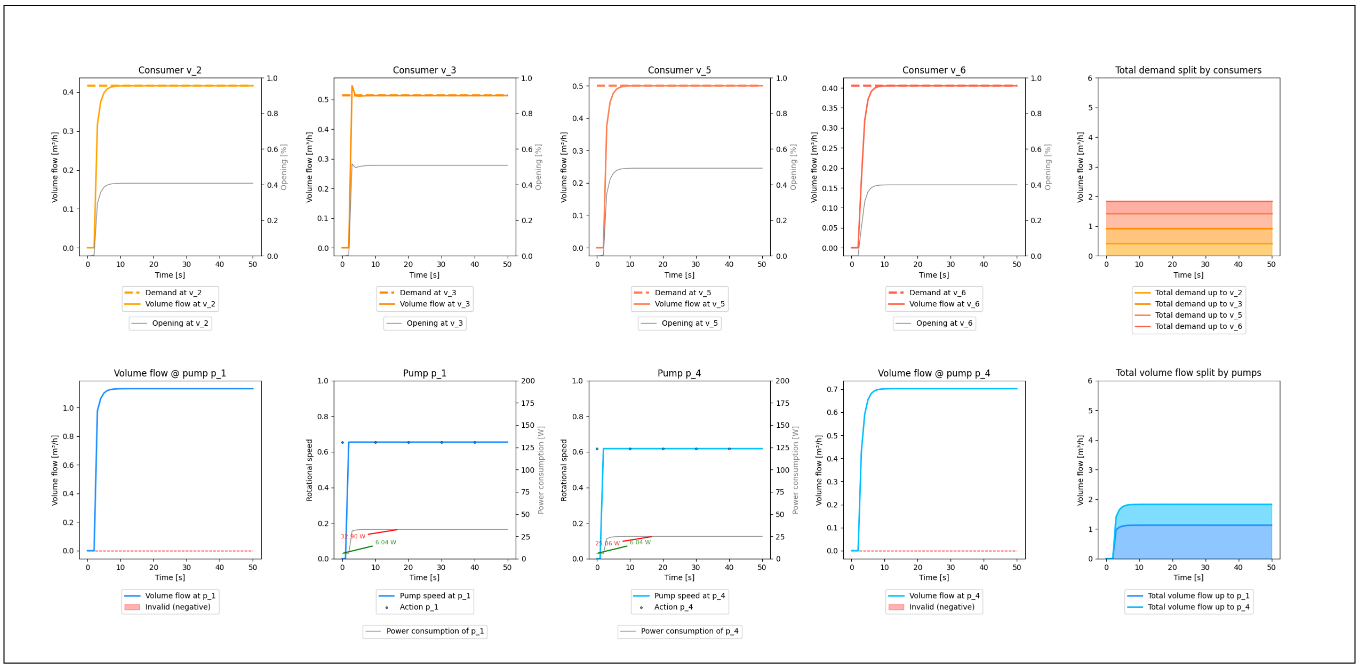


Figure 5.6.: Scenario with close to minimal load, showing reduced pump speeds in both pumps.

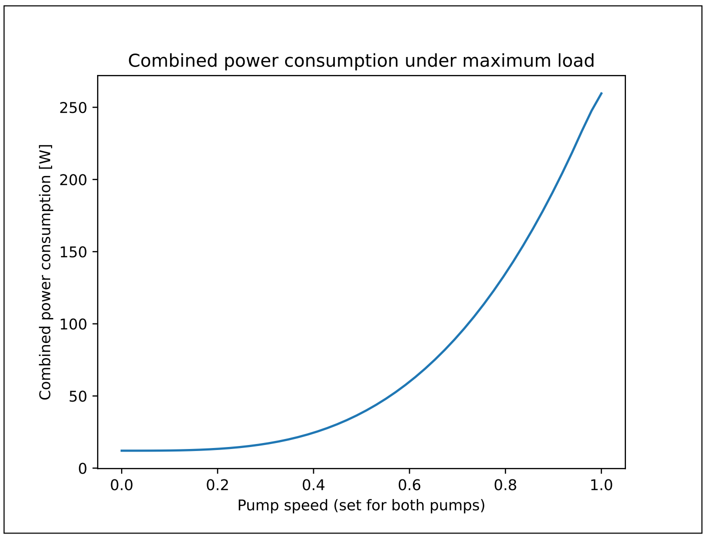


Figure 5.7.: Plot of the power consumption of both pumps, with maximum load in the network. The pump speed is set for both pumps equally.

The exact deviation from the demand cannot be deciphered from the valve openings o_i. However, it can be guaranteed that a consumer, whose valve is closed partially, is satisfied with the volume flow he is receiving:

$$o_i < 1 \implies |d_i - f_i| = 0 \tag{5.9}$$

An opening of $o_i = 1$ is ambiguous. It can either express that the pressure is precisely sufficient to provide the exact amount of volume flow necessary to meet the demand. However, it can also signify a deviation in the demand, as the consumer might not get enough water and might open its valve greedily to let as much water pass as possible, thereby also reducing the individual deviation from the demand as far as possible. In terms of pressure the first scenarios can be interpreted as the perfect pressure level in the system, while the latter can be interpreted as insufficient pressure.

In the case of an opening $o_i \neq 1$, the pressure directly correlates to the opening as well. The further the valve is closed the more pressure is available in the system, leading to the conclusion that a valve would not close very far in an efficient system.

When considering a network with just one consumer, the explanation above intuitively shows, that a network is efficient regarding its pressure level, when the valve opening of the consumer is infinitely close to 1, but never 1. Expanding this to a network with multiple consumers, an arbitrary topology and an arbitrary distribution of demand, the intuition gained can still be applied.

In a network with multiple consumers the topology and individual demands obviously affect the required openings as well. Individual consumers might indicate additional pressure available, because of height-differences in the topology or because a different consumer requires a much higher volume flow. But a single consumer can only indicate the local state of the network. But if every single consumer indicates additional pressure available, by showing a partially closed valve, then the pressure is undeniably too high. Vice versa, at least one consumer showing a nearly fully open valve indicates that the pressure in the system is tuned correctly. Thus, the maximum valve opening in the network can be considered as an indicator of efficiency in systems with multiple consumers. With that in mind, optimizing the system in terms of pressure using the valve openings seems logical.

In order to combine the demand and power signals into one metric, we define a completely new reward signal, which effectively removes any of the before mentioned trade-offs. The only issue left to solve is the ambiguous opening value of 1. We decide to generally treat a fully open valve as a deviation from the demand, as a valve opening approaching but never meeting the value of 1, should be approximately equally efficient. As the numerical simulation has finite precision and since demand-fulfillment has a higher priority, we decided, that targeting a specific opening o_{target} close to 1 would be most beneficial. That way the discontinuities in our reward function can be eased. We make use of the exponential function for the demand-deviation (Eq. 5.1) once more and combine two of them to the following piece-wise function:

$$r_{\text{opening}}(o_{\max}) = l + w_{\text{opening}}(h-l) \cdot \begin{cases} r_{\text{opening, left}}(o_{\max}), & o_{\max} \leq o_{\text{target}} \\ r_{\text{opening, right}}(o_{\max}), & o_{\max} > o_{\text{target}} \land o_{\max} < 1 \\ -1, & o_{\max} \geq 1 \end{cases} \tag{5.10}$$

with:

$$r_{\text{opening, left}}(o_{\max}) = a_{\text{left}} \exp\left(-b_{\text{left}}\sqrt{(o_{\max} - o_{\text{target}})^2 + s_{\text{left}}}\right) \tag{5.11}$$

$$r_{\text{opening, right}}(o_{\max}) = a_{\text{right}} \exp\left(-b_{\text{right}}\sqrt{(o_{\max} - o_{\text{target}})^2 + s_{\text{right}}}\right) \tag{5.12}$$

and $a_{\text{left}}, a_{\text{right}}, b_{\text{left}}, b_{\text{right}}$ defined exactly as in Equation 5.1, with the exception of different parameters. Using the points $(p_{x,\text{left}}, p_{y,\text{left}})$ and $(p_{x,\text{right}}, p_{y,\text{right}})$ the width of the exponential functions can be tuned. Using the common smoothness term, the peak can be tuned on both sides of the target opening o_{target}, which forms the maximum of the reward signal.

As a default, we define the function plotted in Figure 5.8, with a smoothness of $s = 0.0001$, a left cutoff point at $p_{\text{left}} = (0.5, 0.001)$ and a right cutoff at $p_{\text{right}} = (0.05, 0.001)$ relative to the target opening of $o_{\text{target}} = 0.95$.

The results show a significant improvement. In the upper load cases the strategy remains indistinguishable from the one learned using the relative power penalty. The pumps run at high speeds if needed and always meet the demand (Fig. 5.9). With decreasing demand, the pumps decrease their speed as well, in medium cases there is still no noticeable difference in pump speed or power consumption. But as to be expected, in the low load cases the opening is optimized nearly as well as in any other case, showing a better behavior compared to the relative power penalty. Even in a minimal load case the openings approached 80% (Fig. 5.10). While the difference in power consumption of roughly $20\,\text{W}$, is modest, it is nonetheless present. Out of all experiments, the target opening provides the best results.

However, the success comes at a cost. The target opening reward signal has two central assumptions, that are indispensable. Firstly, it assumes that a consumer will act greedily, in the sense that it will always try to optimize the opening in regards to its demand. This should not pose a problem, as this behavior is only required during training, where it can always be enforced by the simulation. Secondly, we assume a strictly monotonously increasing power to pump speed ratio as shown in Figure 5.7 or rather we assume that minimizing the pressure

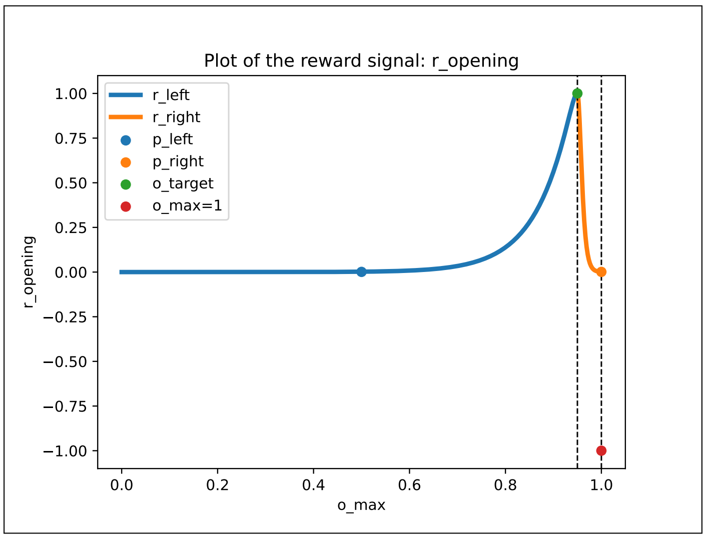


Figure 5.8.: A plot of the reward signal r_{opening} with default parameters. It shows the different segments of the piecewise function, as well the points that the exponentials are defined on.

at any time results in the lowest power consumption as well. If certain speeds of the pump are more efficient than others, this assumption can be violated and our target opening metric becomes flawed. Although the assumption is generally valid for most pumps, it is essential to remain aware of its potential limitations

Additionally, a major drawback is that a significant amount of exploration noise is needed for agents to discover effective strategies purely by chance. This is due to the lack of information about the severity of the deviation in cases where a valve is fully open. A fully open valve could indicate either a tiny negligible deviation or a complete failure to supply any water whatsoever. Yet, both deviations result in equal penalties. The agents struggle with this, as they cannot learn from experiences that all receive the same reward, preventing them from obtaining a meaningful gradient towards the functional strategies.

Consequently, the agents must explore with high noise. Only after randomly discovering a functional strategy that satisfies the demand can they begin training effectively. Therefore, training is slowed tremendously, especially in the early stages. With low noise, the agents often cannot escape from their initial strategy at all, remaining stuck.

To ensure convergence to a functional strategy, we adjust the training to run in two phases. First, we run a high-noise training phase to ensure full action space coverage, running it for only a few epochs. This serves the sole purpose of finding a working strategy. As the replay buffer is filled with very noisy experiences during this first training, it is reset, removing the influence of overly random actions. Then, it is refilled with low noise experiences to start the next phase. The second training phase serves to fine-tune the strategy from the first. This ensures convergence in our simple task, as running both pumps at full speed is always sufficient to meet any demand, which is a strategy easily discovered. However, in more complex tasks, even a high-noise warmup phase may not be sufficient to prevent agents from remaining stuck in their initial strategy.

5.1.4. Combined Reward for Optimal Agent Performance

Each of the reward signals from the previous sections has its own strengths and weaknesses:

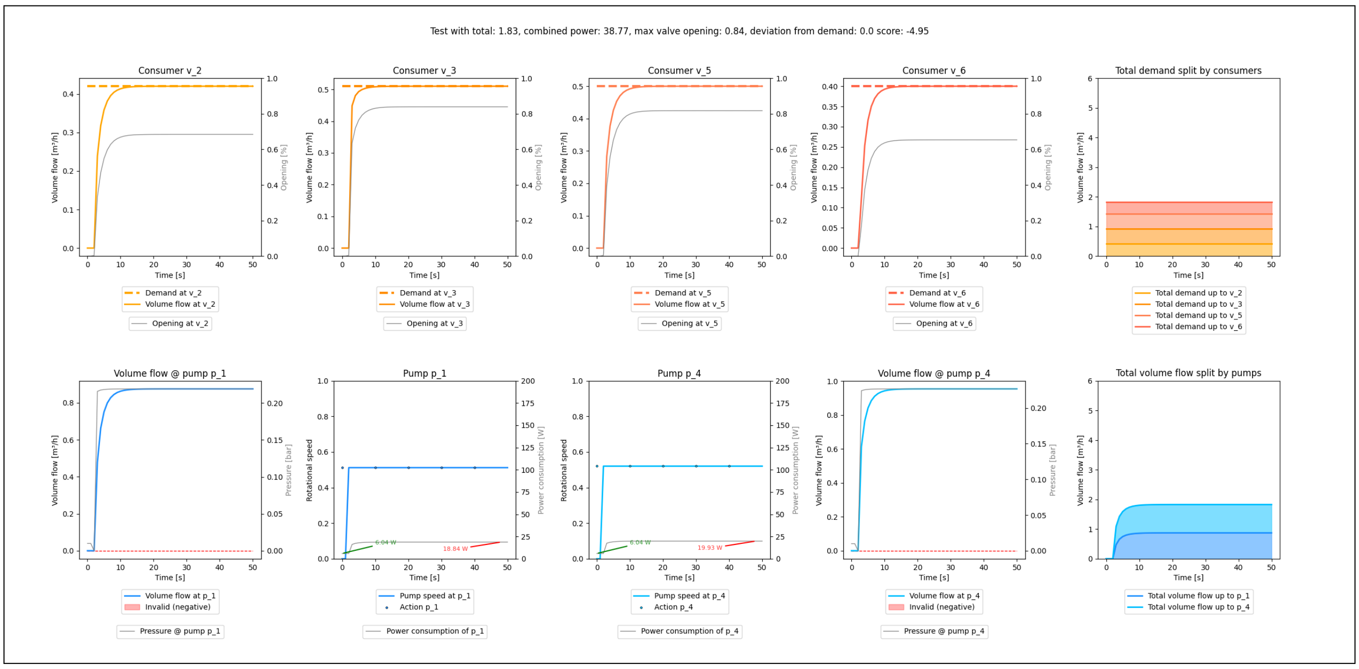


Figure 5.9.: Scenario with close to maximum load, showing similar behavior as working strategies before.

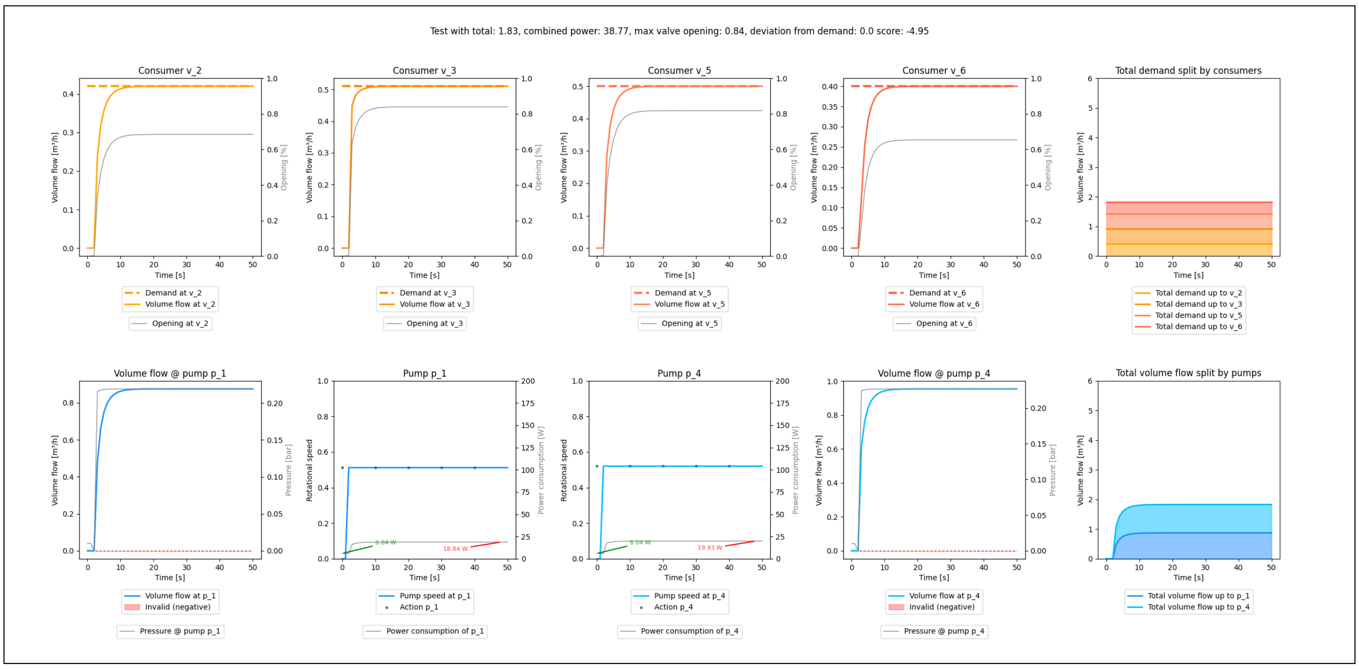


Figure 5.10.: Scenario with close to minimum load, showing the improved openings and improve power consumption, when compared with Figure 5.6.

The exponential function we defined for the deviation from the demand (Eq. 5.1) works well, independent of the width of the peak. It can form a key criterion to build on, as it ensures any resulting behavior is fully functional and does not compromise the demand.

The raw power penalty (Eq. 5.5) struggles with the trade-off between itself and the demand signal. With a low weight, it does not violate the demand, but it falls short in terms of efficiency. The latter can be improved with a higher weight, causing deviations from the demand in turn. Due to the difficulty in tuning the weights, we disregard it.

The power penalty relative to the output performs better. Tuning its weight to find a balance between a perfect demand fulfillment and lower power consumption is much easier. Once tuned, the efficiency in the higher load cases is by far superior and saves power efficiently in high load cases. The main drawback of this signal is the reduced accuracy in lower load cases, where, although significant amounts of power are saved, pressure is still wasted.

The target opening is performs equally as good. It creates reward signals that are more independent of the load in the network, treating low valve openings the same under any load. It also shows significant reductions in pump speed and power, leading to the most efficient behavior overall. Contrary to all other signals, it has the ability to reward a strategy that incorporates a safety buffer of additional pressure in the system and shows no trade-off between efficiency and functionality, making it unique in that sense. On the other hand, it is limited in its application, due to its two underlying assumptions.

In addition to all signals listed above, a negative flow penalty is essential to discourage the agents from abusing errors in the simulation.

Building on the individual strengths of the signals, we test a combined reward function integrating multiple signals to balance efficiency throughout all load cases, demand fulfillment and system robustness. As a fundamental component we use the demand-deviation with a high weight. Naturally, we use a negative flow penalty. With a significant lower weight, the agents are guided towards efficient pump speeds by the relative power penalty. The weight is much lower compared to the others, as the power penalty is based on the power values without any mapping, leading to a much greater range for the output of the signal compared to the exponential terms which are limited to their individual ranges $[l, h]$. To improve the accuracy in low load cases, a target opening is also included.

The resulting reward function (Eq. 5.13) compensates the weaknesses of the two efficiency metrics with the strengths of the other. The convergence towards efficient metrics is helped, as a large fraction of the reward-consisting of the power penalty and the demand signal-helps to obtain a meaningful gradient. The power consumption is not entirely disregarded, as it is explicitly included with its own signal. On the other hand, the accuracy in lower load cases is improved by the target opening. In this setup the importance of the assumption of a strictly monotonously increasing power to speed ratio is also lessened, due to the power penalty being added.

$$r(\delta_{\max}, o_{\max}, f_1, f_4, P_1, P_4) = \begin{aligned} & r_{\text{demand}}(\delta_{\max}) \\ & + r_{\text{pow-per-flow}}(f_1, f_4, P_1, P_4) \\ & + r_{\text{neg-flow}}(f_1, f_4) \\ & + r_{\text{opening}}(o_{\max}) \end{aligned} \tag{5.13}$$

The optimized values after a tuning of parameters and weights can be seen in Table 5.1.

Table 5.1.: Parameters for the individual reward functions in Eq. 5.13

Parameter	r_{demand}	$r_{\text{pow-per-flow}}$	$r_{\text{neg-flow}}$	r_{opening}	
				left	right
w	10	2	3	0.1	
l	-1	/	/	0	
h	0	/	/	1	
p_x	0.05	/	/	0.5	0.05
p_y	0.001	/	/	0.001	0.001

5.2. Impact of Partial Observability on Agent Performance

Building upon the results from the previous chapter, we investigate the effect of partial observability on the performance of the agents. The task is defined on the circular network. Using the optimized reward function in Equation 5.13, we train the agents on the previously used fully observable state, consisting of the four demands. In addition, we introduce a new partially observable system, in which each pump observes only the two demands of the consumers closest to them. This results in the pump p_1 observing the two demands of consumers c_2 and c_5 and in consumers c_3 and c_6 being observed by pump p_4. Otherwise, the setup of the circular network remains unchanged. That way, both trainings are initialized equally, enabling a direct comparison of the effects of partial observability on efficiency and performance.

The first comparison is done on a test set, consisting of 19 tests, selected to show different load cases in the network. We select 5 distinct total loads cases:

1. Minimal load: all demands are at the minimum.
2. Low load: all demands are at the minimum except for one, resulting in a total load of 25%.
3. Medium load: two consumers demand the maximum flow and two demand the minimum flow, resulting in a total load of 50%.
4. High load: three demands are at the maximum and one is at the minimum, resulting in a total load of 75%.
5. Maximal load: all demands are at the maximum.

For each of the load cases we test every combination of the above described total demands, with an additional test, where the total demand is split uniformly across the different consumers. The exception being the minimal and maximal load, where the uniform split forms the only possible configuration. Table 5.2 visualizes all test cases and their individual demands.

We compare four different methods over seven different seeds: fully-observable IDDPG trained over 100 epochs, partially-observable IDDPG trained over 100 epochs, partially-observable IDDPG that was early-stopped after 20 epochs and finally a strategy that always runs the pumps at full power as a baseline. The latter is obviously far from efficient.

Table 5.2.: An overview of the individual and total demands in the 19 selected test cases.

Total load ($\mathrm{m^3\,h^{-1}}$)	Test #	d_2 ($\mathrm{m^3\,h^{-1}}$)	d_3 ($\mathrm{m^3\,h^{-1}}$)	d_5 ($\mathrm{m^3\,h^{-1}}$)	d_6 ($\mathrm{m^3\,h^{-1}}$)
Minimal: 1.6	0	0.4	0.4	0.4	0.4
Low: 2.6	1	0.65	0.65	0.65	0.65
	2	1.4	0.4	0.4	0.4
	3	0.4	1.4	0.4	0.4
	4	0.4	0.4	1.4	0.4
	5	0.4	0.4	0.4	1.4
Medium: 3.6	6	0.9	0.9	0.9	0.9
	7	1.4	1.4	0.4	0.4
	8	0.4	1.4	1.4	0.4
	9	0.4	0.4	1.4	1.4
	10	1.4	0.4	0.4	1.4
	11	1.4	0.4	1.4	0.4
	12	0.4	1.4	0.4	1.4
High: 4.6	13	1.15	1.15	1.15	1.15
	14	1.4	1.4	1.4	0.4
	15	1.4	1.4	0.4	1.4
	16	1.4	0.4	1.4	1.4
	17	0.4	1.4	1.4	1.4
Maximal: 5.6	18	1.4	1.4	1.4	1.4

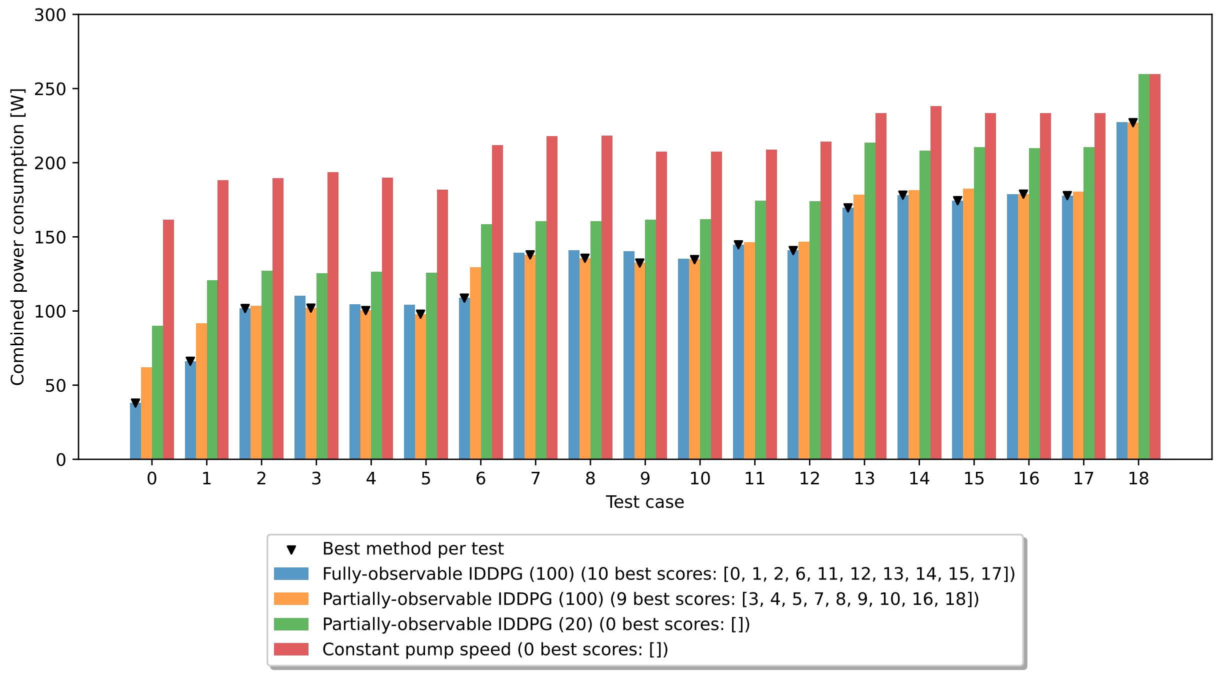


Figure 5.11.: Test results for the power consumption on the selected 19 test cases from Table 5.2 compared for the four considered methods.

The results for power consumption, maximum valve opening and deviations from the demand are shown in Figure 5.11, Figure 5.12 and Figure 5.13. For the parameters used in these trainings please refer to appendix B.

For every test case except the maximum and for every method the results show a clear improvement in power consumption and valve opening over the naive implementation of a constant pump speed of 100%.

The difference between the individual methods is less uniform. The test cases 0, 1, 6 and 13 with even demand distribution, show the clearest trend in power consumption as well as in the valve openings. In each case the superiority of full observability is evident. The two partially observable methods follow, with the early-stopped one performing consistently worse, which is generally the case for all tests. In the remaining test cases, the power consumption of the fully observable IDDPG is comparable to that of the longer partially observable training, with each approach outperforming the other in different instances, but always by a negligible amount.

A second trend can be noted in the results for the deviation from the demand. Only early-stopped partially-observable IDDPG and fully-observable IDDPG are robust. The agents trained with partial observability show deviations in the test cases 2, 3, 4 and 5, as well as 11 and 12 after training for more than 20 epoch (Fig. 5.13). The deviations appear to result from overfitting, as they occur only after 20 epochs and are linked to extreme edge cases. All cases share the characteristic, that one pump observes minimal demand, while the other pump observes either one valve with maximum demand or, in tests 11 and 12, two valves, leading to the greatest discrepancy between the two observed states allowed by our demand constraints. Especially with these state differences, it is difficult to infer the global state of the network from the partial observations of the agents. One agent assumes the network to be under low load, while the other agent is struggling severely in need of additional support from the idling agent. Due to the excess-shifting in our demand calculation (Chap. 4.1.1) these extreme discrepancies are improbable and underrepresented in our training data, making it more likely

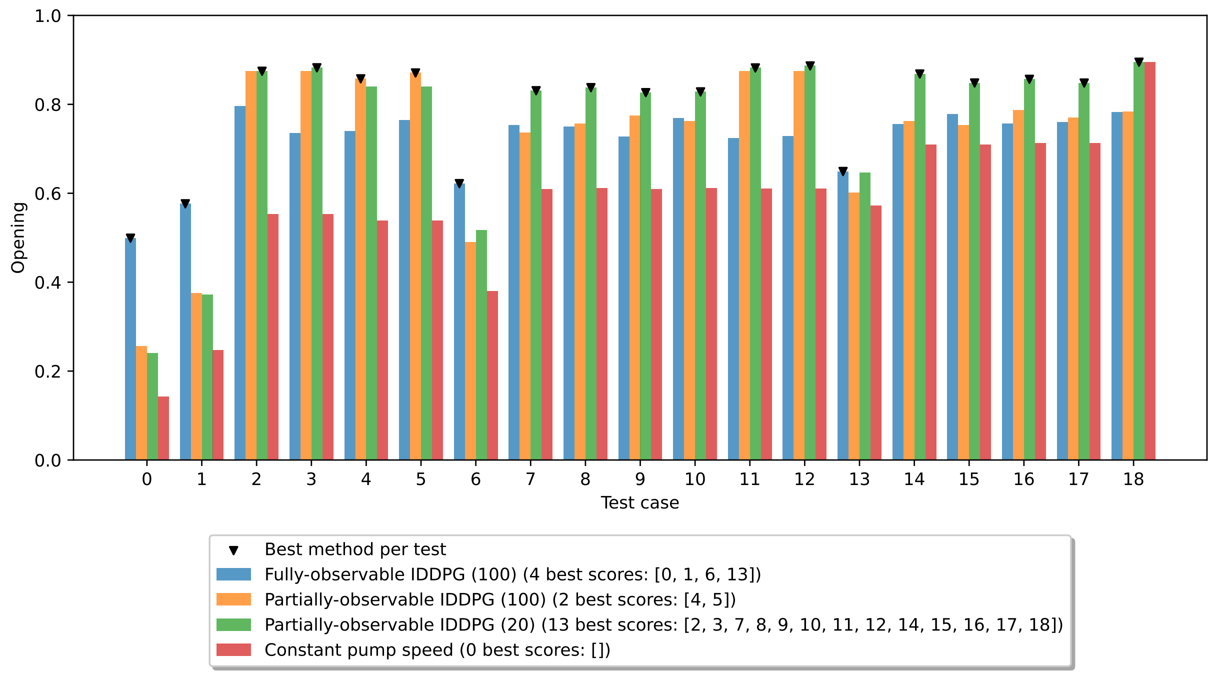


Figure 5.12.: Test results for maximum of all openings on the selected 19 test cases from Table 5.2 compared for the four considered methods.

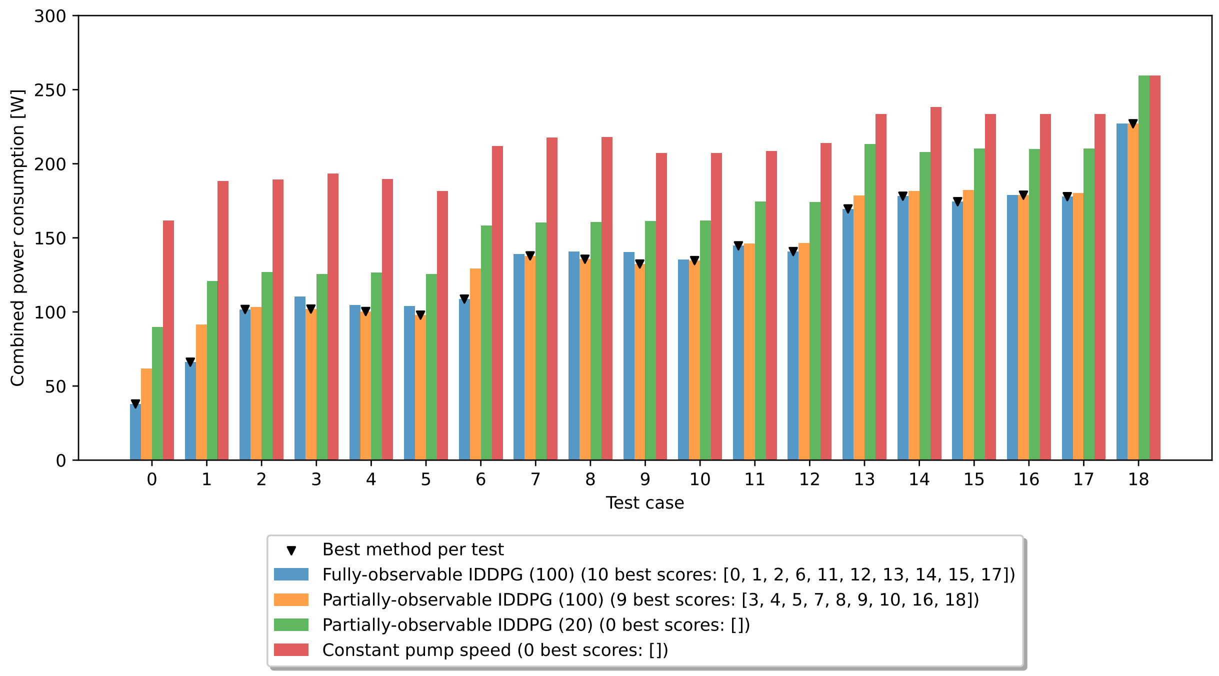


Figure 5.13.: Test results for the deviation from the demand on the selected 19 test cases from Table 5.2 compared for the four considered methods.

for the agents to overfit to the far less severe cases. Referring back to the distribution of the standard deviation (Fig. 4.4) this is fairly obvious, as the failing test cases all show a standard deviation of ≈ 0.5. The problem of overfitting is corroborated further by the fact, that the standard evaluation, conducted after each training over 1000 entries drawn from the same distribution as the training data, shows no deviations, even after 100 epochs. In conclusion, the absence of information about the demands of other consumers, combined with frequent experiences where observation differences are reduced, leads the agents to misjudge the global state of the network. This misjudgment causes deviations, as the pump that receives maximal demand lacks sufficient support from the rest of the system.

In addition to this, another noteworthy pattern emerges when examining the power consumption behavior across different total loads. The difference in efficiency between the fully-observable IDDGP agents and the ones trained for 100 epochs in partial observability depends on the overall load in the network. Especially in evenly distributed load cases this can be verified. With an increasing total load in the systems the gap between both methods narrows until they are indistinguishable in the maximal load case. This is intuitive as the likelihood for big differences in demand between individual consumers diminishes with higher load, thereby reducing the necessity for global information about all consumers.

To further analyze this pattern we run another test. Instead of selecting the test cases manually we randomly generate a test set of much larger size. Similarly to our demand calculation for the training data, we draw the individual demands of the consumers using a dirichlet distribution and excess-shifting. However, we do not draw the input value for the total demand from a distribution but from a linearly increasing set of total loads. This results in a test set with randomly sampled individual demands that are sorted by the global demand. This way the effects of the global load on the difference between two methods can be studied in detail. We run a direct comparison of the two methods mentioned above, the fully-observable IDDPG and the partially-observable IDDPG for 100 epochs.

The results for both, the power consumption (Fig. 5.14) and the opening (Fig. 5.15) show the previously noticed pattern in more detail. If the load in the network exceeds 50% both methods perform very much alike, favoring partial-observability slightly, although this negligible difference is likely to be caused by variance in the test set or variance in the trainings of the individual seeds. In any case, the difference in the upper half of the load cases is negligible. When the network load falls below 50%, the fully-observable approach achieves notably better results. A clear correlation of the degree of improvement to the total network load can be observed. With decreasing load the difference increases until it balances out at roughly $35\,\mathrm{W}$ of power being saved, with the maximum valve opened further by close to 40%.

Ultimately, all approaches presented can be used to significantly reduce power consumption and to increase efficiency compared to naive methods like a constant pump speed. Fully-observable agents show consistently better results than the other methods, if small trends in the test data are disregarded, matching with the expectation, since they are able to utilize more information about the network. In the case of partial observability a choice has to be made. When overfitting the agents to a specific network they show results close to full observability, however their ability to extrapolate to different load cases decreases, affecting their robustness. When they are early-stopped they show better extrapolation to different load cases, but their efficiency is reduced. Nonetheless, their efficiency is significant, halving the power consumption in minimal loadcases.

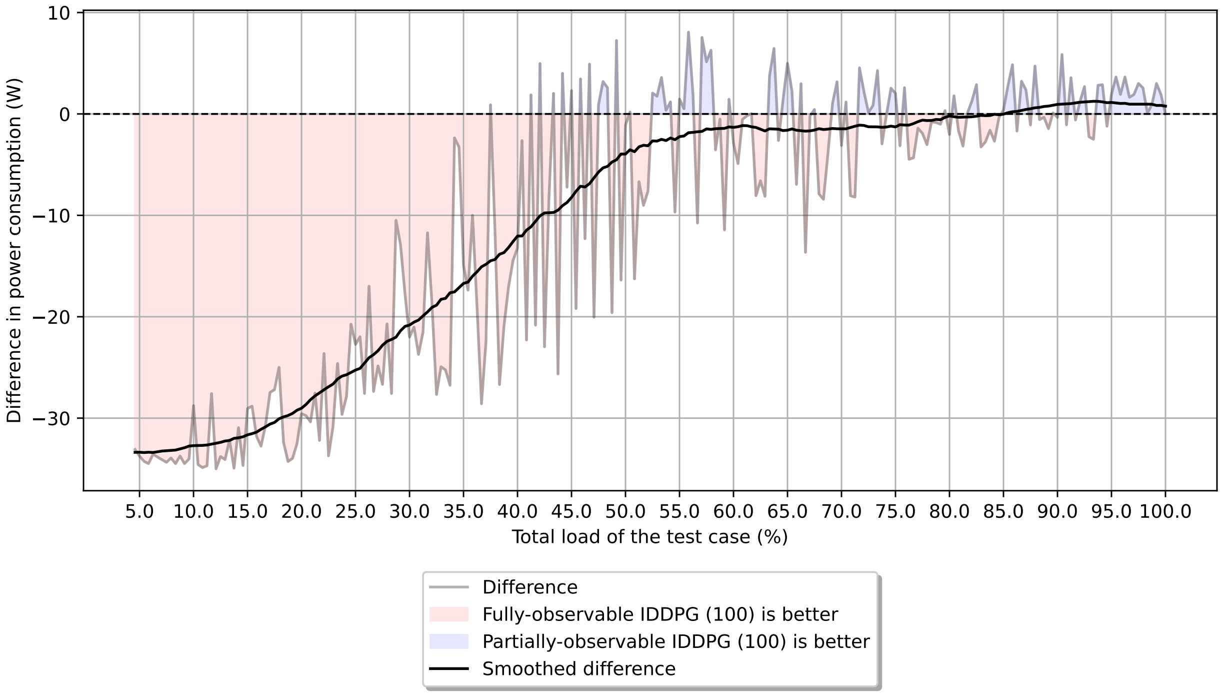


Figure 5.14.: Test results for the direct comparison of fully-observable IDDPG (100) and partially-observable IDDPG (100) in power consumption depending on the total load in the network, with an additional representation of the difference smoothed with a gaussian kernel with a width of 20.

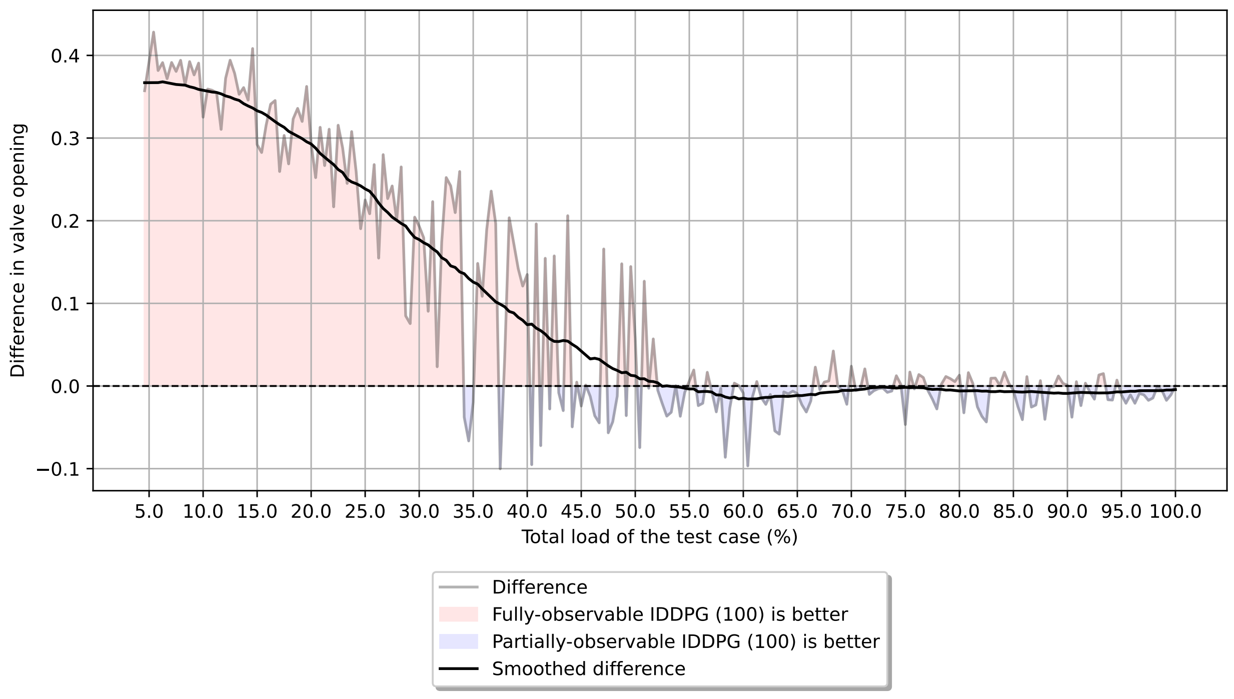


Figure 5.15.: Test results for the direct comparison of fully-observable IDDPG (100) and partially-observable IDDPG (100) in the maximum opening depending on the total load in the network, with an additional representation of the difference smoothed with a gaussian kernel with a width of 20.

5.3. Temporal reasoning for water preservation

As a final experiment, we compare the ability to cooperate of the two different MARL algorithms IDDPG and FACMAC. The experiment is performed on the tank network described in section 4.2. The difficulty in this task resides in the temporal reasoning required to understand the concept of nightly water preservation. Figure **??** displays the demand over the cycle of a full day.

While running the pump at full speed is generally sufficient to meet the demand during most of the day, the peak hours at 10 in the morning and 8 in the evening require a volume flow exceeding the pump's capacity. Hence, effective cooperation and coordination between the pump and the tank is essential. The agents have to preserve water in the tank when the demand is low and use it during peak hours to support the pump.

Specifically, the pump must provide additional pressure in low demand periods at the same time as the tank is opening its valve, only then can water flow into the tank. In some networks it is sufficient for demand-fulfillment to simply increase the inertia of the system by keeping the tank valve open throughout the entire day. However, this task is specifically designed to only test the temporal planning capabilities of the agents, disregarding efficiency for the first time. Thus, we scale the demand and the components of the system accordingly to increase the complexity of the task. The tank not only needs to serve as an increase in inertia, but needs to strategically decide how to spend the preserved water. Otherwise, it will run empty before the second peak in the evening. Also, we only consider the demand-fulfillment for the reward.

For both algorithms, we perform an extensive grid search to optimize the learning rates and other hyperparameters. The results for IDDPG, however, are unsatisfactory. In every training, the algorithm fails to form a strategy that consistently fulfills the demand. While it often meets the first peak, it never manages to preserve water for the second, and at times, it even fails to provide enough water for either peak.

FACMAC behaves similarly in less successful runs, but it shows functional strategies in more successful runs. In multiple trainings a strategy is discovered, that satisfies the demand throughout the entire day. However, the training is unstable, as the agents deviate from their learned behavior intermittendly. While they sometimes converge back towards a working strategy, they frequently deviate entirely.

A second, finer grid search fixes these issues for FACMAC. With the optimized parameters, the algorithm is fully capable of learning the dynamics of this temporal and cooperative task. See appendix C

This experiment shows the superiority of FACMAC in cooperative tasks compared to IDDPG. The coordination of the agents in IDDPG is heavily reliant on exploring appropriate action pairs for both tank and pump simultaneously. In contrast, FACMAC uses centralized learning and backpropagates through its mixing network to the individual critics and can thereby adopt improvements in the tactics of the individual agents more easily. Even when other agents do not consistently cooperate during training, the central gradient estimator of FACMAC can update the weights more precisely. At the same time, IDDPG treats the actions of other agents as part of the environment, evaluating only the effects of individual actions of their respective components. As a result, it can get stuck, if these individual actions do not form a meaningful gradient towards better performance on their own. In the results, this limitation becomes apparent, as IDDPG consistently fails to converge to an effective strategy, reflecting its susceptibility to getting stuck in suboptimal solutions. We conclude that decentralized learning is sufficient for more independent tasks, as seen in the circular network, while cooperative tasks in networks like our tank network require centralized learning for effective operation.

6. Conclusion

In this thesis we explore the application of Multi-agent reinforcement Learning in the operation of urban WDNs, with a focus on efficiency. We show that its application can be extended from industrial WDNs to urban WDNs with more cyclic topologies, ensuring efficient operation under more complex flow dynamics.

We investigated several reward structures, proving that targeting a specific maximum valve opening excels at optimizing the pressure level in the network across different load cases, while a power penalty relative to the volume flow provides a more accurate definition of efficiency. Ultimately, we come to the conclusion, that a multifaceted reward function, combining demand deviation, power penalties and a maximum target valve opening, yields the most robust and efficient results. By successfully applying the resulting reward function, we were able to train agents that are capable of balancing a reduced power consumption with a flawless compliance with the demand.

Furthermore, our results highlight the importance of available information during execution in our comparison between agents observing the entire network and agents observing only parts of the network. We confirm the expectation that agents working in a full-observable setting achieve the highest efficiency overall, while also showing the ability to extrapolate to new unseen load cases. We furthermore discover a trade-off in the partial observability setting. Long trainings prone to overfitting show unexpectedly similar efficiency to fully-observable settings at the cost of robustness. In turn, early-stopped trainings prove to be more robust and better at extrapolating to new load cases, at the cost of efficiency. Nonetheless, our efficiency metric guarantees a more efficient execution in any case compared to naive baselines.

Finally, we demonstrate the significance of centralized training for cooperative temporal tasks in an experiment comparing IDDPG with FACMAC on their agents' ability to preserve water. The centralized learning approach from FACMAC enables its agents to successfully preserve water for a later use during peak demand hours, otherwise the pumps alone are incapable of fulfilling the demand. In contrast IDDPG fails to meet the demand consistently, due to its decentralized learning hindering its ability to converge to a successful cooperation between agents.

In conclusion, this thesis establishes that MARL can significantly enhance efficiency in the operation of WDNs. We introduced tailored reward signals, studied the impact of observability, as well as the difference between centralized and decentralized learning approaches, laying the groundwork for smart, sustainable water distribution systems using MARL.

7. Outlook

While we have shown that MARL can be used effectively to operate WDNs efficiently, there is much left to be explored.

In future work, it could be investigated how well agents can learn key characteristics of any general network topology, testing their ability to handle networks with previously unseen topologies. In this context structural data could be incorporated into the state, either in a simple form of only rough information such as distances between components or even in the complex form of a mathematical model of the entire topology. That way the gap between MARL and optimal control strategies could potentially be reduced, while removing the need to retrain the agents for different topologies, if successful. Another option to make MARL more applicable to arbitrary WDNs is the investigation of Transfer learning, which could enable agents to transfer knowledge learned in one topology to other, unseen topologies.

Additionally, Recurrent Neural Networks (RNNs) could be added and investigated. By keeping track of the behavior of the network and the agents' past actions over multiple time steps, RNNs could improve the agents' understanding and anticipation of the global state, potentially enhancing performance in environments with partial observability. Long-term consequences of earlier actions could also be grasped more easily. This would help with temporal dependencies and planning, crucial aspects of operating WDNs with storage elements.

Bibliography

[1] L. Kraemer and B. Banerjee, “Multi-agent reinforcement learning as a rehearsal for decentralized planning,” *Neurocomputing*, vol. 190, pp. 82–94, 2016.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2019.

[3] B. Peng, T. Rashid, C. A. S. de Witt, P.-A. Kamienny, P. H. S. Torr, W. Böhmer, and S. Whiteson, “Facmac: Factored multi-agent centralised policy gradients,” 2021.

[4] K. T. Logan, J. M. Stürmer, T. M. Müller, and P. F. Pelz, “Comparing approaches to distributed control of fluid systems based on multi-agent systems,” 2023.

[5] N. Trifunović, *Introduction to Urban Water Distribution: Theory*. Routledge, 2020.

[6] R. Pérez and G. Sanz, “Modelling and simulation of drinking-water networks,” in *Real-time monitoring and operational control of drinking-water systems*, Springer International Publishing, 2017.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2nd ed., 2018.

[8] “Deep deterministic policy gradient — spinning up documentation,” 2025. [Online; accessed: Apr. 10, 2025] `https://spinningup.openai.com/en/latest/algorithms/ddpg.html`.

[9] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML’14, p. I–387–I–395, JMLR.org, 2014.

[10] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

[11] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” 2020.

[12] Modelica Association, “Modelica,” 2025. [Online; accessed: Nov. 21, 2024] `https://www.modelica.org/`.

[13] Modelica Association, “Functional mock-up interface,” 2025. [Online; accessed: Nov. 21, 2024] `https://fmi-standard.org`.

[14] D. Inturri, K. Logan, M. Leštáková, T. C. Meck, and P. Pelz, “Sofirpy - co-simulation of functional mock-up units (fmus) with integrated research data management.”.

[15] P. Fritzson, A. Pop, K. Abdelhak, A. Asghar, B. Bachmann, W. Braun, D. Bouskela, R. Braun, L. Rogovchenko-Buffoni, F. Casella, R. Castro, R. Franke, D. Fritzson, M. Gebremedhin, A. Heuermann, B. Lie, A. Mengist, L. Mikelsons, K. Moudgalya, and P. Östlund, "The openmodelica integrated environment for modeling, simulation, and model-based development," *Modeling, Identification and Control: A Norwegian Research Bulletin*, vol. 41, pp. 241–295, 10 2020. [Online; accessed: Nov. 21, 2024] `https://www.mic-journal.no/ABS/MIC-2020-4-1.asp/`.

[16] Wdwd, "Sprungantwort eines idealen p-reglers." Wikipedia, The Free Encyclopedia, 2010. [Online; accessed: Apr. 10, 2025] `https://commons.wikimedia.org/wiki/File:Idealer_P_Sprungantwort.svg`.

[17] Wdwd, "Sprungantwort eines idealen pi-reglers," 2010. [Online; accessed: Apr. 10, 2025] `https://commons.wikimedia.org/wiki/File:Idealer_PI_Sprungantwort.svg`.

[18] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters, "Mushroomrl: Simplifying reinforcement learning research," *Journal of Machine Learning Research*, vol. 22, no. 131, pp. 1–5, 2021.

[19] A. Reddi, M. Tölle, J. Peters, G. Chalvatzaki, and C. D'Eramo, "Robust adversarial reinforcement learning via bounded rationality curricula," 2023.

[20] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Multivariate Distributions*, vol. 1. New York: Wiley, 2 ed., 2000.

A. Explanation of Training Parameters

Table A.1.: Table explaining each parameter for our training scripts

Parameter	Description
`seed`	The seed value set at the beginning of the training process, ensuring reproducibility across runs by fixing random number generators (e.g., Python, Numpy, Torch).
`gamma`	The discount factor of the environment.
`lr_actor`	The learning rate for the actor optimizer.
`lr_critic`	The learning rate for the critic optimizer.
`critic_warmup`	A boolean flag indicating whether to run a two phase training helping the critic initialize with a warmup training phase with high noise, before the full training begins.
`initial_replay_size`	The initial number of experience samples stored in the replay buffer before starting training.
`max_replay_size`	The maximum size of the experience replay buffer, limiting how much past experience the agent can use during training.
`batch_size`	The number of samples used in each fitting step when updating the model parameters.
`n_features`	The size of the hidden layer of the neural network.
`tau`	The soft update parameter used for Target Networks, controlling the rate at which the target network is updated towards the online network.
`sigma`	A list of tuples representing noise parameters for exploration in the environment. The tuple specifies an epoch-count and the sigma value for the policy noise at that time. The values in between are interpolated using the `decay_type`
`decay_type`	The type of decay function applied to the noise values, either exponential or linear decay.
`n_epochs`	The total number of epochs the agent will train for.

Parameter	Description
`n_episodes_learn`	The number of episodes used for the agent to learn in the training step in each epoch.
`n_episodes_test`	The number of episodes the agent will be tested on in each epoch, used to evaluate performance during training.
`n_steps_per_fit`	The number of time steps per training fit, determining how frequently the agent's weights are updated during training.
`num_agents`	The number of agents in the environment.
`n_episodes_final`	The number of episodes the agent will be evaluated on during the final evaluation.
`n_episodes_final_render`	The number of episodes to render or visualize during the final evaluation phase, for visual evaluation.
`n_epochs_per_checkpoint`	The number of epochs between saving a checkpoint of the model, allowing recovery and analysis after training sessions.
`state_selector`	A list of indices representing which elements of the state space to use for training, allowing selective focus on relevant state features.
`observation_selector`	A list of index pairs representing which observations should be used for learning for each agent.
`criteria`	A dictionary specifying which reward signals to use and supplying the weights and parameters for each signal.
`demand`	The demand profile for the environment, specifying the distribution or other key characteristics dependent on the network
`job_counter`	A counter tracking the number of jobs that have been completed or processed. (Only relevant for the wandb integration and is automatically set)

B. Parameters For The Observability Experiment

Table B.1.: Table showing the parameters used for the three methods Fully-observable IDDPG (100), Partially-observable IDDPG (100) and Partially-observable IDDPG (20) in experiment 5.2

Parameter	Fully-Observable IDDPG	Partially-Observable IDDPG
seed	$0, 42, 420, 4366, 7359, 738377, 873837$	
gamma	0.99	
lr_actor	2×10^{-4}	
lr_critic	1×10^{-3}	
critic_warmup	True	
initial_replay_size	5000	
max_replay_size	$60\,000$	
batch_size	200	
n_features	80	
tau	5×10^{-3}	
sigma	$(0, 1), (1, 0.075), (50, 0.025), (70, 0.01)$	
decay_type	exponential	
n_epochs	100	100 (full) 20 (early-stopped)
n_episodes_learn	200	
n_episodes_test	6	
n_steps_per_fit	1	
num_agents	2	
n_episodes_final	1000	
n_episodes_final_render	200	
n_epochs_per_checkpoint	20	
state_selector	$(0, 1, 2, 3)$	

Parameter	Fully-Observable IDDPG	Partially-Observable IDDPG
`observation_selector`	$[(0, 1, 2, 3)]$	$[(0, 2), (1, 3)]$
`criteria`	See dictionary in appendix B.1	
`demand`	`[ "uniform_global", 0.4, 1.4 ]`	

B.1. Criteria

```
{
    "demand": {
        "w": 10.0,
        "bound": 0.05,
        "value_at_bound": 0.001,
        "max": 0,
        "min": -1
    },
    "power_per_flow": {
        "w": 0.1
    },
    "negative_flow": {
        "w": 3.0,
        "threshold": -1e-06
    },
    "target_opening": {
        "max": 1,
        "min": 0,
        "w": 2.0,
        "target": 0.9,
        "left_bound": 0.7,
        "value_at_left_bound": 0.001,
        "right_bound": 0.025,
        "value_at_right_bound": 0.001
    }
}
```

C. Parameters For The Temporal Reasoning Experiment

Table C.1.: Table showing the optimal parameters for FACMAC in experiment 5.3

Parameter	Fully-Observable IDDPG	Partially-Observable IDDPG
`seed`	0	
`gamma`	0.99	
`lr_actor`	1.05×10^{-4}	
`lr_critic`	4×10^{-4}	
`critic_warmup_episodes`	30	
`initial_replay_size`	5000	
`max_replay_size`	$35\,000$	
`batch_size`	200	
`n_features`	80	
`tau`	5×10^{-3}	
`sigma`	$(0, 1), (1, 0.075), (20, 0.01)$	
`decay_type`	`exponential`	
`n_epochs`	100	150
`n_episodes_learn`	6	
`n_episodes_test`	3	
`n_steps_per_fit`	1	
`num_agents`	2	
`n_episodes_final`	1	
`n_episodes_final_render`	1	
`n_epochs_per_checkpoint`	25	
`state_selector`	(0)	
`observation_selector`	$[(0)]$	$[(0)]$
`criteria`	See dictionary in appendix C.1	

Parameter	Fully-Observable IDDPG	Partially-Observable IDDPG
demand	"tagesgang"	

C.1. Criteria

```
"demand": {
        "w": 1.0,
        "bound": 0.3,
        "value_at_bound": 0.001,
        "max": 0,
        "min": -1
    }
```