Technical University of Crete

School of Production Engineering and Management

Diploma Thesis

Energy efficient control of production lines with the use of Reinforcement Learning

Manganas Grigorios

Chania, September 2025

Πολυτεχνείο Κρήτης

Σχολή Μηχανικών Παραγωγής και Διοίκησης

ΠΟΛΥΤΕΧΝΕΙΟ
ΚΡΗΤΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΠΑΡΑΓΩΓΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

Διπλωματική Εργασία

Ενεργειακά αποδοτικός έλεγχος γραμμών παραγωγής με χρήση Ενισχυτικής Μάθησης

Μαγγανάς Γρηγόριος

Χανιά, Σεπτέμβριος 2025

Abstract

Optimization of the production processes is a major topic in industrial engineering. New technologies such as Reinforcement Learning (RL) have emerged, allowing for novel approaches to the issue of optimal production control in all its aspects, including production planning, maintenance and safety administration, logistics etc. Research on this field focuses on training of agents in simulation environments mimicking the behavior of their real-world counterparts. In this thesis, a simple 2-machine environment with intermediate buffers and a client queue is developed and fitted with an RL agent as its master controller, using the Proximal Policy Optimization (PPO) method. This agent is then trained on a number of different scenarios, simulating environment variable fluctuations often met in real-world production systems, such as increased demand or production costs. Though in a smaller scale than a real industrial facility, training showed effective control of the production line and, in some of the trained agents, resilience in more demanding scenarios. These findings may assist in the future development of RL agents for use in the field of industrial control and pave the way for more challenging problems to be tackled.

Περίληψη

Η βελτιστοποίηση των γραμμών παραγωγής είναι ένα μείζον ζήτημα στον τομέα της βιομηχανίας. Νέες αναδυόμενες τεχνολογίες όπως η Ενισχυτική Μάθηση (Reinforcement Learning, RL) επιτρέπουν την εφαρμογή νέων προσεγγίσεων στο πρόβλημα του βέλτιστου ελέγχου της παραγωγής σε όλους του τους τομείς, περιλαμβανομένου του προγραμματισμού παραγωγής, της διαχείρισης εργασιών συντήρησης και ασφάλειας, της εφοδιαστικής αλυσίδας κλπ. Η υπάρχουσα έρευνα σε αυτό το πεδίο εστιάζει στην εκπαίδευση πρακτόρων σε περιβάλλοντα προσομοιώσεων που μιμούνται τη συμπεριφορά υπαρκτών συστημάτων παραγωγής. Σε αυτή την εργασία, αναπτύχθηκε μια απλή γραμμή παραγωγής με 2 μηχανές, ενδιάμεσες αποθηκευτικές μονάδες και ουρά αναμονής πελατών, στην οποία εφαρμόστηκε ένας πράκτορας Ενισχυτικής Μάθησης σαν κύριος ελεγκτής που χρησιμοποιεί τη μέθοδο Βελτιστοποίησης Πλησιέστερης Πολιτικής (Proximal Policy Optimization, PPO). Αυτός ο πράκτορας έπειτα εκπαιδεύεται σε διαφορετικά σενάρια που προσομοιώνουν μεταβολές στις περιβαλλοντικές παραμέτρους, όπως αυξημένη ζήτηση προϊόντων ή κόστους παραγωγής. Παρ'όλο που η εκπαίδευση γίνεται σε μικρότερη κλίμακα από μια πραγματική βιομηχανία, η διαδικασία έδειξε αποτελεσματικό έλεγχο των μηχανών της γραμμής παραγωγής και, σε κάποιους από τους πράκτορες, αντοχή στις συνθήκες των πιο απαιτητικών σεναρίων. Αυτά τα ευρήματα ενδέχεται να βοηθήσουν στη μελλοντική ανάπτυξη πρακτόρων Ενισχυτικής Μάθησης για χρήση στον βιομηχανικό έλεγχο και να δείξουν το δρόμο για την αντιμετώπιση πιο προκλητικών ζητημάτων.

Thesis Committee

Professor Ioannidis Efstratios (Supervisor) | School of Production Engineering and Management.

Professor Doitsidis Eleftherios | School of Production Engineering and Management.

Professor Tsinarakis Georgios | School of Production Engineering and Management.

Acknowledgements

I would like to thank the following people, who have played a critical role in me reaching this point in my life and being able to complete this thesis.

First, I would like to wholeheartedly thank my parents for their never-ending love and support throughout my whole academic journey, since its early beginnings and to this day.

I would also like to thank my brother, my extended family and my friends for the patience, support and love they showed, making my academic journey an unforgettable and cherished experience.

I would like to extend my gratitude to Mr. Ioannidis Efstratios and Mr. Doitsidis Eleftherios for their guidance and effort. Without them, I wouldn't have had the opportunity to work on a topic I am keenly interested in and this thesis would not have been up to the desired standards.

Table of Contents

1. Introduction

A production line is a manufacturing system in which the products undergo sequential processing in order to form an end product and fulfill part of the market's demand [2]. This process includes a number of workstations (where processes such as assembly and inspection take place) and intermediate storages (buffers). Numerous optimization methods and algorithms are targeted towards these industrial processes, as small gains in manufacturing costs can lead to great gains in profitability, especially in lines with great production volumes.

In the current era, where production systems utilize a plethora of electronic sensors for all aspects of manufacturing, a large amount of data is available to be used in the understanding and optimization of manufacturing processes. Methods included in the family of Artificial Intelligence (AI), such as Reinforcement Learning (RL), can make effective use of all the data provided, training agents on specific simulation environments resembling their real-world counterparts in order to control parts of an industrial process. The wide application range of those methods allows for great flexibility on industrial control, as the controlling agent forms connections between data and outcomes. Specifically, the PPO algorithm utilizes different policies, which the agent has discovered and optimized during training, in order to match observations to their desired actions.

This thesis aims to examine the ability of a PPO agent to control a simple production line with 2 machines, intermediate buffers and client demand. The agent is tasked with discovering and applying the optimal control policies that balance meeting client demand and the costs of production, storage, and energy consumption. Different, slightly modified scenarios have also been developed and new agents have been trained on them in order to assess the need of training new agents in case variables of the environment such as client demand or energy costs are changed.

2. Literature Review

2.1 Production lines and optimization

2.1.1 Definitions

A production line can be defined as a type of manufacturing system in which products undergo processing, assembly or inspection by a number of workstations in a fixed sequential order [2]. Since the Industrial Revolution and Industry 1.0, when production of goods slowly transitioned from traditional ways like at-home handcrafting of items to more organized structures such as factories, production and assembly lines have had an increasingly significant impact on the way goods are manufactured.

Production lines are broadly implemented for the production of goods across a wide variety of industries, namely semiconductor manufacturing, food industry, automotive industry etc. An example of an engine assembly line can be seen in Figure 2.1.

Figure 2.1: Assembly line of an internal combustion engine. A man is seen inspecting an item on the line [1].

A production line may be categorized by its production volume, dividing it into three major categories:

- Low-volume production: Items are usually large and require longer processing times from each workstation, such as computers and furniture.
- High-volume production: Items are usually smaller and move faster through the workstations, requiring little processing times, such as bottles, nails and metal sheets.
- Continuous flow production: Processing of fluids, such as oil, milk, beer etc [2].

Production lines are systems in which each item moves independently from other parts and are characterized as "asynchronous transfer" systems, allowing for the system to not be fully balanced in order to operate in a smooth manner. Therefore, production lines are commonly used for assembly operations.[3]

Nowadays, information and communication technologies play a major role in the operation of production lines globally, with emerging intelligent and smart technologies such as big data, cyber-physical systems (CPSs), digital twins (DTs) and the (Industrial) Internet of Things (IoT) redefining the existing ways of product manufacturing [4]. The development of informatization itself has been divided by Zhou et al. [5] into three stages:

- Starting from the middle of the 20th century until the middle of the 1990's, informatization was restricted in the digital stage with computing, communications and control applications being the main features.
- From the mid-1990's, the Internet was largely popularized and allowed different devices to interconnect and exchange information.
- In the present day, with the application of technologies such as big data, cloud computing and the Industrial Internet of Things, existing artificial intelligence (AI) models achieve major breakthroughs and constitute the main feature of this stage, the new-generation AI technology.

2.1.2 Optimization

Production lines are widely used across multiple sectors of the manufacturing industry. With the extent of their implementation and the volume of products manufactured, it is obvious that optimization of production lines can significantly improve a company's revenue and provide a competitive advantage. Optimization includes, and is not limited to, balancing of lines in terms of activity duration (Sabadka et al. [6]), reducing energy consumption (Xia et al. [7] and Meike, Pellicciari, Berselli [8]), task and production scheduling (Maccarthy and Liu

[9]) and most importantly for millions of workers globally, maintenance and operator safety (Ding et al. [10] and Vatn and Aven [11]).

As discussed by Sobottka et al. [12], modern industrial planning approaches rely heavily on simulation and simulation-based optimization methods. Though computationally expensive, sustainability and demand for quick planning prove simulation-based methods a valuable solution in the field of industrial production planning and control. As demonstrated in the same paper, the computational time required to solve a planning problem can exponentially increase as the complexity increases.

Since real-life environments can now be simulated digitally, algorithms can be used to assist in the optimization process. The use of optimization algorithms in simulations has been studied by Tekin and Sabuncuoglu [13] in their method review. Tsourveloudis, Doitsidis and Ioannidis [14] implemented an evolutionary work-in-process scheduling fuzzy controller in order to satisfy the random demand for final products of a production system while keeping the inventories of the process at a minimum level. Nguyen, Mei and Zhang [15] examine the advantage of genetic programming and evolutionary algorithms in a complex environment such as production scheduling. In another paper, Li, González and Zhu [16] present a genetic algorithm for optimization of a remanufacturing system in a simulated environment.

Overall, the field of optimization algorithms is a constantly improving sector, fueled by decreasing computational costs and development of new algorithms and methods which pave the way for better-optimized production processes. Figure 2.2 shows the bibliographic map containing the connections between keywords around the term “production line optimization”.

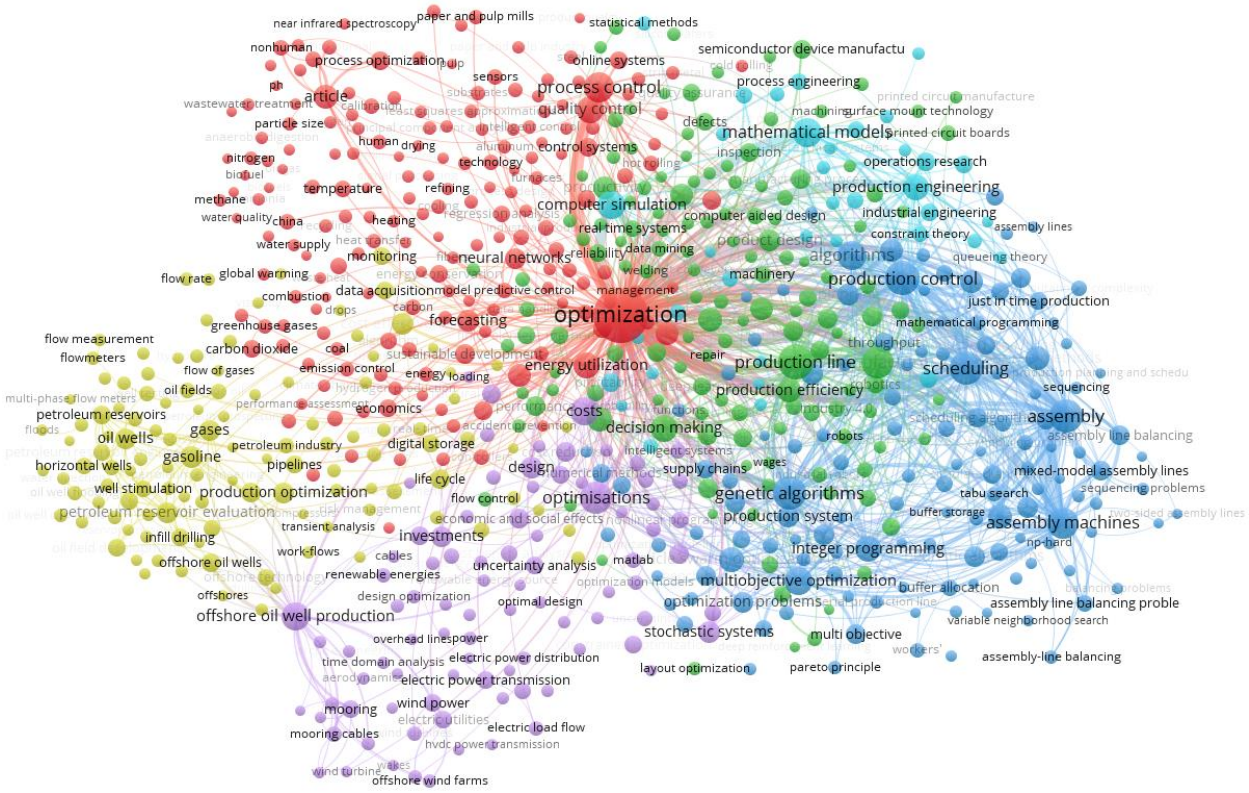

Figure 2.2: Bibliographic map for the term "production line optimization".

2.2 Agent and simulation environment

Agent

In order to correctly describe the simulations and the real-life scenarios an agent finds itself in, some key terms must be defined and taken into consideration. In their book [17], Russell and Norvig concisely define an agent as “anything that can perceive its environment through sensors and acts on it through actuators”. The agent in an AI environment is equivalent to the controller in control theory. Mathematically, the agent function describes the agent’s behavior and maps any sequence of perception to an action.

Numerous studies have demonstrated the efficiency of production and assembly line optimization using agents in simulations. As far back as 1999, Fleury et al. [18] proposed a multi-agent method in order to minimize the negative effects of random events in the manufacturing process. Another multi-agent approach has been presented by Iannino et al. [19] for the self-optimization of steel production in order to overcome hindrances such as

lack of both adaptability of existing techniques and autonomous communication between production elements. Alkhalefah et al. [20] utilize a genetic algorithm to obtain an optimal buffer size for a serial production line, reducing the buffer sizes and therefore the production line cost. Li and Rong [21] test the Ant Colony Optimization (ACO) algorithm on real-world data for optimization of production scheduling, while Chen et al. [22] use the same algorithm (ACO) to solve the problem of producing prefabricated components with limited production capacity. Waschneck et al. [23] present a solution to a production scheduling problem as a Markov Decision Process with Google DeepMind's Deep Q Network, a Deep Reinforcement Learning agent. Neto et al. [24] also proposed deep reinforcement learning as a means of optimizing the maintenance of a scrap-based steel production line with a remarkable improvement in production costs.

With the variety of methods proposed for solving a large volume of different problems, it is concluded that simulation with control agents implementing various algorithms can offer viable and sustainable solutions to an industry despite the complexity of its environment.

Environment

Accurately describing an environment has a major impact on the design and function of the agent that's implemented. The criteria for describing an environment according to Russell and Norvig [17] are briefly presented as follows:

- Observability: An environment can be described as "fully observable" or "partially observable", depending on whether or not the environment sensors provide the agent with full access to the environment condition and variables.
- Level of determinism: If the current state of the environment combined with the agent's current action can fully describe the next state of the environment, then it is regarded as a "deterministic environment", otherwise it is a "non-deterministic environment". Another term to describe the environment is "stochastic" which is used if the probability of a shift of conditions to happen is clearly stated within the environment.
- Stability: An environment can be labeled as "static", "dynamic" or "semidynamic". A dynamic environment may change while the agent is thinking of its next action, while static does not. A semidynamic environment is of particular interest, because although the environment itself might not change, the agent's measure of efficiency or reward might. An example is an environment where nothing changes, but the longer an agent is thinking of its next action and more time passes, the current reward is diminishing.

- Number of agents: An environment can be named “single-agent” or “multiagent” depending on the number of agents acting on it. Though not always clearly distinctive, the number of agents in an environment can be measured by considering whether or not an element of the system modifies its behavior in response to an agent’s actions with the goal of maximizing a reward.
- Continuity: Environments can be described as “discrete” or “continuous”. The description applies to the environment’s state, the way time is considered and the agent’s actions and observations. For example, a card game with no time restrictions can be considered as a discrete environment, while driving or walking outside is considered a continuous environment.

2.3 Reinforcement Learning

Definition

Reinforcement Learning (RL for short) is defined by Wiering and Van Otterlo [25] as a general class of algorithms in the field of machine learning aimed at allowing an agent to learn the optimal behavior in its environment. This discovery of the optimal behavior is aided only by a scalar reward signal as the agent’s feedback. The agent’s goal is to choose and perform actions that maximize the reward signal.

Russell and Norvig [17] categorize RL in model-based and model-free learning. In model-based learning, the agent learns by observing the impact of its actions on the environment and its reward value. In model-free learning, the agent attempts to make a more direct way of behaving. This can be achieved by either action-utility learning or by policy search.

Rewarding

In RL, the agent typically does not have access to the environment’s reward function. This means that the agent has to experiment with different actions and observe the change in its reward value feedback in order to find and exploit different policies [26]. Russell and Norvig [17] state that the use of a reward function in the system has the advantage of being easy to implement while also being simple for someone who does not always know what the optimal action would be to set up. Eschmann [27] examined the design of reward functions in RL by presenting its manifestations in nature through survival, fitness etc.

Bibliographic map

Figure 2.3 shows the bibliographic map for the search query (“production line optimization” AND “reinforcement learning”) on Scopus. It is evident that Reinforcement Learning has a very wide range of applications within the industry, including production control.

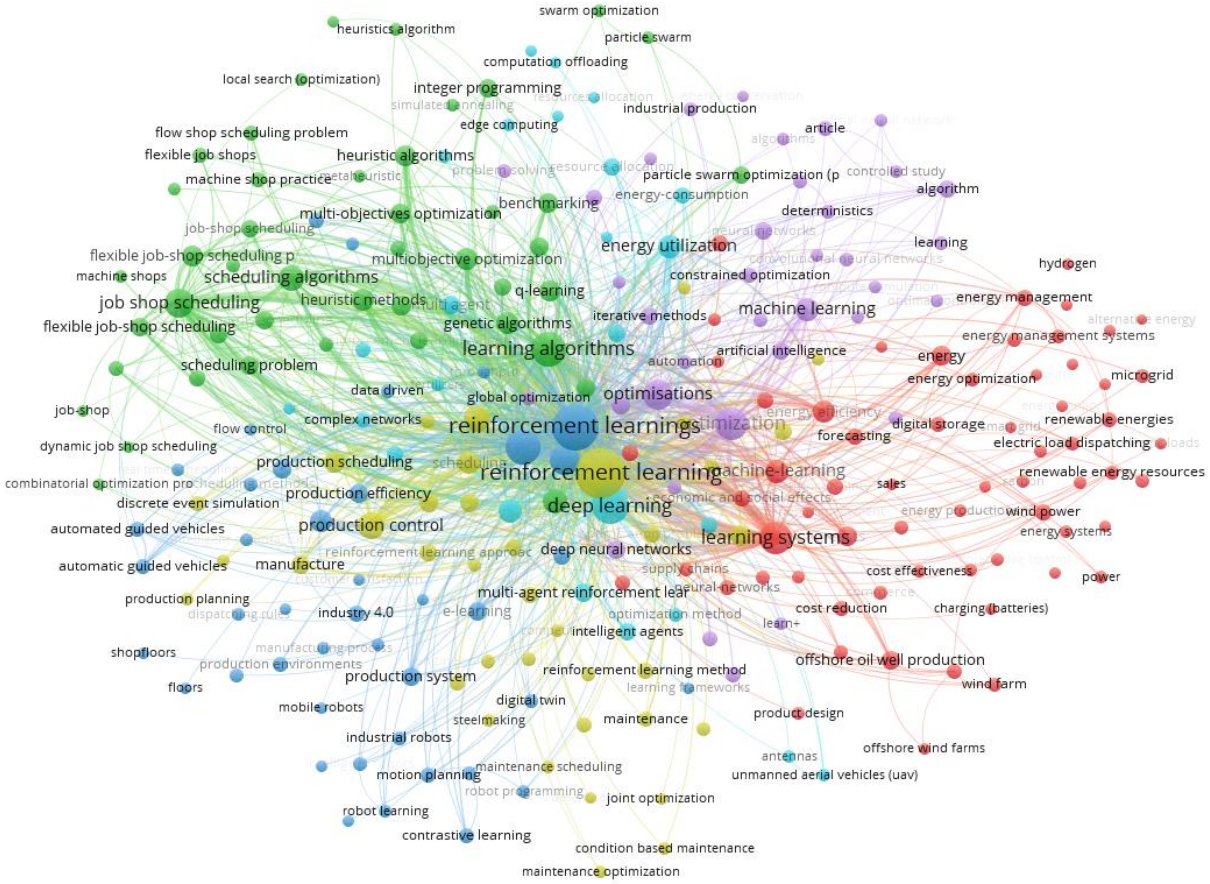

Figure 2.3: Bibliographic map for the "production line optimization" AND “reinforcement learning” query on Scopus.

2.4 Policy agents

Policy search

Policy search is a commonly used strategy for optimization using RL algorithms. Russell and Norvig [17] define a policy as a function linking states (observations) to their appropriate actions. Essentially, the agent “reads” the state observations and decides on the action it should take based on the expected reward of the available policies.

Bagnell and Schneider [28] presented some of the advantages of policy search including the existence of converging policy search algorithms and extensions to multi-agent systems. It is also stated that policy search algorithms have had significant improvements and have been applied in activities such as helicopter control and game-playing. Yahya et al. [29] presented a policy learning parallelization system for autonomous robots so that gained experience can be acquired and shared across multiple agents, enforcing Bagnell and Schneider's [28] arguments. Gabel and Riedmiller [30] utilized policy gradient RL to optimize the policies of agents used for sequential decision-making problems in job-shop scheduling. Esteso et al. [31] presented a literature review encompassing different RL approaches in production planning and control problems. These problems include issues regarding supply and inventory management, truck and route assignment etc. It is evident that RL and agents utilizing policy methods have a wide range of applications and have been proven useful in tackling industrial issues in simulations.

Proximal Policy Optimization (PPO) algorithm

Schulman et al. [32] proposed the Proximal Policy Optimization family of algorithms in order to tackle issues of pre-existing policy optimization algorithms such as implementation complexity. The PPO algorithm was tested across different domains, such as humanoid running and steering simulation and on the Arcade Learning Environment [33], which was introduced by Bellemare et al. and is used to benchmark domain-independent AI technologies. In the benchmarks presented, the PPO algorithm had promising results, even outperforming other algorithms it competed against.

Actor/Critic method

The actor-critic method has proven its usefulness as a framework, leading to a good learning performance. In this method, two different networks are used: the actor model and the critic model. The actor learns to match current observations from the environment to the desirable action (essentially, the policy to be applied). This action is then applied to the simulation and a reward is received from the environment, which is then used by the critic model. The critic evaluates if the performed action led the environment in a better state than it was before and this feedback is passed to the actor model to optimize its policy. By comparing the utility of its current policy to the previous one, the agent can make changes to its current policies or explore other options.[34]

3. Production Line Design

3.1 Design philosophy of production lines

3.1.1 Production line behavior

As mentioned earlier, production lines in general are used in the manufacturing of high volumes of products with restricted customizability and flexibility. Part transfer in production lines is asynchronous, which means that the parts are transferred independently of each other. The main problems a production line may face are blocking and starvation of parts, both of which stem from finite buffer capacities and unpredictability of machine behaviors, such as breakdowns. The term “starvation” is used to describe a machine not having enough input or raw materials to work with and therefore not being able to operate. On the other hand, “blocking” means that the buffer placed right after the machine under consideration has been filled and can’t store any more of its output [3].

A simple production line can be described using a serial server-buffer queuing network. An example of a queuing network with four servers and four buffers can be seen in Figure 3.1.

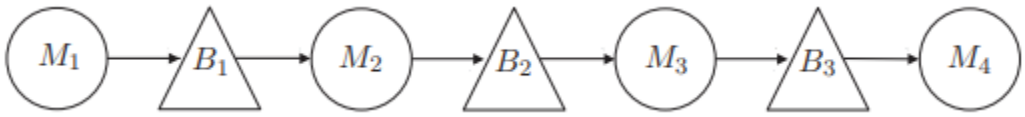

Figure 3.1: Example of a server-buffer queuing network [2].

3.1.2 Buffers

In a production line, buffers receive and store items from upstream machines and provide them to downstream machines. Their main purpose is to divide the production line in smaller segments so that a possible machine failure will pose a smaller threat on the stability of the system as a whole [35].

In a serial production line, malfunction of one machine may cause issues in the whole line. In an extreme scenario of a production line with a number of machines directly connected

to each other with no intermediate buffers, malfunction of one machine can lead to the entire line to be shut down until the issue is resolved [36].

Buffers can be represented as separate queues consisting of the separate items residing in them. The two main queuing types for buffers in terms of time of arrival and exit priority are First in-First out (FIFO) and Last in-First out (LIFO).

- FIFO queuing policy prioritizes the item with the earliest arrival time of all the available items in the buffer for exiting the buffer. In short, the first item that arrives in the buffer is also the first one to leave it.
- LIFO queuing policy prioritizes the last item entering the buffer. This means that the last item arriving in the buffer will be the first one to exit.

In this thesis, FIFO has been chosen as the queuing policy for the buffers in the production line.

3.2 Environment description and main objective

The main objective of this thesis is to assess the ability of an RL PPO agent to efficiently control a small production line, as well as its ability to adapt to changes in environment variables.

As raw materials enter the system, they undergo sequential processing by the machines and are stored in buffers until they are either processed in the next stage or used to fulfill a client order. The efficiency of the current control policy is measured using an objective function which takes into consideration parameters such as buffer levels, pending client orders, energy consumption etc.

3.2.1 Choice of environment

Simulation takes place in MATLAB's Simulink environment, core component of which is the SimEvents discrete-event simulation engine and component library. SimEvents allows the creation of discrete-event models, such as production lines, using entity generators, queues, graphical displays, signal probing and other useful components.

The main advantages of SimEvents are its simple graphical interface, the customizability of its components and its seamless integration within the MATLAB ecosystem.

- Graphical interface: Every component in the SimEvents library is graphically depicted and able to be freely moved, rotated, dragged and dropped within the workspace. This allows for both faster and easier construction of comprehensible models.
- High customizability: The components in the SimEvents library have highly customizable properties, such as, but not limited to, variable entity generation times, programmable entity attributes and selective signal logging and probing.
- Seamless integration: SimEvents includes components that can be programmable for specific tasks and function blocks which support MATLAB code. Furthermore, SimEvents models can input data from and output data to the base MATLAB workspace, thus making it simple to save, load, process or plot any available signal outside the Simulink environment.

3.2.2 SimEvents component overview

Entity generation and handling

The five basic components used in this system for entity generation, storage, flow, processing and termination are shown in Figure 3.2:

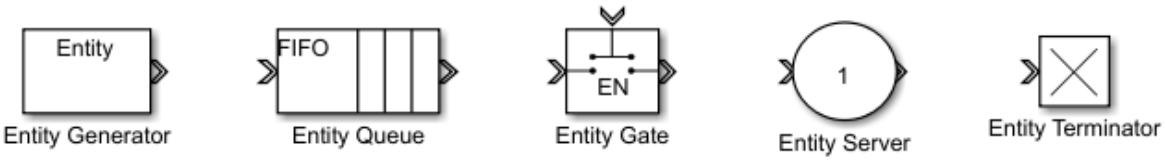

Figure 3.2: Components used for entity handling.

- Entity generator: Generates an entity after a specified time period has elapsed. Entity attributes can be added and modified using scripts.

- Entity queue: Accumulates incoming entities and releases them when conditions allow it. FIFO means that exit priority is given to the earliest arriving entity available in the queue.

- Entity gate: Allows a single entity to flow from the input port to the subsequent component when an appropriate message is received.

- Entity server: Processes one entity at a time. Processing time can be specified through the entity's own attributes, environment signals or a MATLAB script.

- Entity terminator: Terminates an incoming entity and its attributes from the environment.

Discrete-event charts

Discrete-event charts are a core component of discrete-event simulation. These blocks can be used to receive, process and send messages within the Simulink environment. They provide graphical state transitions and execute in an event-based rather than time-based fashion [37]. An example of a discrete-event chart block with one input and one output is shown in Figure 3.3:

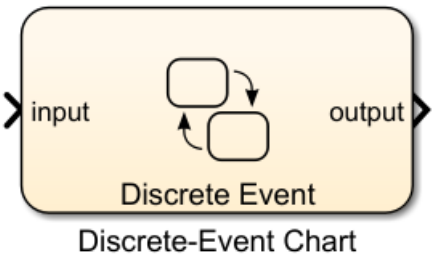

Figure 3.3: Discrete-event chart in the Simulink environment.

Inside the discrete-event chart block, the user can define states and transitions between them. Upon the arrival of a message at the chart input, the discrete-event chart responds in the following manner:

- If the chart is waiting for an incoming message, it wakes up and makes possible transitions. If multiple messages are queued, those with the highest priority are processed first.
- If the incoming message does not require a response, the chart does not wake up and the message is queued [37].

An example of the states and transitions inside a discrete-event chart can be seen in Figure 3.4:

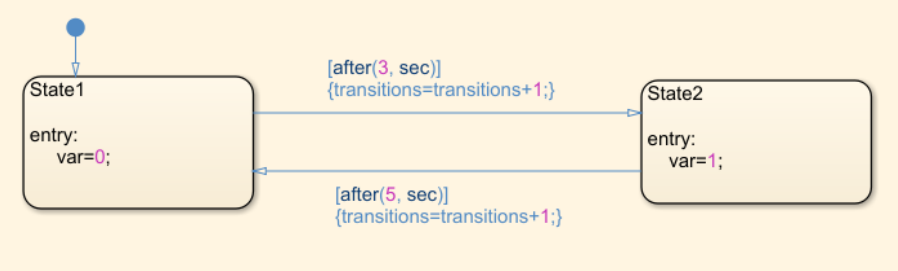

Figure 3.4: Example of a discrete-event chart and its transitions.

The example chart consists of two possible states with no input or output. The blue arrow above the "State1" box indicates the initial state of the chart. Upon entering the state, the chart changes the value of the "var" variable to 0. After 5 seconds, it transitions to "State2" where the value of "var" changes to 1 and increases the value of the "transitions" variable by 1. After 3 seconds in "State2", the chart switches to "State1" and increases the "transitions" variable by 1 unit.

Signal logging and transmission

In Simulink and SimEvents, signals can be generated by statistical measuring of components, function blocks or discrete-event chart output. Component statistics vary between different components and are used to measure metrics such as the average intergeneration time between entity generations, entity queue levels, machine utilization etc. The four main components used in this system for signal handling are shown in Figure 3.5:

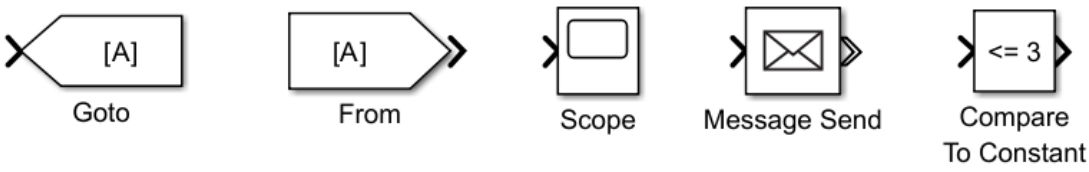

Figure 3.5: Signal transmission and logging components.

- "Goto" component: Transmits a signal throughout the Simulink environment.
- "From" component: Receives a signal transmitted inside the environment.
- Scope: Used to plot the incoming signal in respect to the simulation time.

- Message send: Sends a message when input is received.
- Compare to constant: Compares the block input to a constant. If the result of the comparison is “True”, the output will be “1”. Otherwise, the output will be “0”.

3.3 Production line implementation

3.3.1 Production line initialization

The production process starts with the generation of entities as raw materials by an entity generator. Entities are assigned a random processing time for each of the production stages, as well as a unique product number using a custom script inside the “Unprocessed Product Generator” block. The corresponding code can be seen in Figure 3.6. Product numbers start at 1 and are increased by 1 for every generation, so that every entity has its own product number. Two separate processing times for each stage are randomly drawn from the exponential distribution, (see Appendix A: Probability distributions for more details) with a mean value of μ_1 and μ_2 seconds respectively.

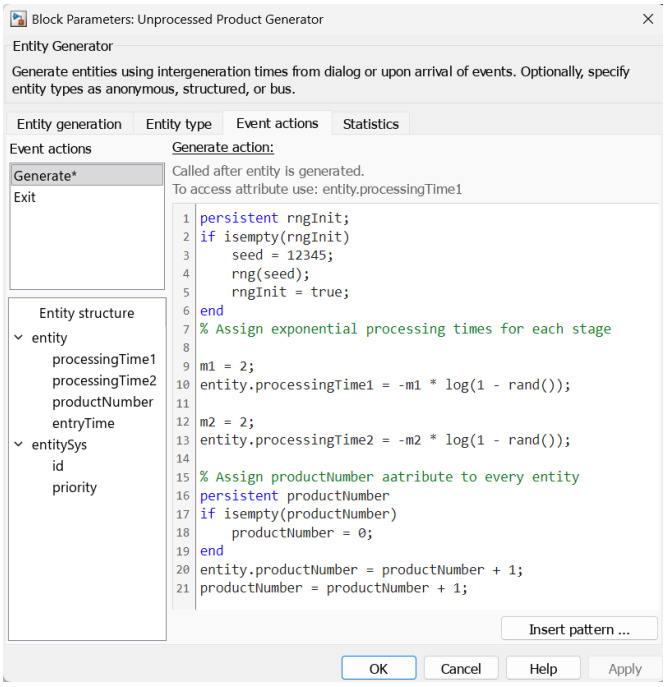

Figure 3.6: Raw materials generator script.

Raw materials are produced in excess and are stored in a FIFO buffer placed directly after the generator. The buffer level signal is then transmitted into the environment and probed using a scope, as can be seen in Figure 3.7.

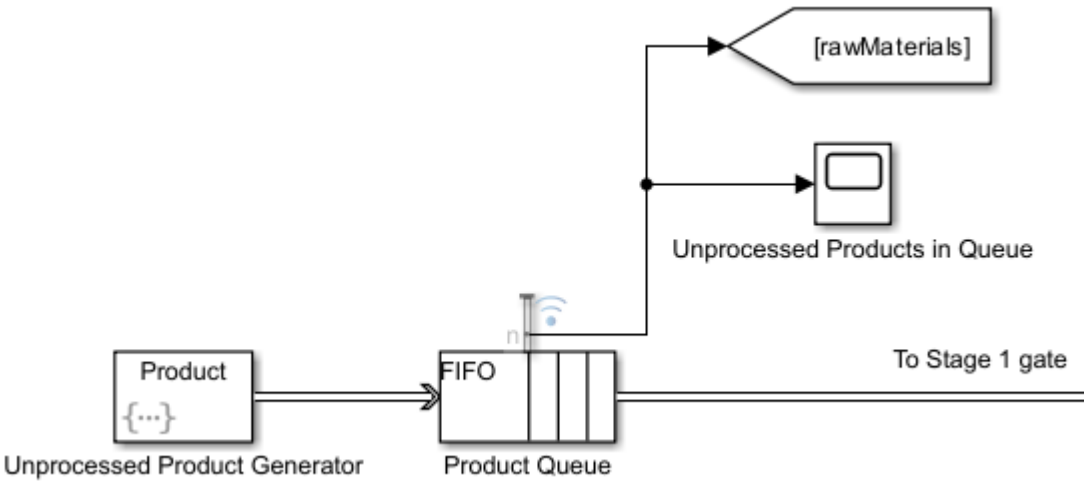

Figure 3.7: Generation of raw materials.

3.3.2 Machine states

Machine states are simulated using discrete-event charts. The four possible states used in this model are:

1. OFF: Machine is switched off and is not processing any entities.
2. SETUP: Machine is transitioning from OFF state to either IDLE or OPERATION states. SETUP state lasts for a standard amount of time, during which the machine is not processing any entities.
3. IDLE: Machine is switched on and not processing any entities but is ready to do so at any time.
4. OPERATION: Machine is switched on and is processing an entity.

Each machine state has its own “status value” and “energy consumption coefficient”, depending on its current state. Assigned status and energy consumption values have been set as following:

- OFF state: Status=0 and Energy consumption=0
- SETUP state: Status=3 and Energy consumption C_3. Setup also takes a standard amount of time for both machines in this model.
- OPERATION state: Status=2 and Energy consumption C_2.
- IDLE state: Status=1 and Energy consumption C_1.

Machine state transitions are directly governed by the model's RL agent. An incoming environment signal is used as input in the discrete-event chart and depending on its value, which is an integer ranging from 0 to 2, the machine switches to the corresponding state according to the state values explained above. Figure 3.8 shows the complete discrete-event chart used for both machines in this model. For machine 2, the indices of the variables are changed from "VariableName1" to "VariableName2":

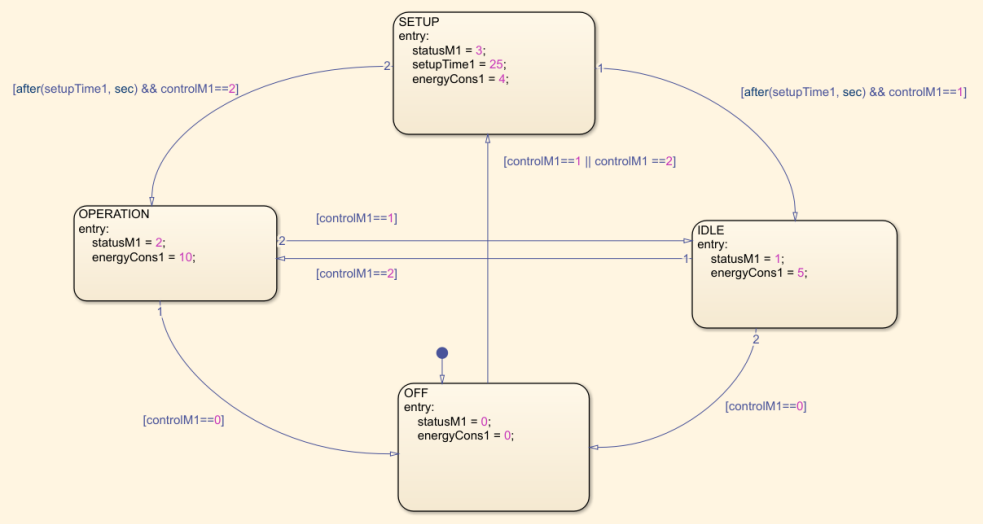

Figure 3.8: Discrete-event chart used for both machines in this model.

For the machine to transition from its OFF state to SETUP, it must receive either a "1" or "2" input. Upon switching to SETUP and after the setup time has elapsed, the machine will either transition to IDLE or OPERATION according to its input. At the beginning of each simulation, machines start in OFF state by default. The output of the discrete-even chart is two separate signals: its current state value and its energy consumption.

Machine state chart implementation

Entity processing takes place in the "Entity server" component of the model. The machine states chart is responsible for enabling or disabling the entity gates that allow an item to move from its current queue to the machine for processing.

The input signal is transmitted directly from the RL agent of the environment and is responsible for the state transitions of the chart. The output signal of the chart ranges from 0 to 3 for the machine state value and receives a value from the set {0, C_1, C_2, C_3} for the machine energy consumption coefficient, depending on the current machine state. If the signal value is equal to 2, and therefore the machine is operational, a message is transmitted to the entity gate and an entity passes through for processing.

Probing the individual input and output signals of the discrete-event charts allows for a better understanding of the model's behavior and therefore scopes are added in each signal transmission line.

The configuration of machine 1 in the model is shown in Figure 3.9.

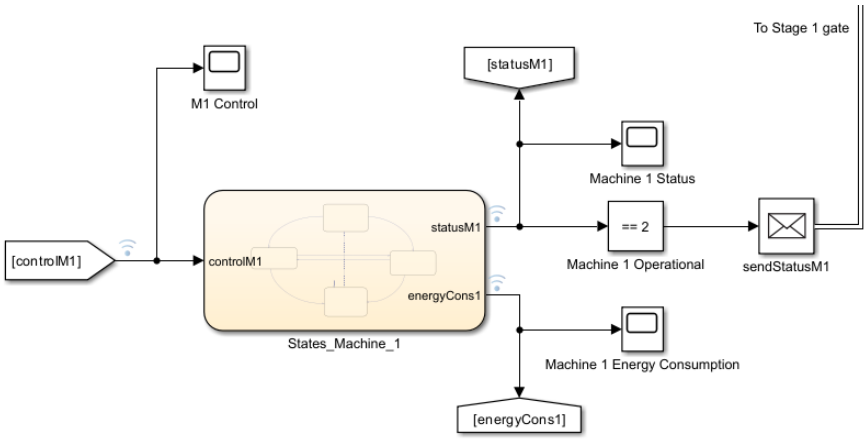

Figure 3.9: Implementation of the machine control chart for Machine 1.

3.3.3 First processing stage

Accumulated raw materials are available for processing in Stage 1. When the discrete-event chart for Machine 1 changes its status to 2 (OPERATION), a message is sent to the Stage 1 entity gate and one entity from the raw materials buffer is allowed to move to the first processing stage, where it will undergo processing for a time period equal to its *entity.processingTime1* attribute, which is assigned as shown in 3.3.1. Upon completion of its processing, the product exits the entity server and is stored in a subsequent FIFO entity

queue, which serves as the buffer for Stage 1 (B1), and if the gate is still open, the next entity from the raw materials buffer proceeds to Stage 1. The buffer level is then transmitted to the environment. Both the utilization of Stage 1's entity server and its buffer level are probed using a scope. The configuration can be seen in Figure 3.10.

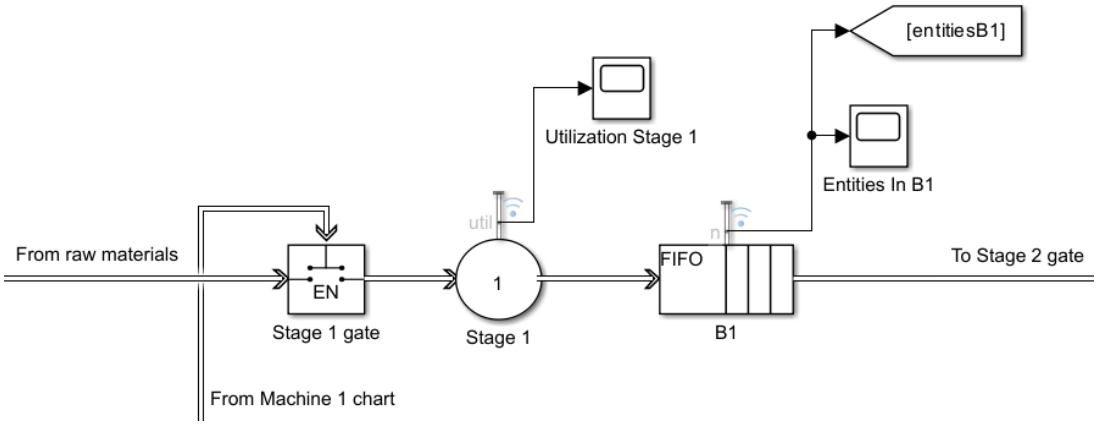

Figure 3.10: First processing stage.

3.3.4 Second processing stage

Machine 2 event chart and Stage 2 processing follow the same principles as described in 3.3.2 and 3.3.3 respectively and are shown in Figure 3.11 and Figure 3.12.

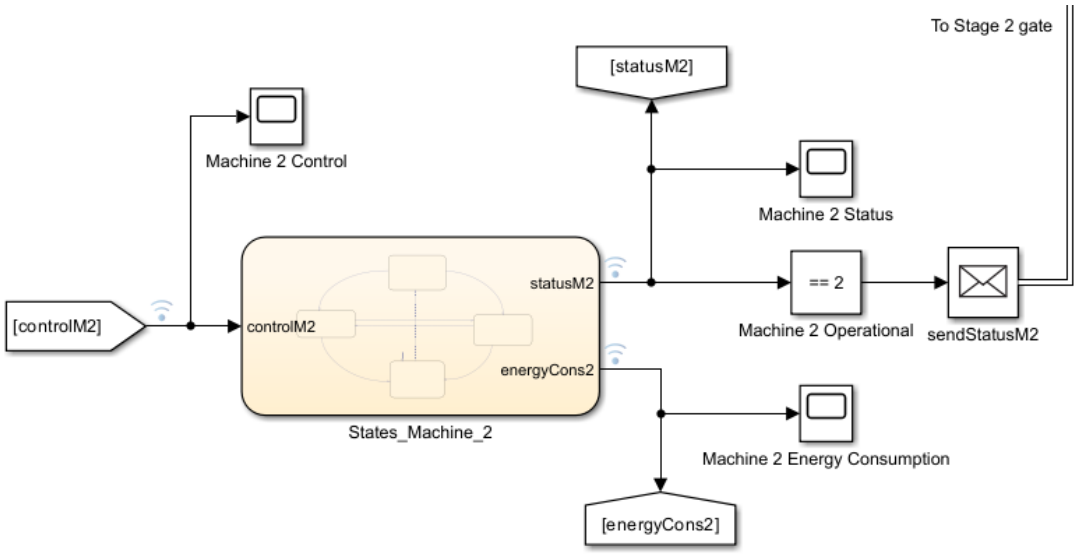

Figure 3.11: : Implementation of the machine control chart for Machine 2.

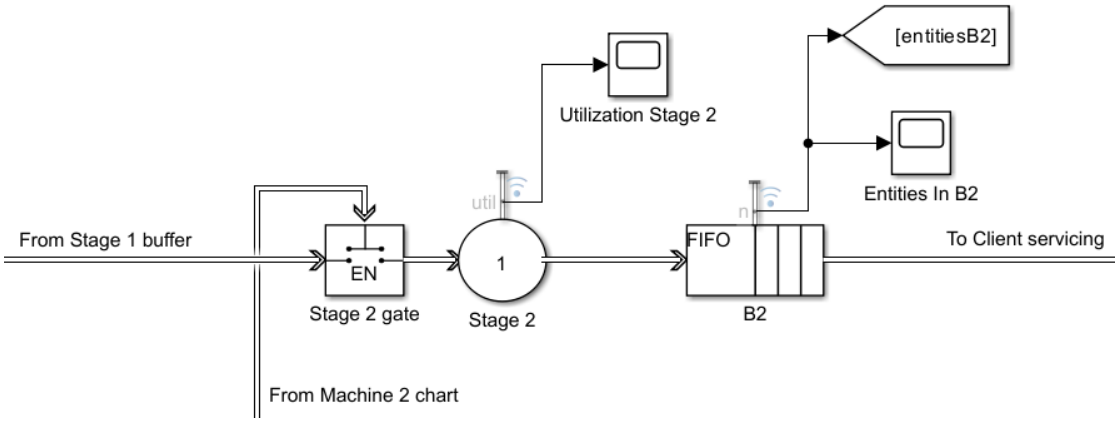

Figure 3.12: Second processing stage.

When the discrete-event chart of Machine 2 changes its status to 2 (OPERATION), the Stage 2 gate allows one of the accumulated entities of the Stage 1 buffer (B1) to move for processing in Stage 2. Processing time in this stage is dictated by the entity's *entity.processingTime2* attribute. After it has been processed, the entity has completed its production cycle and is stored in the Stage 2 buffer (B2). As in Stage 1, Stage 2 utilization and its relevant buffer are being probed using a scope, with the buffer's level transmitted to the environment.

3.3.5 Client generation

Initial generation and acceptance

Clients are initially generated using an entity generator component with a time period of 1 second. This means that every 1 second, one client is generated. Clients are then accumulated in a subsequent FIFO buffer and wait until they are accepted in the system's client queue. Each new generated client is assigned a unique client number, which starts from 1 for the first client and is increased by 1 for every new generation. Clients are also assigned with an *"entryTime"* attribute used by a subsequent entity server in order to simulate different arrival times. The attribute follows the uniform distribution (equal chances of a number appearing across the whole range of the distribution) with a mean time of 4 for the Baseline scenario. The code of the client generator for the attribute customization can be seen in Figure 3.13.

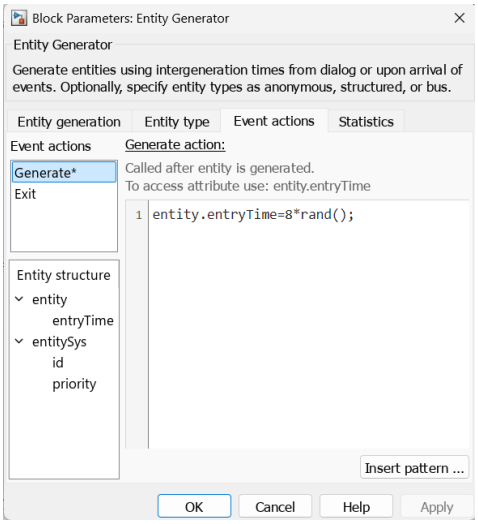

Figure 3.13: Entity generator block code.

The *rand()* function randomly selects number in the space [0,1] following the uniform distribution (refer to Appendix A: Probability distributions for more details). The mean of this distribution is μ=0.5. Therefore, in order to get a mean time of μ', the randomly selected numbers are multiplied by 2*μ', extending the space [0,1] to [0,2μ'] and setting the new mean time to μ'. The selected number is then used as the client's *"entryTime"* attribute.

In case the client accumulator is full, client generation halts and resumes when a spot in the buffer has been released. The amount of clients waiting for entrance in the system is measured using a scope, as seen in Figure 3.14. Accumulated clients are then passed to the system's client queue by an entity server, which serves the entities for as long as their assigned *entryTime* attribute, simulating random arrival times.

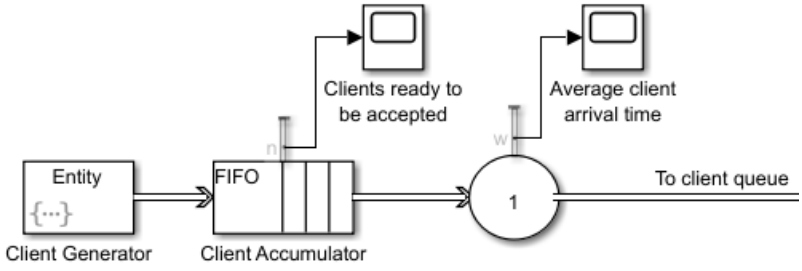

Figure 3.14: Initial client generation system.

System client queue

After passing from the generation system, clients enter the system queue. Clients in the system queue are considered as "pending orders" that wait to be serviced. The client queue array can be seen in Figure 3.15. Statistics of interest of this queue are the average queue length, the average waiting time for the clients and the amount of clients waiting. The length of the client queue is also transmitted to the environment.

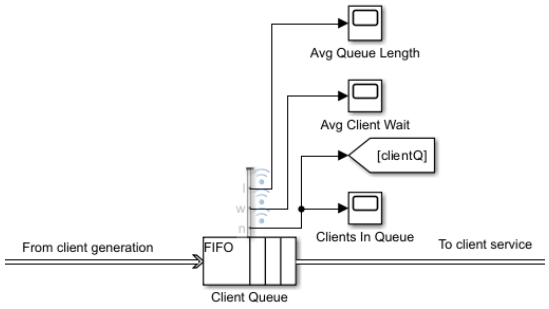

Figure 3.15: Client queue array.

3.3.6 Client servicing and termination

In order for the client servicing process to take place, both of the following conditions must be true:

1. A client is waiting in the system's client queue.
2. An entity exists in Stage 2's buffer (B2).

If both conditions are met, a "Composite entity creator" block removes one item from the B2 buffer and one client from the client queue, thus completing the client's request. Each client only requests one item from the system. After the request has been completed, the client is then terminated from the system. The total amount of clients serviced is being probed from the "client terminator" block, as seen in Figure 3.16.

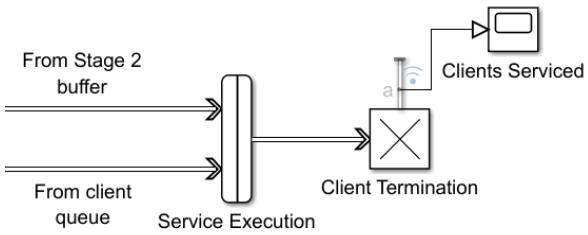

Figure 3.16: Client servicing and termination subsystem

3.3.7 Complete production line configuration

The production line in a compact form is presented in Figure 3.17.

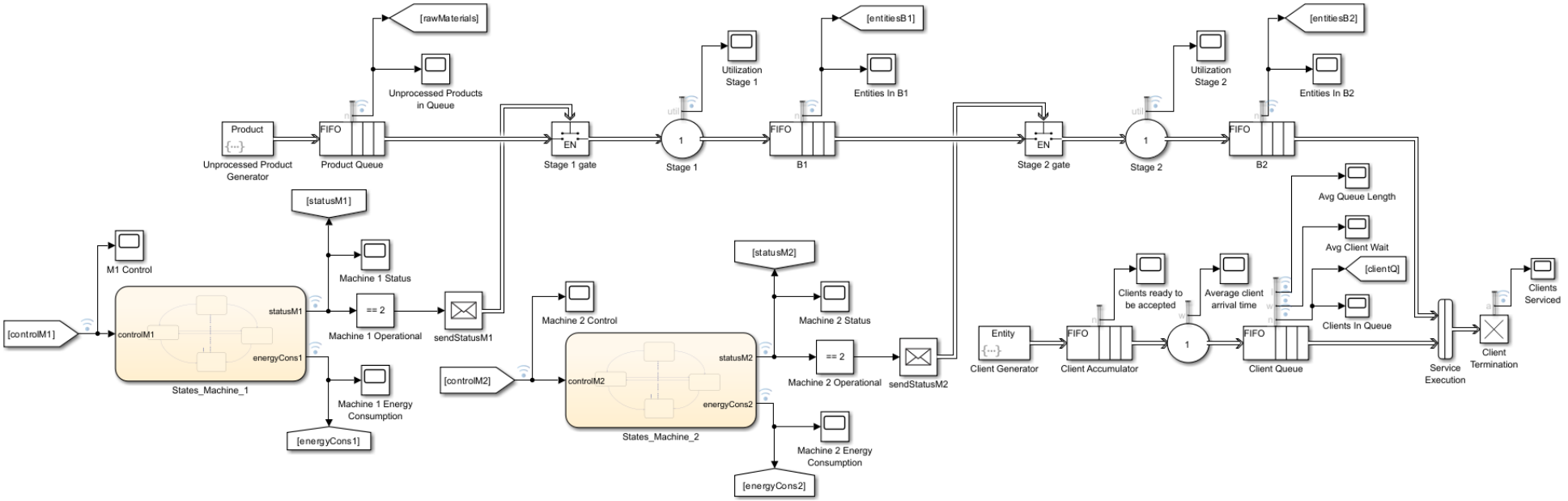

Figure 3.17: Complete system layout.

3.4 Agent subsystem

Inside the same environment as the production line model, the control module is placed. It consists of the objective function calculation module, the observation module, the RL agent block and the agent's output transformer module.

3.4.1 Objective function

Target of the objective function

In order to assess the performance of the production line during its simulation cycle, certain variables need to be measured and taken into consideration. Depending on the goal of the optimization taking place, each of these variables is assigned its own coefficient. These variables, each paired with its own coefficient, are then summed, forming the objective function. In this scenario, storage of entities in buffers, operation of the machines and pending clients all are treated as negative effects in the process. However, since the agent is by default attempting to maximize the targeted objective function and all the measured variables are positive integers, the sign of the function is reversed, prompting the agent to minimize a negative value.

Objective function implementation

The objective function is implemented in the environment using a MATLAB function block. The environment signals important enough for the optimization process are:

- entitiesB1: The amount of entities present in Stage 1's output buffer (B1), waiting to be processed in the second stage of the line. It is important that the buffer is neither empty nor full to ensure that both Stage 1 and Stage 2 can operate without being blocked or starved.
- entitiesB2: The amount of entities present in Stage 2's output buffer (B2), ready to be picked up by clients. An empty buffer for Stage 2 means that clients will be waiting in the queue, while a completely full buffer will cause Stage 2 to halt its production.
- clientQ: The amount of clients waiting in the system's client queue. Pending orders affect the system negatively, as they populate the system and in real-life scenarios may cause critique for the business and lead to loss of revenue.

- energyCons1: Energy consumption in Stage 1, as dictated by the stage's discrete-event chart.
- energyCons2: Energy consumption in Stage 2, as dictated by the stage's discrete-event chart.
- rawMaterials: The amount of raw materials in the system's initial buffer. Taking into account the accumulation of raw materials will assist the agent in its early training stages to commence the production process.

In order to create the objective function, a MATLAB script has been used The weight coefficients for the input signals have been carefully balanced through trial and error during the earlier stages of the model's development. The output of this function is the calculated reward value. The MATLAB script, as well as the module in the simulation environment, can be seen in Figure 3.18 and Figure 3.19 respectively.

```
function reward = calcReward(entitiesB1, finishedProd, clientQ,...
                             energyCons1, energyCons2,rawMaterials)

% Set weight coefficients for the reward function
bufferCost=0.01;
readyproductsCost=0.015;
clientwaitCost=0.65;
energyCost=0.1;
rawMaterialCost=0.0025;

% Calculate the total cost and the reward function
reward=-(entitiesB1*bufferCost + finishedProd*readyproductsCost +...
    clientQ*clientwaitCost+(energyCons1+energyCons2)*energyCost+...
    rawMaterialCost*rawMaterials);

return;

end
```

Figure 3.18: MATLAB script used for the calculation of the objective function.

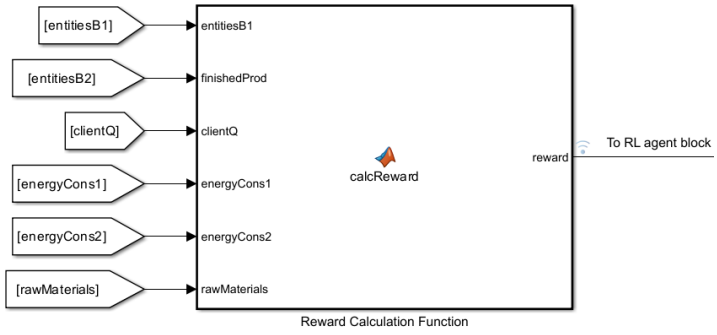

Figure 3.19: Reward calculation component.

3.4.2 Observation collection

Observations vector

The RL agent requires a vector with all the environment variables the user wants it to take into account, referred to as the "observations" vector. In order to create that vector, another MATLAB function block is used in the model.

Data collection and function implementation

The data needed for the function are transmitted in the environment by the model. For the observation function, the important variables are the length of the client queue, the buffer levels for both stages and raw materials and the states of both machines in the system. Initial inputs of these variables have been set using a "Unit Delay" block to 0 to represent the machines being turned off in the first time step of the simulation.

The observations function utilizes a simple MATLAB script to normalize all buffer levels in respect to their maximum capacities and create an output vector including those levels and the states of each machine. Implementation of this concept is done in a similar manner to the reward calculation function and is shown in Figure 3.20 and Figure 3.21.

```
function data = observations(rawMaterials, entitiesB1,...
                             entitiesB2, clientQ, statusM1, statusM2)

    %Define maximum buffer capacity
    maxBufferCapacity = 50;
    rawMaterialsCapacity=500;

    %Normalize buffer levels
    rawMaterialsLevel = rawMaterials/rawMaterialsCapacity;
    bufferLevels = [entitiesB1;entitiesB2] / maxBufferCapacity;

    %Normalize client buffer level
    clientLevel =  clientQ / maxBufferCapacity;

    %Create machine states vector
    machineStates = [statusM1; statusM2];

    %Combine normalized observations into a single vector
    data = [rawMaterialsLevel; bufferLevels; clientLevel; machineStates];

end
```

Figure 3.20: Observations function script.

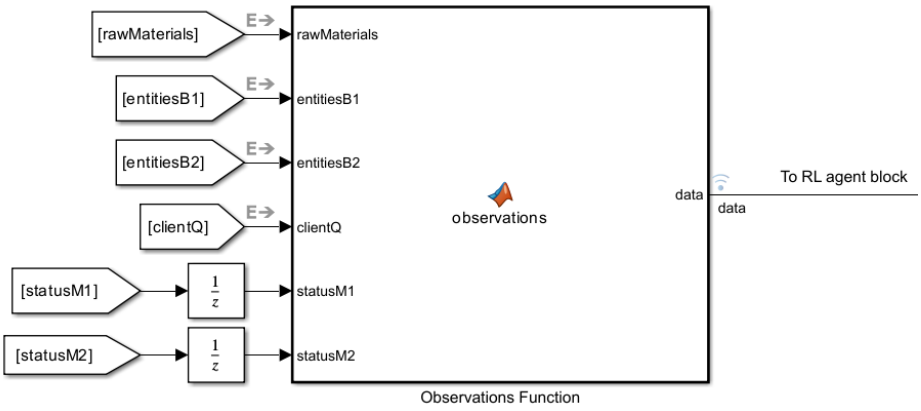

Figure 3.21: Observations function component.

3.4.3 Reinforcement Learning agent block

In order to implement an RL agent in the system, the RL agent block component is required. The block's inputs are the observations vector as described in 3.4.2, the reward value as described in 3.4.1 and a third logical input which terminates an episode simulation when true. Since such action is not required in this system, it has been connected to a "constant" block whose output is "0" at all times. Implementation of the RL agent block in the system can be seen in Figure 3.22.

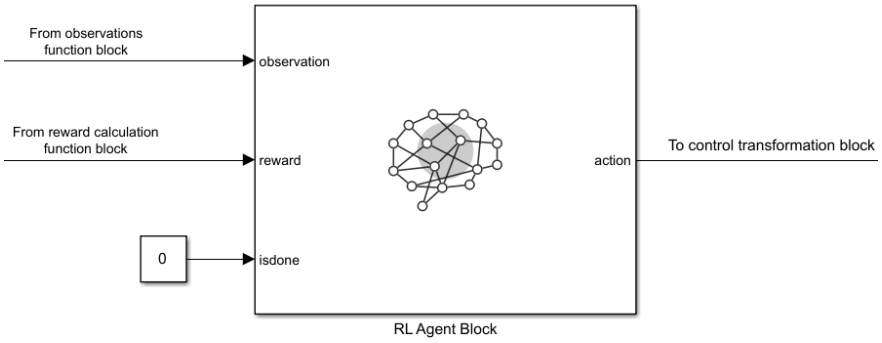

Figure 3.22: Implementation of the RL agent block.

A comprehensive diagram of how the agent block communicates with the environment can be found in Simulink and is shown in Figure 3.23. In this system, "observation", "reward" and "isdone" inputs are provided by the observations and the reward function, as well as the "constant" block that has been set to 0. The "action" output of the agent block is handled by the "Control transformation" function used in the next step.

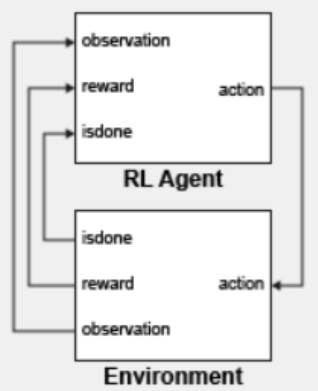

Figure 3.23: Agent-Environment interaction.

3.4.4 Control transformation function

The output of the RL agent block is a 2x1 vector containing the desired state of each machine in the next step. In order to decode that vector and feed it to the machines, it is necessary to split the vector in a single integer for the first machine and a single integer for the second machine. To achieve this, a function block has been placed after the agent block containing a simple MATLAB script which splits the vector and can be seen in Figure 3.24. The first and second elements of the "action" vector are the chosen states for Stage 1 and Stage 2 respectively.

```
1   function [controlM1, controlM2] = transformControl(action)
2
3   controlM1 = action(1);  %Control for M1
4   controlM2 = action(2);  %Control for M2
5
6   end
```

Figure 3.24: Action transformation function.

The two outputs are then transmitted into the environment as "controlM1" and "controlM2" and are probed using scopes. In order to create the desired initial state for the machines at the start of the simulation, two "Unit delay" blocks have been placed before the signal transmission and their initial conditions are set to "0". The complete array for the signal transformation and transmission can be seen in Figure 3.25.

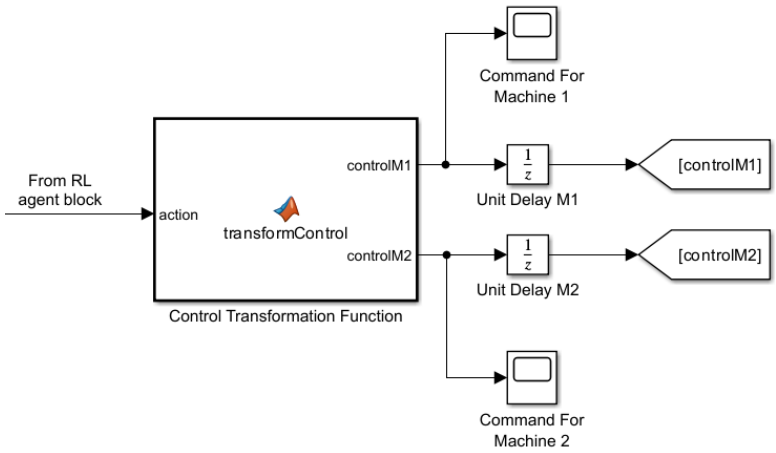

Figure 3.25: Agent output transformation and transmission.

3.4.5 Complete configuration of the agent integration subsystem

The complete depiction of the agent subsystem (reward and observation functions, agent block and output transformation blocks) can be seen in Figure 3.26. As the transmission and reception of the signals between the subsystem and the main production line is achieved through "From" and "Goto" blocks, their placement relative to each other makes no difference in their interaction.

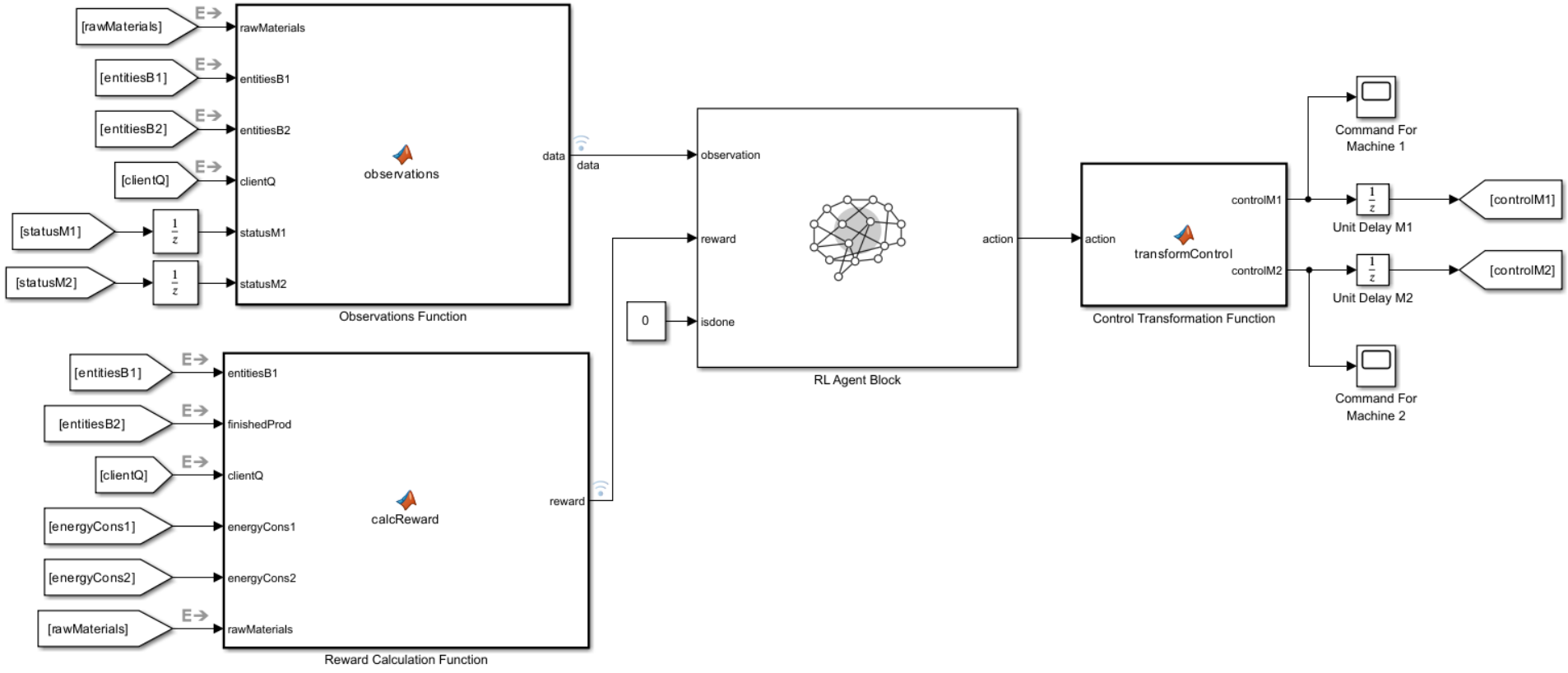

Figure 3.26: Agent subsystem.

4. Reinforcement Learning agent implementation

In order to create and train the agents to control the production line, a MATLAB script was used for each agent. The scripts utilize functions from the basic MATLAB installation as well as functions from the Parallel Computing and Reinforcement Learning toolboxes to execute the training process and to create parallel workers to speed it up if necessary.

4.1 Simulink-Base workspace interaction

As explained earlier, the MATLAB base workspace and the Simulink environment are fully compatible with each other and support data importing and exporting. In this thesis, MATLAB scripts are used to train the RL agents. In order to achieve this, the training script uses the location of the agent block in each scenario as input and interacts with the simulation environment through it. The Simulink environment automatically outputs the logged data from the simulation to the base workspace once the simulation has concluded.

4.2 Choice of scenarios

The basic system that has been created is considered the "baseline scenario" for this analysis. In most real-life scenarios, variables like client influx, processing times and energy costs can change drastically, thus impacting the function of a production line and requiring adaptation in order to maintain the optimal point of operation. In this thesis, a number of different scenarios are examined and are designed as explained below:

- High demand scenario: Influx of clients in the system has been increased.
- High demand v2: On top of increasing the client influx, pending orders now contribute a much bigger penalty to the agent's reward function.
- Slower Production: Processing times for both stages have been increased.
- Energy cost increase: Cost coefficient of the machines' energy consumption in the objective function has been increased.

For each scenario, a different agent has been trained. The goal of having different scenarios is to examine how each change affects the behavior of an agent, thus rendering it necessary

or not to have different agents trained for different scenarios. The 5 different agents will be tested through all 5 scenarios, including the baseline one, and compared side-to-side.

4.3 Training script

4.3.1 Initialization

The first step in the training script is to initialize the environment. First, the file name and the block name of the RL agent block are specified, as seen in Figure 4.1.

```
%% Initialization
module = 'prodline_Baseline';%Environment loading
agentBlock = [module '/RL Agent Block'];%Agent block
```

Figure 4.1: Initialization.

Observations

Now that the agent block has been specified, the observation specifications need to be set. The *rlNumericSpec* function will be used to store the specifications of the observations vector. The desired specifications are the buffer levels, the client queue level and the machine states, as stored in the observations vector. The arguments of this function are the dimensions of the vector and the lower and upper limits for each of its elements and its output is stored in a variable named “observationInfo”. Though not restricted to storing just that, the variable has been named “Machine states”.

Actions

After setting the observation parameters, it is time for the action parameters to be set. The possible states the agent can choose from are 0 (OFF), 1 (IDLE) or 2 (OPERATION) for each machine. The SETUP state is a transitional state in the chart and the agent cannot choose it directly. This means that the possible actions the agent can choose from are all the possible combinations of the numbers 0, 1 and 2 in 2-dimensional vectors which can be seen in Figure 4.2. Each row of the matrix represents a possible action of the agent and each column represents the states of stages 1 and 2 in a specific action.

	1	2
1	0	0
2	1	0
3	2	0
4	0	1
5	1	1
6	2	1
7	0	2
8	1	2
9	2	2

Figure 4.2: Possible combinations of numbers 0, 1 and 2.

The matrix is then converted to a cell data structure and passed to the *rlFiniteSetSpec* function to create the variable for storing the specifications of the action space, which is named “Machine actions”. The reason this function is not the same as the one used in the observations is that observations are continuous values ranging from 0 to 1 (except for the machine states which are integers), but the action space has a limited amount of elements and no available values in between them.

Environment object

Finally, the module, the agent block and the observation and action information are used to create a Simulink environment object using the *rlSimulinkEnv* function. The corresponding script can be seen in Figure 4.3.

```
%Possible machine states
observationInfo = rlNumericSpec([6,1], ...
    'LowerLimit', [0;0;0;0;0;0],...
    'UpperLimit', [1;1;1;1;3;3]);
observationInfo.Name = 'Machine states';

%Creation of the possible states matrix
actM1=[0,1,2];
actM2=actM1;
actionMatrix = combvec(actM1,actM2)';

%Splits actionMatrix into 16 different cell arrays with dimension 2
%each. Then it passes them as possible agent states
actionInfo = rlFiniteSetSpec(mat2cell(actionMatrix, ones(1,9), 2));
actionInfo.Name = 'Machine actions';

env = rlSimulinkEnv(module,agentBlock, observationInfo, actionInfo);
```

Figure 4.3: Observation and action information and environment object creation.

4.3.2 Creation of the actor and critic networks

Actor network

The actor network consists of a feature input layer, two sets of alternating fully connected layers and rectified linear unit layers, another fully connected layer containing the weights and a final softmax layer. The function of each layer is explained below:

- Feature input layer: Inputs feature data to a neural network and applies data normalization.
- Fully connected layer: Multiplies the input by a weight matrix and adds a bias vector, if any.
- Rectified linear unit layer (reluLayer): Any input value less than zero is set to zero.
- Softmax layer: Applies a softmax function to the input values.

After the network has been initialized, it is used to create the actor of the agent using the *rlDiscreteCategoricalActor* function, along with the observation and action information variables created earlier. The full script of the actor's initialization can be seen in Figure 4.4.

```
%% Creation of the critic and actor networks
observationDimension = observationInfo.Dimension(1);
actionDimension = size(actionMatrix,1);

actorNet = [
    featureInputLayer(observationDimension)
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(64)
    reluLayer()
    fullyConnectedLayer(actionDimension,...
    'WeightsInitializer', 'zeros')
    softmaxLayer("Name","actionProbability")];

actor = rlDiscreteCategoricalActor(actorNet,...
        observationInfo, actionInfo);
```

Figure 4.4: Creation of the actor network.

Critic network

The agent's critic network is created in a similar manner to the actor one. It is the same as the network previously created with the only difference in the network itself being the absence of the final softmax layer. Finally, the critic is created by using the *rlValueFunction*. The corresponding code can be seen in Figure 4.5 below.

```
criticNet = [
    featureInputLayer(observationDimension)
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(64)
    reluLayer()
    fullyConnectedLayer(1,...
    'WeightsInitializer', 'zeros')];

critic = rlValueFunction(criticNet,...
        observationInfo);
```

Figure 4.5: Creation of the critic network.

4.3.3 RL agent configuration

Actor/Critic learning options

The first step towards configuration of the agent is to set the actor's and the critic's learn rates and gradient thresholds, which can have a great impact on the training process. A very high learning rate can make the agent too aggressive, thus rendering its training both unstable and its final result unreliable. A value that's too low will hinder the agent's training by making it too slow and therefore prone to being trapped in local optima. The gradient threshold option will ensure that, even if the maximum learning rate somehow fails to make the agent less aggressive, the agent will still collect experience at a specified maximum pace. The script responsible for implementing the aforementioned variables can be seen in Figure 4.6.

```
%% RL agent parameters
rlActorOpt = rlOptimizerOptions("LearnRate", 3e-3,...
    "GradientThreshold", 0.4);

rlCriticOpt = rlOptimizerOptions("LearnRate", 7e-4,...
    "GradientThreshold", 0.25);
```

Figure 4.6: Learning options configuration.

Agent options

Before the creation and training of the PPO Agent, it is necessary to configure its training settings. The settings that were changed beyond their default values for the agent are:

- Clip factor: Limits the change in each policy update step.
- Experience horizon: Number of steps to calculate the advantage.

- Entropy loss weight: Higher entropy loss weight promotes agent exploration by applying a penalty if the agent is too certain about what action to take, therefore helping the agent escape local optima.
- Mini batch size: Batch size for splitting the experience horizon. A value that's too small can hinder the agent's convergence by making it too slow.
- Advantage estimate method: A method for estimating advantage values. The setting used in this training is "gae" which stands for "generalized advantage estimator".
- Discount factor: Discount factor applied to the selected advantage estimation method.

In addition to those settings, the actor/critic options configured earlier are also included in the agent's configuration. Finally, the agent object is created using the *"rlPPOAgent"* function and passing as input the actor and critic networks as well as the agent options configured in this section, as shown in Figure 4.7.

```
agentOptions = rlPPOAgentOptions(...
    'ClipFactor', 0.025,...
    "ExperienceHorizon", 2048,...
    'EntropyLossWeight', 0.02,...
    "MiniBatchSize", 256, ...
    'AdvantageEstimateMethod', 'gae',...
    "ActorOptimizerOptions", rlActorOpt, ...
    "CriticOptimizerOptions", rlCriticOpt,...
    'DiscountFactor', 0.992);

agentObj = rlPPOAgent(actor, critic, agentOptions);
```

Figure 4.7: Agent configuration.

4.3.4 Training options

Before running the training algorithm for the agent, it is necessary to configure its parameters. The options chosen to be changed are the following:

- Max episodes: Maximum number of episodes to train the agents. If no other termination criteria have been met, the training process terminates after the specified episodes have elapsed. A higher number of episodes in the absence of other termination criteria significantly increases the training time but helps ensure the agent's convergence.

- Max steps per episode: Maximum number of environment steps for training per episode.
- Score averaging window length: Window length for averaging scores and rewards for the agent.
- Verbose: Displays training progress in the command line. Does not affect the training process.

The configuration of the training options variable can be seen in Figure 4.8.

```
%% Training options
trainOpts = rlTrainingOptions(...
    'MaxEpisodes', 20000,...
    'MaxStepsPerEpisode', 5000,...
    'ScoreAveragingWindowLength', 100,...
    'Verbose', true);
```

Figure 4.8: Training options.

4.3.5 Process parallelization

Agent training is a very time-consuming process. In devices with multiple CPU cores and sufficient RAM available, the training process can be distributed across different workers and therefore drastically cut training times down. The script shown in Figure 4.9 asks the user for the desired amount of workers to be used during training. If the user wishes to cancel process parallelization, it can be done by selecting “0” or “1” when prompted. Otherwise, the script checks for active parallel pools and compares it to the user input, deleting the previous and creating a new pool if the selected amount of workers is different from the active pool or simply starting a new one if none are already active.

```
%% Process parallelization
poolsReq=input("Enter the amount of pools to use for " + ...
               "parallel computing (0 or 1 to disable parallelization): ");
if ~isempty(gcp('nocreate'))
    poolsOnline=gcp('nocreate').NumWorkers;
else
    poolsOnline=0;
end
if poolsReq==0 || poolsReq==1
    if ~isempty(poolsOnline)
        delete (gcp('nocreate'));
        trainOpts.UseParallel = false;
    end
elseif poolsReq~=poolsOnline
    delete(gcp("nocreate"));
    parpool('local',poolsReq);
    trainOpts.UseParallel=true;
    trainOpts.ParallelizationOptions.Mode = "async";
else
    trainOpts.UseParallel=true;
    trainOpts.ParallelizationOptions.Mode = "async";
end
```

Figure 4.9: Process parallelization script.

4.3.6 Training execution and results

In order to commence the training process, all that is required is to call the *"train()"* function. Using the agent object and the environment objects and the training options previously specified, training of the agent will start and the output will be its training statistics, with the final agent being exported to the workspace automatically. The trained agent object and the training stats are then saved using the *"save"* function. The corresponding script can be seen in Figure 4.10.

```
%% Training execution
trainingStats = train(agentObj, env, trainOpts);

% Save the agent and the training stats
save('agent_prodlineActive.mat', 'agentObj');
save('trainingStats_2machines.mat', 'trainingStats');
```

Figure 4.10: Agent training and saving.

5. Training and simulation results

5.1 Scenario outlines

In this chapter, the results of the agents' training and simulation testing are presented. For this purpose, 5 Simulink models have been developed. One of those models is the "Baseline" and the rest are slightly different in order to simulate changes in the environment. Those changes are higher client demand, increased processing times or higher energy cost coefficients. For the higher demand scenario, two different models have been used. To conclude, training has taken place in 5 different scenarios: Baseline, Higher Demand, Higher Demand V2, Slower Production and Higher Energy Cost.

An agent has been trained on each of those scenarios, producing 5 different agents. Then, each of those agents is tested in simulations longer than their training in all 5 scenarios. This way, the performance of each agent when faced with environment variable changes outside its training can be assessed, therefore proving their robustness when variables fluctuate. The objective function used in the model is the following:

$$R = -(m_{B1}c_{B1} + m_{B2}c_{B2} + m_c c_c + (EC_1 + EC_2)c_{el} + m_r c_r).$$

Equation 5-1.

The variables of the function are explained below:

- R: Reward value
- m_{B1}: Average number of entities in the buffer after Stage 1.
- c_{B1}: Cost of storing an entity in the buffer after Stage 1.
- m_{B2}: Number of entities in the buffer after Stage 2, waiting to be handed to customers.
- c_{B2}: Cost of storing an entity in the buffer after Stage 2.
- m_c: Clients in the client queue.
- c_c: Cost of keeping a client in queue.
- EC_1/EC_2: Energy consumption coefficient of Stage 1 and Stage 2 respectively.
- c_{el}: Cost of one energy unit.
- m_r: Number of entities in the "raw materials" buffer.
- c_r: Cost of storing one entity of raw material.

5.2 Agent training results

In this chapter, the training results of the agents are presented. Each agent is being trained in its own version of the “Baseline” model. The x and y axis of each diagram shows the episode number and the cumulative reward value (reward values of each step in the episode, totaling 5.000 simulation steps) respectively. The average reward value for the last 100 episodes is traced with a red line. It is reminded that each training took place for a total of 20.000 episodes with 5.000 simulation steps per episode.

The training results of the agents can be seen in the following figures:

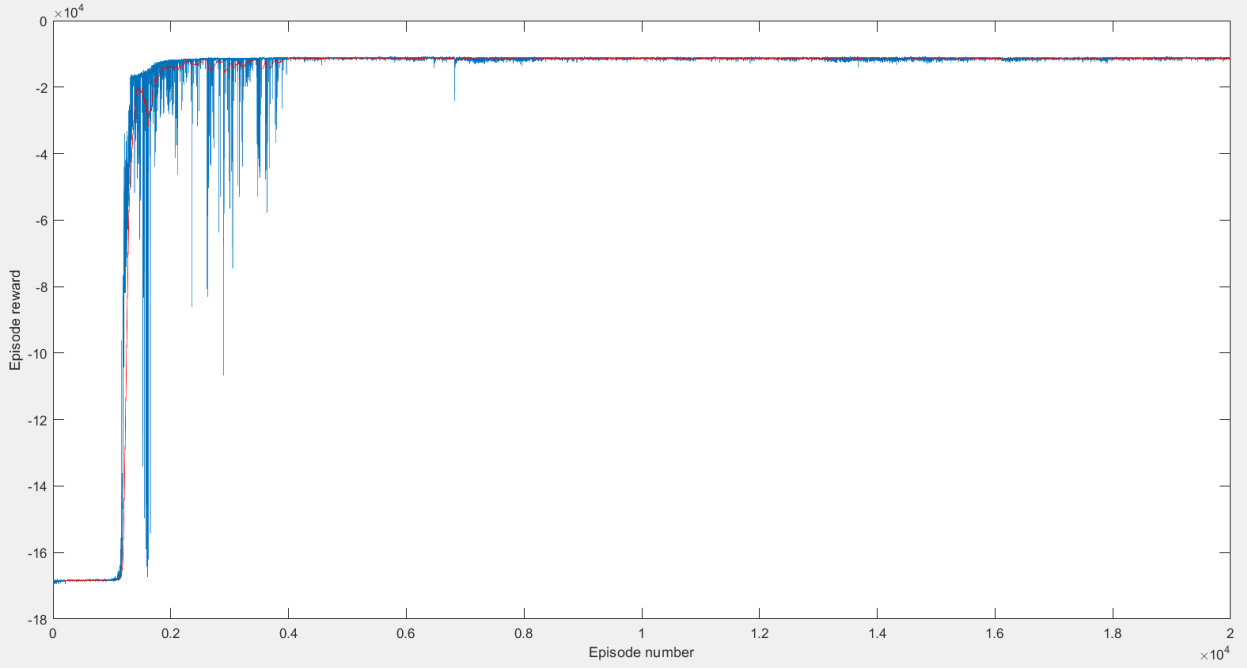

Figure 5.1: Training of the "Baseline" agent.

Remarks:

- Agent shows stability for the most part of its training process after the initial steep experience gain.
- Agent converges at simulation end.

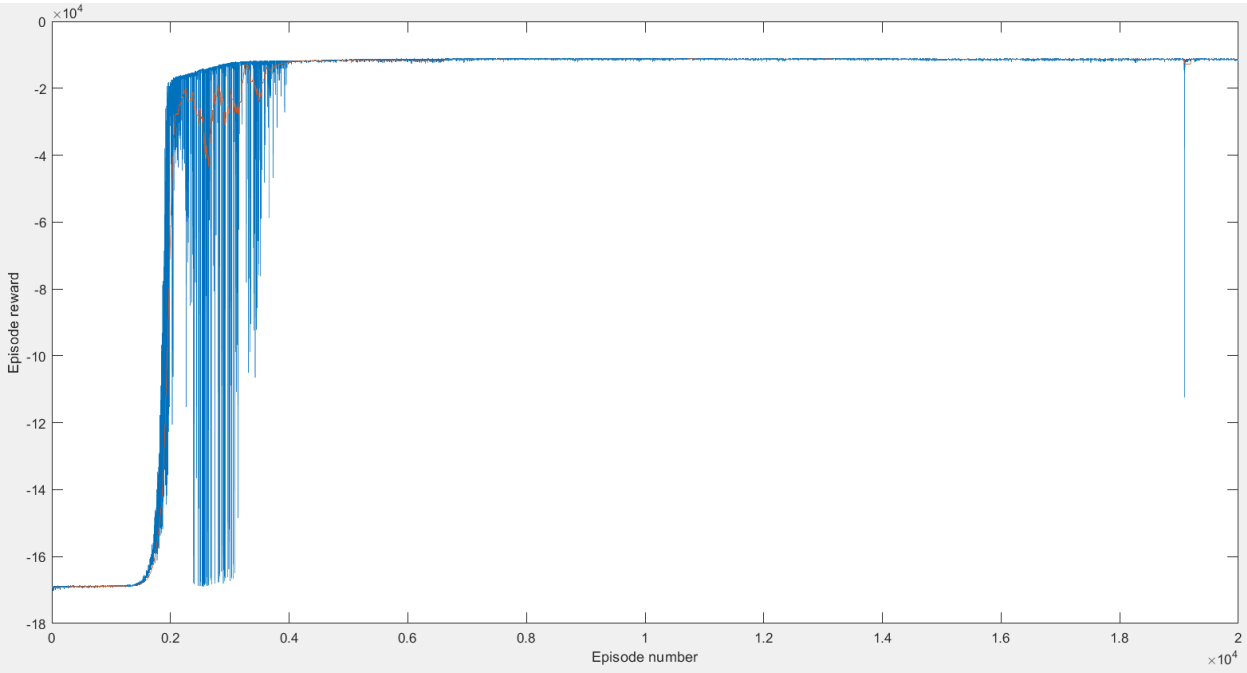

Figure 5.2: Training of the "HighDemand" agent.

Remarks:

- Agent shows less stability for a short time period after the initial steep reward slope.
- An unexpected performance drop appeared for a single episode around the 19.000 episode mark, possibly caused by the stochasticity of the agent's policy.
- Agent converges at the end of the training.

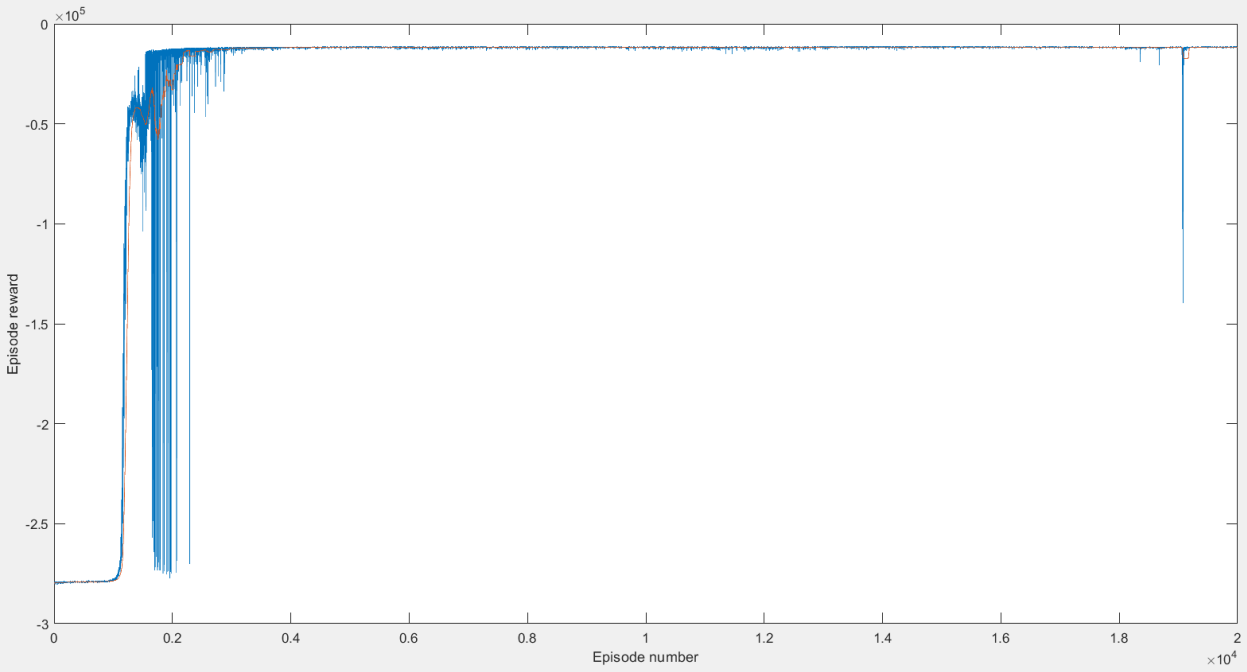

Figure 5.3: Training of the "HighDemandV2" agent.

Remarks:

- Agent is more stable than the “HighDemand” version after the initial reward exploitation, but still less stable than the “Baseline” agent.
- As in the “HighDemand” agent, a performance drop appeared for a single episode around the 19.000 episode mark, possibly due to the stochasticity of the agent’s policy.
- Agent converges at the end of the training.

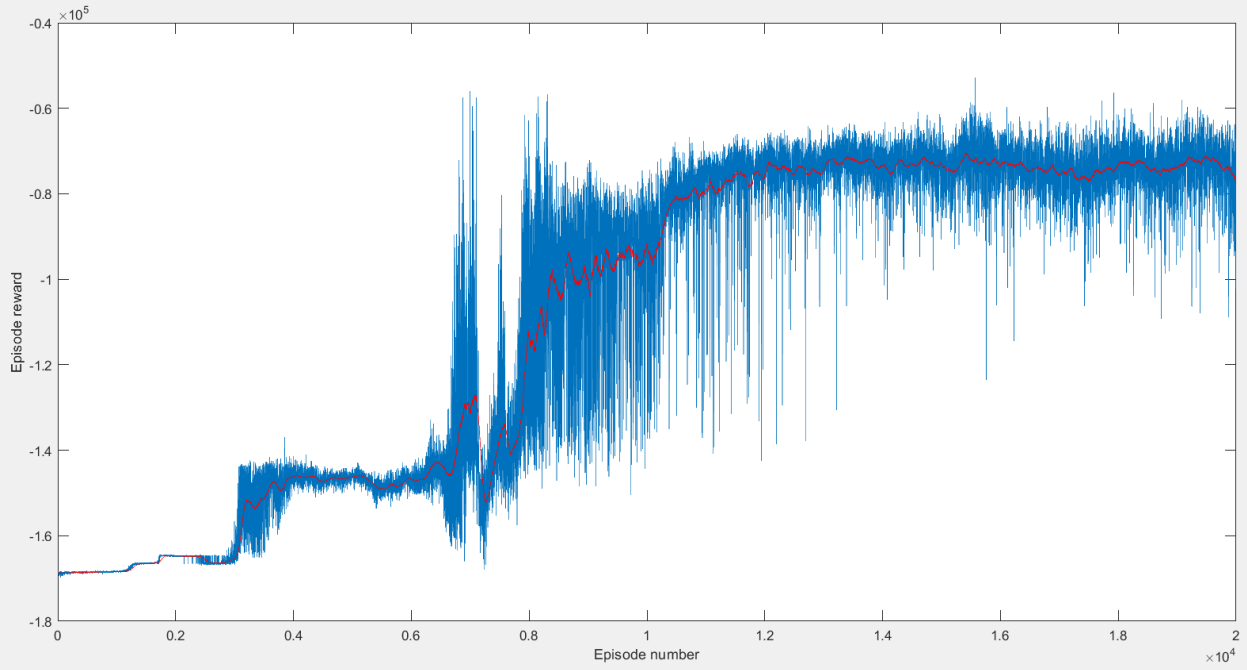

Figure 5.4: Training of the "SlowerProduction" agent.

Remarks:

- Agent shows a general instability in its reward during training.
- Though unstable, the agent achieved a significant improvement of its rewards since the beginning of the training process. Its order of magnitude is within what the previous agents achieved.
- The agent did not converge at the end of the simulation and its reward fluctuated heavily.

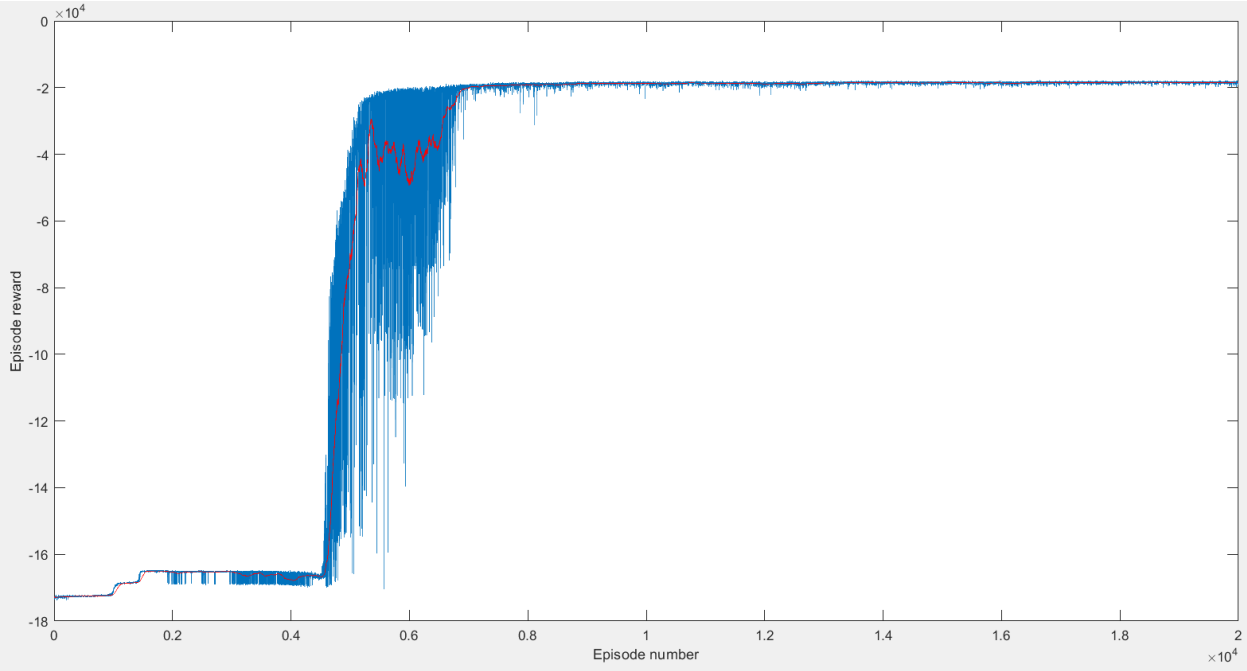

Figure 5.5: Training of the "EnergyCost" agent.

Remarks:

- The agent took a longer time than the previous versions before starting its steep reward gain slope.
- Agent exhibited a slight instability after the initial reward gains.
- Agent converged at the end of the simulation.

In conclusion, the agents generally exhibited a smooth performance during training. This hints that the training parameters were well set and the number of episodes of the training process was sufficient for most agents to converge. Agent convergence contributes to a system whose behavior will not vary widely with small tweaks of its environment variables.

5.3 Simulation results

Agent assessment

After simulation, Simulink outputs the desired data into the basic MATLAB workspace, the most important of which being the average reward, which is the sum of the reward values in each step divided by the number of simulation steps (20.000 for every agent). Other metrics such as average client waiting times, average client queue length, buffer levels etc. will be used to make an outline of the general agent's behavior.

5.3.1 Baseline scenario

This is the first scenario used for agent training. The baseline scenario agent will be compared to the other agents trained in their respective slightly modified models in order to assess the robustness of the controller. The objective function (Equation 5-1) coefficients for this scenario are presented in Table 5-1.

Table 5-1: Objective function coefficients for the Baseline scenario.

Variable name (script)	Variable (Equation 5-1.)	Value
bufferCost	c_{B1}	0.01
readyproductsCost	c_{B2}	0.015
clientWaitCost	c_c	0.65
energyCost	c_{el}	0.1
rawMaterialCost	c_r	0.0025

Average reward values

The agents are loaded in the workspace one by one and are used to run the simulation of the model for 20.000 steps. The average reward for the baseline scenario model for all agents is presented in Table 5-2. It is reminded that, although the "reward" actually represents costs, it has a negative sign (as seen in Equation 5-1), therefore greater values are more desirable.

Table 5-2: Average simulation reward values.

Agent	Average reward
Baseline	-2.575
HighDemand	-2.583
HighDemandV2	-2.625
SlowerProduction	-3.010
EnergyCost	-2.688

From the reward values, it can be argued that the baseline agent outperforms the rest, with the "HighDemand" agent following closely. The worst performance in this metric was exhibited by the "SlowerProduction" agent.

Machine utilization

The next metric under examination is the utilization of Machines 1 and 2. Utilization receives values between 0 and 1 and represents the total machine uptime compared to the total simulation time. Higher values mean the machines have been running for longer time in total throughout the simulation. Lower values mean that the machines have been spending more time in IDLE or OFF states. The results can be seen in Table 5-3.

Table 5-3: Average machine utilization for each agent.

Agent	Machine 1 utilization	Machine 2 utilization
Baseline	0.510	0.493
HighDemand	0.509	0.492
HighDemandV2	0.511	0.493
SlowerProduction	0.611	0.612
EnergyCost	0.510	0.492

In contrast to the average reward values, no direct conclusion about the agents' performance can be made from the utilization values alone. However, the agents can be compared to each other. While all agents stay within very narrow margins of each other in terms of machine utilization, the "SlowerProduction" agent stands out with higher values. This means that the agent kept the machines in OPERATION status for longer than the others.

Client metrics

In terms of client metrics, the average client wait times and the average queue lengths will be examined. In a sense, those values can be considered interchangeable, but both will be presented, as they can be used to form conclusions about overall client satisfaction, facility requirements to support the number of waiting customers etc. The simulation results are shown in Table 5-4.

Table 5-4: Average client wait times and queue lengths for each agent.

Agent	Average client wait time	Average queue length
Baseline	0.072	0.018
HighDemand	0.057	0.014
HighDemandV2	0.048	0.012
SlowerProduction	0.045	0.011
EnergyCost	0.493	0.122

It appears that the SlowerProduction agent is better at servicing clients, as implied by the lower waiting times and queue lengths. The HighDemandV2 agent shows similar performance in terms of those metrics, but slightly worse. The worst performance across all client metrics belongs to the EnergyCost agent.

Average Stage 2 buffer level

Another interesting metric is the average level of the buffer after Stage 2. The Stage 2 buffer contains ready products waiting to be given to customers from the client queue. The average levels can be seen in Table 5-5.

Table 5-5: Average Stage 2 buffer level for each agent.

Agent	Average buffer 2 level
Baseline	5.974
HighDemand	6.419
HighDemandV2	7.449
SlowerProduction	26.089
EnergyCost	2.970

The values show that the SlowerProduction agent keeps the highest stock of ready products by far compared to the rest of the agents, while the EnergyCost agent keeps the lowest stock.

A higher average stock means that if a client surge were to manifest, the system would be better prepared to respond without leaving clients wait until more products are prepared, at the expense of higher storage costs. The other 3 agents (Baseline, HighDemand and HighDemandV2) are relatively close in terms of stocked products.

Conclusion

In this scenario, it can be stated that the agent with the best performance was the Baseline agent. Though not as good as the Slower Production agent in terms of client metrics, it showed the best average reward in the simulations while performing well enough in client metrics and kept a reasonable stock of ready products to diminish the negative effect of a surge in incoming clients in the system.

5.3.2 High Demand scenario

In this scenario, the average arrival times (the entry time attribute of the clients as described in 3.3.5) of the clients have a reduced mean value of μ'=3 compared to the initial μ=4 of the baseline scenario. The objective function remains unchanged compared to the baseline scenario and is presented in Table 5-6.

Table 5-6: Objective function coefficients for the HighDemand scenario.

Variable name (script)	Variable (Equation 5-1.)	Value
bufferCost	c_{B1}	0.01
readyproductsCost	c_{B2}	0.015
clientWaitCost	c_c	0.65
energyCost	c_{el}	0.1
rawMaterialCost	c_r	0.0025

Average reward values

The average reward value for every agent in the HighDemand scenario is shown in Table 5-7. It is reminded that the only agent trained on this scenario is the "HighDemand" agent.

Table 5-7: Average simulation reward values.

Agent	Average reward
Baseline	-3.459
HighDemand	-2.716
HighDemandV2	-2.689
SlowerProduction	-3.154
EnergyCost	-3.446

Of all the agents, the HighDemandV2 agent shows the best average reward value. HighDemand agent exhibits a marginally worse performance. Third in terms of reward is the SlowerProduction agent, with Baseline and EnergyCost agents receiving the worst values.

Machine utilization

The average machine utilization of each agent for both machines is presented in Table 5-8.

Table 5-8: Average machine utilization for each agent.

Agent	Machine 1 utilization	Machine 2 utilization
Baseline	0.680	0.658
HighDemand	0.672	0.659
HighDemandV2	0.673	0.659
SlowerProduction	0.801	0.814
EnergyCost	0.677	0.659

Like previously, the SlowerProduction agent makes considerably more use of the machines' available time, with the rest of the agents being more or less similar to each other.

Compared to the values seen in the Baseline scenario (Table 5-3), the values are remarkably higher in this scenario. This can be explained by the fact that the influx of clients has been greatly increased on average. More clients entering the system means more need for products and therefore more machine usage.

Client metrics

The client metrics of this scenario for all agents can be seen in Table 5-9.

Table 5-9: Average client wait times and queue lengths for each agent.

Agent	Average client wait time	Average queue length
Baseline	3.382	1.124
HighDemand	0.557	0.185
HighDemandV2	0.290	0.097
SlowerProduction	0.098	0.032
EnergyCost	3.378	1.123

In this scenario, the client metrics values have been remarkably increased across all agents. The SlowerProduction agent shows a significantly better performance than the other agents, both in terms of client wait times and average queue length. Both Baseline and the EnergyCost agents show signs of struggling to keep up when compared to the other agents, though their numbers pose no threat to the overall stability of the system (being unable to serve the clients and have a full queue).

Average Stage 2 buffer level

As explained earlier, the average Stage 2 buffer level indicates the average amount of ready products in stock. The results of each agent can be seen in Table 5-10.

Table 5-10: Average Stage 2 buffer level of each agent.

Agent	Average buffer 2 level
Baseline	1.936
HighDemand	3.832
HighDemandV2	4.824
SlowerProduction	25.232
EnergyCost	1.985

Regarding the buffer level, the SlowerProduction agent holds a larger stock of products than the other agents. Baseline and EnergyCost agents are struggling to keep up with demand, while HighDemand and HighDemandV2 are performing slightly better. Once again, SlowerProduction seems better prepared to deal with a possible demand spike.

Conclusion

In this scenario, agents were tested with a higher customer arrival rate. HighDemandV2 agent showed the best performance in terms of average reward, with HighDemand being

close. It is reminded that HighDemand was the agent trained on this scenario. However, when client wait times and client queue length is considered, SlowerProduction agent had the best performance. The same agent also exhibited the highest levels of stocked ready products and machine utilizations.

5.3.3 High Demand V2 scenario

This scenario retains the increase of the customer arrival rates that was made for HighDemand in 5.3.2. Additionally, the client waiting cost coefficient in the objective function (Equation 5-1) has been increased to 1.1 from 0.65 (a +69% increase) in order to more harshly penalize pending orders. The objective function coefficients can be seen in Table 5-11. The agent trained on this scenario is the HighDemandV2 agent.

Table 5-11: Objective function coefficients for the HIghDemandV2 scenario.

Variable name (script)	Variable (Equation 5-1.)	Value
bufferCost	c_{B1}	0.01
readyproductsCost	c_{B2}	0.015
clientWaitCost	$\boldsymbol{c_c}$	**1.1**
energyCost	c_{el}	0.1
rawMaterialCost	c_r	0.0025

Average reward values

The average reward values of all agents for the HighDemandV2 scenario is presented in Table 5-12.

Table 5-12: Average simulation reward values.

Agent	Average reward
Baseline	-3.023
HighDemand	-2.758
HighDemandV2	-2.720
SlowerProduction	-3.173
EnergyCost	-3.973

Across all agents, the one with the best performance is the HighDemandV2 one. HighDemand follows not too far behind, with Baseline being in the middle of the lineup. SlowerProduction and EnergyCost come in the last places, with EnergyCost exhibiting the worst performance compared to the rest of the agents.

Machine utilization

The machine utilization factor for each agent is shown in Table 5-13.

Table 5-13: Average machine utilization for each agent.

Agent	Machine 1 utilization	Machine 2 utilization
Baseline	0.672	0.659
HighDemand	0.672	0.659
HighDemandV2	0.673	0.659
SlowerProduction	0.801	0.814
EnergyCost	0.678	0.658

The machine utilizations in this scenario are almost completely identical to those in the HighDemand scenario, as presented in Table 5-8. This is an expected behavior, as the product demand has not changed. The SlowerProduction agent once again shows the highest utilization rates, with all the other agents showing almost identical performance with each other.

Client metrics

The average client wait times and queue length are presented in Table 5-14.

Table 5-14: Average client wait times and queue lengths for each agent.

Agent	Average client wait time	Average queue length
Baseline	1.201	0.399
HighDemand	0.448	0.149
HighDemandV2	0.247	0.082
SlowerProduction	0.108	0.036
EnergyCost	3.420	1.137

Following the already existing pattern, the SlowerProduction agent scores the best across both client metrics. Between the other agents, HighDemandV2 shows better performance and HighDemand follows behind. The EnergyCost agent ends up in the last place compared to the other agents. Remarkably, when compared to its performance in the last scenario (Table 5-9), the Baseline agent shows a significant improvement in client metrics.

Average Stage 2 buffer level

The average Stage 2 buffer levels for all agents can be seen in Table 5-15.

Table 5-15: Average Stage 2 buffer levels.

Agent	Average buffer 2 level
Baseline	2.980
HighDemand	3.924
HighDemandV2	4.714
SlowerProduction	25.434
EnergyCost	1.875

In this scenario, the SlowerProduction agent keeps the highest stock of ready products. The other agents also keep some stock, though lower, with the lowest being the EnergyCost agent. The EnergyCost agent is more prone to leaving customers to wait, should demand surge suddenly.

Conclusion

In this scenario, on top of the faster client arrival rate, the waiting cost for those clients was also increased to encourage the agents to produce more and cover the extra demand. In terms of average reward, the HighDemandV2 agent had the best performance. The agent only fell behind in terms of client wait times and average queue length when compared to the SlowerProduction agent. However, the agent still performed well and kept a respectable stock of ready products in case of a surge in client arrivals.

5.3.4 Slower Production scenario

This scenario simulates slower machine processing rates for the products. The mean processing times were increased to μ'=3 compared to the initial value of μ=2. The objective function coefficients have been changed back to the same values as in the Baseline scenario, as seen in Table 5-16.

Table 5-16: Objective function coefficients for the SlowerProduction scenario.

Variable name (script)	Variable (Equation 5-1.)	Value
bufferCost	c_{B1}	0.01
readyproductsCost	c_{B2}	0.015
clientWaitCost	c_c	0.65
energyCost	c_{el}	0.1
rawMaterialCost	c_r	0.0025

Average reward values

The average reward values of all agents for the SlowerProduction scenario is presented in Table 5-17.

Table 5-17: Average simulation reward values.

Agent	Average reward
Baseline	-3.459
HighDemand	-3.017
HighDemandV2	-2.994
SlowerProduction	-3.167
EnergyCost	-4.192

The HighDemandV2 agent shows the best performance compared to the other agents, followed by the HighDemand agent. Although it was trained on this exact scenario, the SlowerProduction agent fell in third place in terms of average simulation reward. The Baseline agent came in fourth place and the EnergyCost agent exhibited the worst performance.

Machine utilization

The average machine utilization during the simulation for each agent is shown in Table 5-18.

Table 5-18: Average machine utilization for each agent.

Agent	Machine 1 utilization	Machine 2 utilization
Baseline	0.758	0.735
HighDemand	0.760	0.735
HighDemandV2	0.762	0.736
SlowerProduction	0.801	0.798
EnergyCost	0.766	0.734

In terms of machine utilization, a noticeable increase is observed across all the agents except for the SlowerProduction one. Though increased, machine utilization ratios of the other agents are still much different than the SlowerProduction agent. This increase is expected, as products now require longer processing times by a factor of +50%.

Client metrics

The average client wait time and queue length are shown in Table 5-19.

Table 5-19: Average client wait times and queue lengths for each agent.

Agent	Average client wait time	Average queue length
Baseline	4.668	1.161
HighDemand	1.876	0.465
HighDemandV2	1.525	0.378
SlowerProduction	0.191	0.047
EnergyCost	8.424	2.087

As expected, the SlowerProduction agent once again outperforms the others and this time by a much larger margin, being the only agent with average waiting time less than 1 and less than 0.1 average queue length. The EnergyCost agent performs significantly worse than all the others. The HighDemand and HighDemandV2 agents once again show a good enough performance even when the environment variables are changed.

Average Stage 2 buffer level

The average Stage 2 buffer levels for all the agents are presented in Table 5-20.

Table 5-20: Average Stage 2 buffer levels.

Agent	Average buffer 2 level
Baseline	2.348
HighDemand	3.485
HighDemandV2	4.370
SlowerProduction	21.025
EnergyCost	1.421

Though not significant, the agents demonstrate a decrease in average buffer levels. The SlowerProduction agent once again holds a high level of stocked products, with the EnergyCost agent falling dangerously low and being vulnerable to unexpected demand surges.

Conclusion

In this scenario, the agents were tested in an environment where products require a longer processing time on average. The HighDemand and HighDemandV2 agents demonstrated good enough performance in terms of average simulation reward values and the SlowerProduction agent was once again the best performing agent in terms of client metrics.

5.3.5 Higher energy cost scenario

In this scenario, the energy cost coefficient was increased to 0.02 from its initial value of 0.01, showcasing an increase rate of +100%. This resembles real-world situations where the electricity price can suddenly fluctuate, altering the profitability of a production facility. The objective function coefficients for this scenario are shown in Table 5-21.

Table 5-21: Objective function coefficients for the EnergyCost scenario.

Variable name (script)	Variable (Equation 5-1.	Value
bufferCost	c_{B1}	0.01
readyproductsCost	c_{B2}	0.015
clientWaitCost	c_c	0.65
energyCost	$\boldsymbol{c_{el}}$	**0.2**
rawMaterialCost	c_r	0.0025

Average reward values

The average reward values for this scenario are presented in Table 5-22.

Table 5-22: Average simulation reward values.

Agent	Average reward
Baseline	-3.807
HighDemand	-3.810
HighDemandV2	-3.841
SlowerProduction	-4.446
EnergyCost	-3.894

Though not trained for this scenario, the Baseline agent shows the best performance, followed closely by the HighDemand and HighDemandV2 agents. The worst performance is showcased by the SlowerProduction agent, an expected outcome since this agent exhibits higher machine utilization rates and therefore higher energy costs.

Machine utilization

The utilization of the machines in this scenario is presented in Table 5-23.

Table 5-23: Average machine utilization for each agent.

Agent	Machine 1 utilization	Machine 2 utilization
Baseline	0.510	0.493
HighDemand	0.509	0.492
HighDemandV2	0.511	0.493
SlowerProduction	0.650	0.660
EnergyCost	0.510	0.492

It appears that the agents have now reduced their utilization rates back to the levels shown in Table 5-3 for the Baseline scenario. This is an understandable outcome since both client demand and processing times have returned to their initial values. What would be expected but is not actually happening is a decrease in utilization rates to compensate for the higher operating costs imposed by the increased energy coefficient in the objective function. Furthermore, the EnergyCost agent does not actually make less use of the machines as would be expected in order to save some energy.

Client metrics

The client metrics of all the agents for this scenario are shown in Table 5-24.

Table 5-24: Average client wait times and queue lengths for each agent.

Agent	Average client wait time	Average queue length
Baseline	0.084	0.021
HighDemand	0.061	0.015
HighDemandV2	0.046	0.011
SlowerProduction	0.050	0.012
EnergyCost	0.457	0.113

It is observed that all the agents are performing relatively well in the client metrics, except for the EnergyCost agent, which was trained on this scenario. Remarkably and in contrast to all the previous scenarios, the HighDemandV2 agent marginally outperforms the SlowerProduction agent in both average client wait time and queue length.

Average Stage 2 buffer level

The average buffer levels are shown in Table 5-25.

Table 5-25: Average Stage 2 buffer levels.

Agent	Average buffer 2 level
Baseline	6.135
HighDemand	6.309
HighDemandV2	7.520
SlowerProduction	26.872
EnergyCost	2.949

Following the same trend as in previous scenarios, the EnergyCost agent holds the least amount of ready products in stock. The SlowerProduction agent keeps more stocked products and the rest of the agents hold about the same amounts with each other.

Conclusion

This scenario imposed a greater production cost on the system in the form of an increased energy consumption coefficient. The EnergyCost agent, who was trained on this scenario,

performed poorly when compared to the other agents. The Baseline agent achieved the best average reward value, while the HighDemandV2 agent showed the best performance considering all the examined metrics, balancing customer metrics, reward and stock of ready products.

5.4 Simulation reward plots

In this chapter, the plots containing the reward values throughout the whole simulation for each scenario will be presented. These plots will aid in the understanding of the agents' performance during simulation.

5.4.1 Baseline scenario

In Figure 5.6, the reward value for each agent throughout the entire simulation of the Baseline scenario is presented.

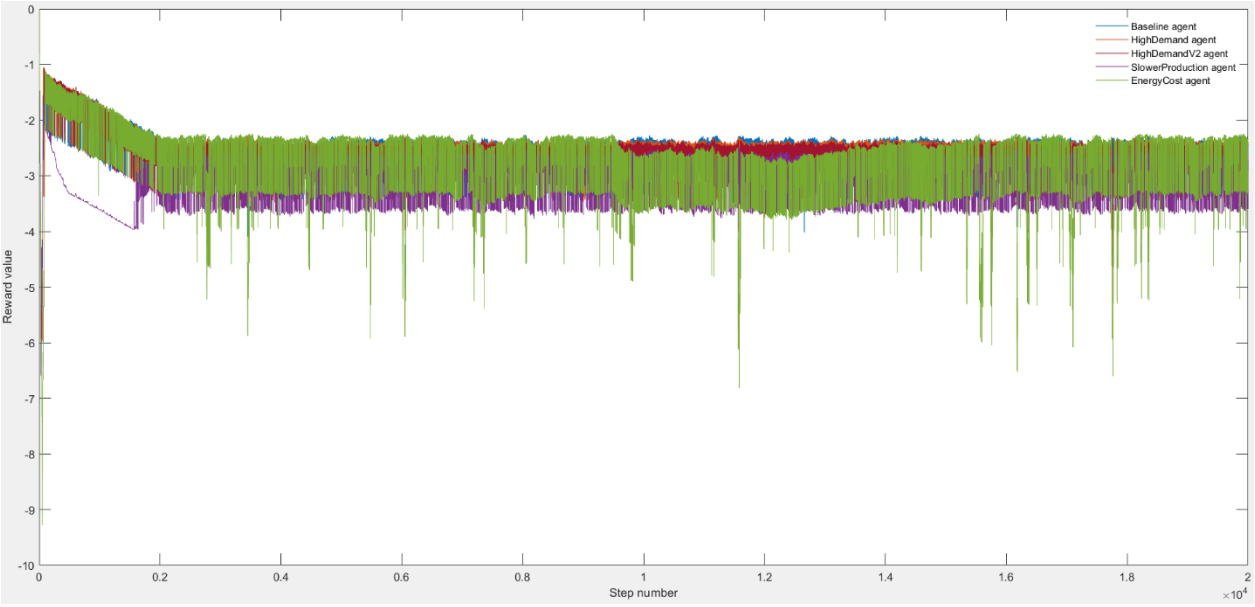

Figure 5.6: Reward values of the agents in the Baseline scenario.

Though not immediately distinguishable, few statements can be made regarding the figure:

- The Baseline agent's peak reward values (blue line) can be seen above all the other curves occasionally, and more intensely during steps 10.000-15.000.

- The HighDemandV2 agent shows more stable performance in areas where other agents might exhibit a performance dip.
- The SlowerProduction agent shows constantly worse reward values than the other agents.
- The EnergyCost agent shows sudden drops in reward, which can be accredited to the low stock of ready products, causing clients who enter the queue to wait and add a penalty to the agent's reward.
- All agents have large negative reward values in the beginning of the simulation, as clients enter the system and wait for the first products to finish processing. The EnergyCost agent is affected the most by this phenomenon.
- In spite of the previous statements, the EnergyCost agent manages to show better performance than the rest occasionally, such as around step 9.000 and between steps 2.000 and 3.000

5.4.2 High Demand scenario

In this scenario, demand for products has been increased. In Figure 5.7, the reward value for each agent throughout the entire simulation in the HighDemand scenario is presented.

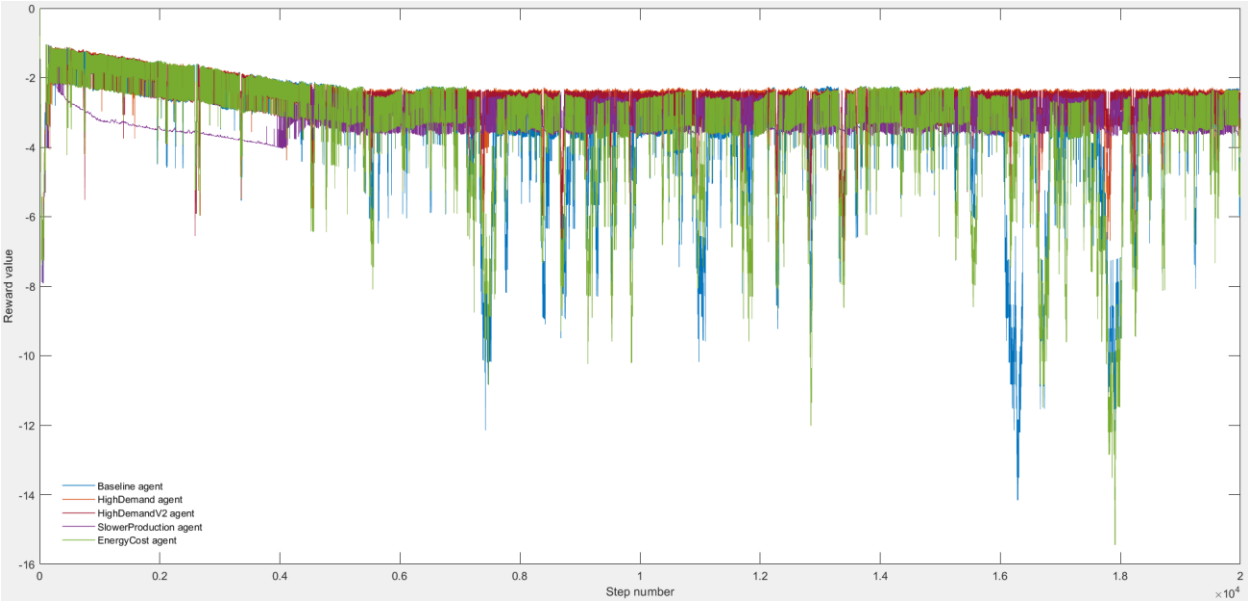

Figure 5.7: Reward values of the agents in the HighDemand scenario.

The differences in the agents' behaviors are less subtle in this scenario. More notably:

- Sudden dips in reward are more intense for all agents. The Baseline and EnergyCost agents are the most affected.
- Though they both exhibit a similar performance, HighDemandV2 is less affected than HighDemand by client surges. It is expected, as both were trained in higher demand scenarios and especially HighDemandV2 received a higher penalty for pending orders during training.
- The EnergyCost agent exhibited the largest dip in reward across all the agents, just before step number 18.000.
- SlowerProduction agent shows no sudden reward drops, most likely caused by the larger stock of ready products the agent keeps.

In this scenario, sudden surges in demand are more seriously impacting the performance of all the agents compared to the Baseline scenario. Sudden drops in performance are indicative of clients rushing in the system's queue while no ready products are stocked, heavily impacting the reward value.

5.4.3 High Demand V2 scenario

In this scenario, in addition to increased demand, pending orders are more heavily penalized. The reward values of the agents during simulation are shown in Figure 5.8.

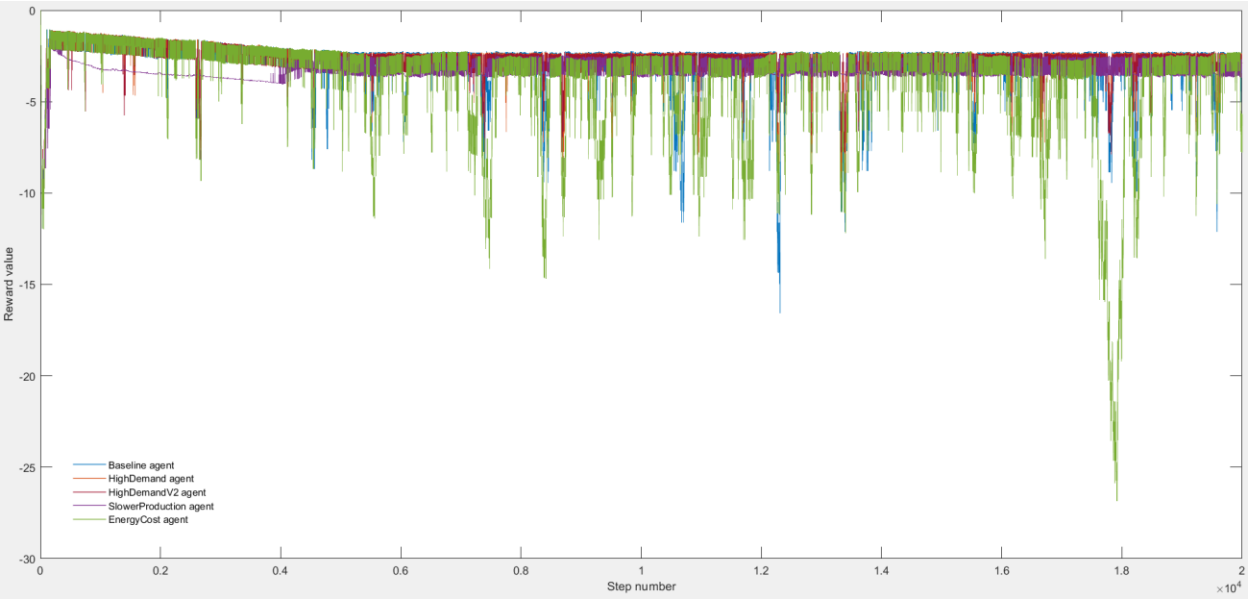

Figure 5.8: Reward values of the agents in the HighDemandV2 scenario.

The main characteristics of the agents' performances are the following:

- Baseline and EnergyCost agents exhibit similar reward drops as those seen previously in Chapter 5.4.2.
- The EnergyCost agent has an immense reward drop around the 18.000th simulation step.
- Though not as notable, HighDemand and HighDemandV2 agents also have performance drops.
- SlowerProduction agent does not show drops in performance.

5.4.4 Slower Production scenario

This scenario simulates longer average processing times for the products. The performance of each agent in the SlowerProduction scenario is shown in Figure 5.9.

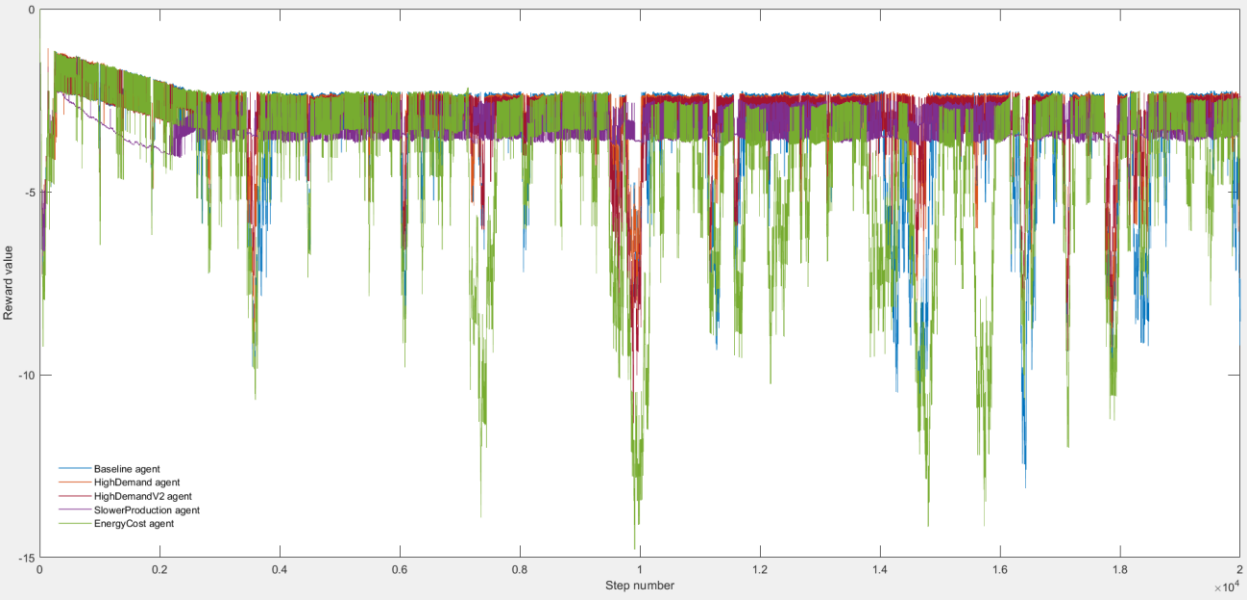

Figure 5.9: Reward values of the agents in the SlowerProduction scenario.

The main takeaways from this scenario are the following:

- All agents are more affected by client surges.
- The EnergyCost agent is less able to respond in time to increased demand, showing more intense and longer-lasting performance drops.

- The agents trained in higher demand scenarios are also impacted, though not as heavily.
- The SlowerProduction agent is not affected by spikes in client demand.
- When generally stable, the EnergyCost and SlowerProduction agents showcase lower reward values than the other agents, more prominent in steps 11.000 through 14.000.

5.4.5 Higher energy cost scenario

In this scenario, the energy cost coefficient of the objective function has been increased, simulating higher energy costs. The reward values of the agents can be seen in Figure 5.10.

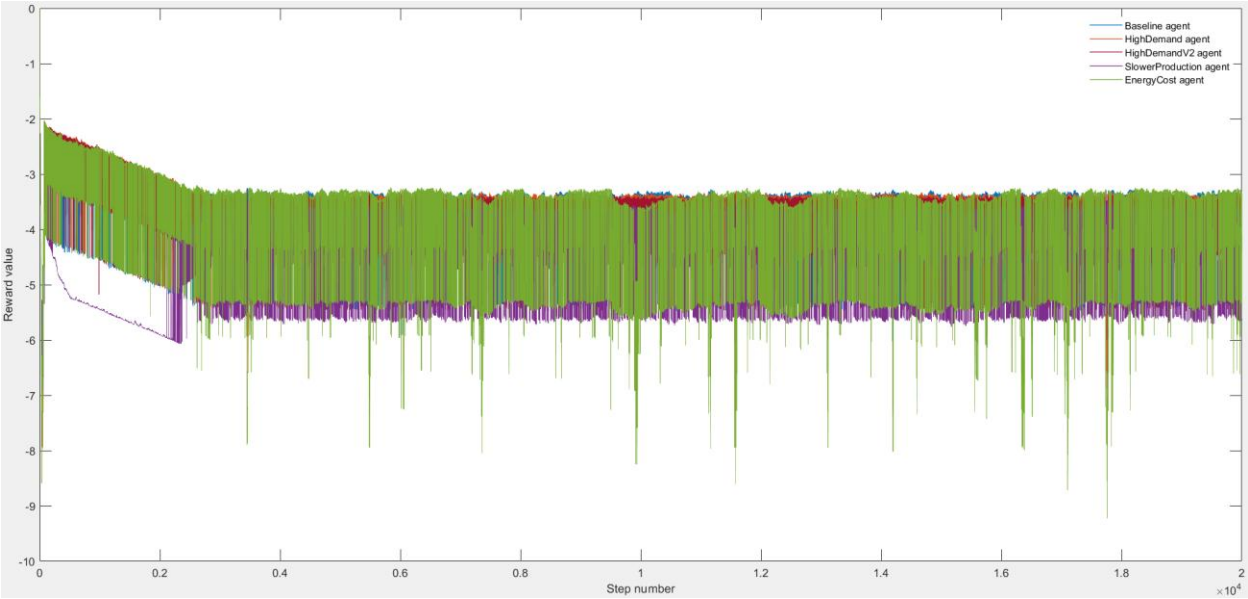

Figure 5.10: Reward values of the agents in the EnergyCost scenario.

By looking at the plot, the following statements can be made:

- All agents exhibit similar behavior.
- The SlowerProduction agent appears to be consistently worse performing than the other agents, as a consequence of higher machine utilization and therefore higher energy expenses.
- The EnergyCost agent seems to outperform the others at times, namely around steps 6.500 and 18.300.

- Sudden reward drops are more subtle in this scenario. Still, the EnergyCost agent is most affected by them.

6. Conclusions and future extensions

In this thesis, a system mimicking a production line has been developed in MATLAB's Simulink environment. The system features two serially connected processing stages controlled by a Reinforcement Learning (RL) agent and a client queue, where clients are accumulated and wait to receive ready products. A discrete-event chart is used to simulate the state of the workstation in each processing stage, which is then directly controlled by the RL agent. The agent has been trained to optimize the cost of running the production line, which includes the costs of storing items in the system's buffers, pending orders and energy consumption. Slightly modified versions of the baseline scenario have been developed to assess the agent's performance in scenarios with increased client demand, slower product processing rates and increased energy costs, as well as the need to train different agents in order to handle those fluctuations of the environment variables. Testing and measurement of different parameters proved that the agents are capable of effectively controlling the production line, but are vulnerable against changes in environment variables, with the effect being more profound on agents who were trained in less demanding scenarios.

It is recognized that there are several restrictions on the empirical research done in this thesis. Therefore, the following future extensions are recommended:

- Developed RL agents can be compared to benchmark control policies to assess their performance against already existing methods of industrial control.
- Events such as machine malfunctions can be added in the simulation process, drastically changing the control process.
- More machines can be added on the production line, increasing the complexity of the problem at hand.
- Other RL optimization methods than the PPO method can be tested.
- Orders higher than one item per client can be simulated.
- Different end products may originate from the same production line and can be combined with customer orders of more than one type of product.

7. Appendices

7.1 Appendix A: Probability distributions

7.1.1 Continuous uniform distribution

In probability theory and statistics, a continuous uniform distribution describes an experiment where its outcome lines between specified bounds, defined by the parameters *a* (lower bound) and *b* (upper bound). The interval may be closed (including the bounds, symbolized *[a,b]*) or open (not including its bounds, symbolized *(a,b)*). For an experiment whose outcomes follow the continuous uniform distribution, all outcomes are equally probable. The main characteristics of the distribution are [38]:

- Probability density function: $f(x) = \begin{cases} \frac{1}{(b-a)}, for\ a \leq x \leq b, \\ 0, for\ x < a\ or\ x > b. \end{cases}$
- Mean value: $\mu = \frac{1}{2}(a+b).$
- Variance: $\sigma^2 = \frac{1}{12}(b-a)^2.$

Figure 7.1 shows the probability density function of a uniform distribution bounded in *[a,b]*:

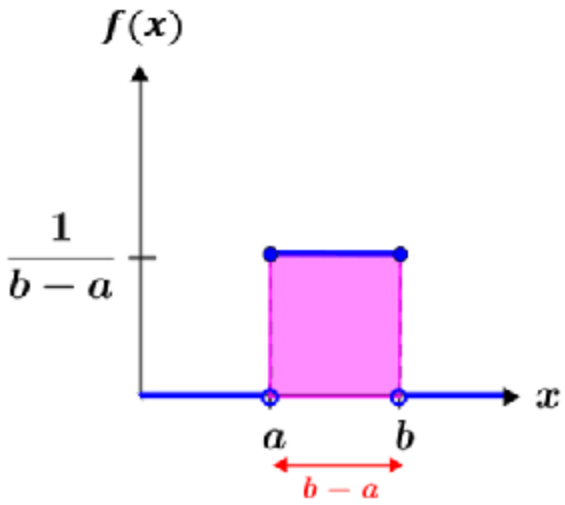

Figure 7.1: Continuous uniform distribution probability density function [39].

7.1.2 Exponential distribution

In probability theory and statistics, the exponential distribution is the probability distribution of the time period between successive events following the Poisson point process. In a Poisson process, events occur continuously and independently of each other at a constant average rate. Those events can range from production errors, client arrivals to natural disasters and other phenomena. The key point of the exponential distribution is that it's memoryless. Its main characteristics are [40]:

- Probability density function: $f(x) = \begin{cases} \lambda e^{-\lambda x}, for\ x \geq 0, \\ 0, for\ x < 0. \end{cases}$

  where λ>0 is the distribution's rate parameter.

- Mean value: $\mu = \frac{1}{\lambda}$.

- Variance: $\sigma^2 = \frac{1}{\lambda^2}$.

An example of the exponential distribution for λ=0.01 and λ=0.005 is shown in Figure 7.2.

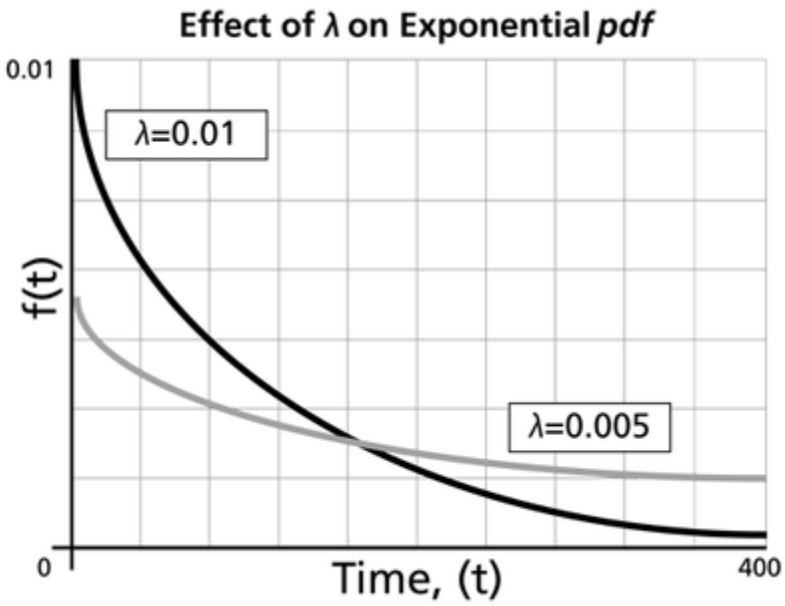

Figure 7.2: Example of an exponential distribution [41].

7.2 Appendix B: MATLAB code for agent training

The full MATLAB code used to train the RL agents is the following:

```
%% Initialization
module = 'prodline_Baseline';%Environment loading
agentBlock = [module '/RL Agent Block'];%Agent block

%Possible machine states
observationInfo = rlNumericSpec([6,1], ...
    'LowerLimit', [0;0;0;0;0;0],...
    'UpperLimit', [1;1;1;1;3;3]);
observationInfo.Name = 'Machine states';

%Creation of the possible states matrix
actM1=[0,1,2];
actM2=actM1;
actionMatrix = combvec(actM1,actM2)';

%Splits actionMatrix into 16 different cell arrays with dimension 2
%each. Then it passes them as possible agent states
actionInfo = rlFiniteSetSpec(mat2cell(actionMatrix, ones(1,9), 2));
actionInfo.Name = 'Machine actions';

env = rlSimulinkEnv(module,agentBlock, observationInfo, actionInfo);

%% Creation of the critic and actor networks
observationDimension = observationInfo.Dimension(1);
actionDimension = size(actionMatrix,1);

actorNet = [
    featureInputLayer(observationDimension)
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(64)
    reluLayer()
    fullyConnectedLayer(actionDimension,...
    'WeightsInitializer', 'zeros')
    softmaxLayer("Name","actionProbability")];

actor = rlDiscreteCategoricalActor(actorNet,...
        observationInfo, actionInfo);

criticNet = [
    featureInputLayer(observationDimension)
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(64)
    reluLayer()
    fullyConnectedLayer(1,...
```

```
    'WeightsInitializer', 'zeros')];

critic = rlValueFunction(criticNet,...
        observationInfo);

%% RL agent parameters
rlActorOpt = rlOptimizerOptions("LearnRate", 3e-3,...
    "GradientThreshold", 0.4);

rlCriticOpt = rlOptimizerOptions("LearnRate", 7e-4,...
    "GradientThreshold", 0.25);

agentOptions = rlPPOAgentOptions(...
    'ClipFactor', 0.025,...
    "ExperienceHorizon", 2048,...
    'EntropyLossWeight', 0.02,...
    "MiniBatchSize", 256, ...
    'AdvantageEstimateMethod', 'gae',...
    "ActorOptimizerOptions", rlActorOpt, ...
    "CriticOptimizerOptions", rlCriticOpt,...
    'DiscountFactor', 0.992);

agentObj = rlPPOAgent(actor, critic, agentOptions);

%% Training options
trainOpts = rlTrainingOptions(...
    'MaxEpisodes', 20000,...
    'MaxStepsPerEpisode', 5000,...
    'ScoreAveragingWindowLength', 100,...
    'Verbose', true);

%% Process parallelization
poolsReq=input("Enter the amount of pools to use for " + ...
               "parallel computing (0 or 1 to disable parallelization): ");
if ~isempty(gcp('nocreate'))
    poolsOnline=gcp('nocreate').NumWorkers;
else
    poolsOnline=0;
end
if poolsReq==0 || poolsReq==1
    if ~isempty(poolsOnline)
        delete (gcp('nocreate'));
        trainOpts.UseParallel = false;
    end
elseif poolsReq~=poolsOnline
    delete(gcp("nocreate"));
    parpool('local',poolsReq);
    trainOpts.UseParallel=true;
    trainOpts.ParallelizationOptions.Mode = "async";
else
    trainOpts.UseParallel=true;
    trainOpts.ParallelizationOptions.Mode = "async";
end

%% Training execution
```

```
trainingStats = train(agentObj, env, trainOpts);

% Save the agent and the training stats
save('agent_prodlineActive.mat', 'agentObj');
save('trainingStats_2machines.mat', 'trainingStats');
```

7.3 Appendix C: Full simulation results

Simulation of the trained agents produced results that were not essential to the performance analysis. For readers interested in a more in-depth analysis, all the produced tables are placed below:

Table 7-1: Average simulation reward.

Average Simulation Reward (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	**-2.575**	-3.459	-3.023	-3.459	**-3.807**
	HighDemand	-2.583	-2.716	-2.758	-3.017	-3.810
	HighDemandV2	-2.625	**-2.689**	**-2.720**	**-2.994**	-3.841
	Slower Production	-3.010	-3.154	-3.173	-3.167	-4.446
	EnergyCost	-2.688	-3.446	-3.973	-4.192	-3.894

Table 7-2: Average Machine 1 utilization.

Average Machine 1 Utilization (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	0.510	0.680	0.672	0.758	0.510
	HighDemand	0.509	0.672	0.672	0.760	0.509
	HighDemandV2	0.511	0.673	0.673	0.762	0.511
	Slower Production	0.611	0.801	0.801	0.801	0.650
	EnergyCost	0.510	0.677	0.678	0.766	0.510

Table 7-3: Average Machine 2 utilization.

Average Machine 2 Utilization (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	0.493	0.658	0.659	0.735	0.493
	HighDemand	0.492	0.659	0.659	0.735	0.492
	HighDemandV2	0.493	0.659	0.659	0.736	0.493
	Slower Production	0.612	0.814	0.814	0.798	0.660
	EnergyCost	0.492	0.659	0.658	0.734	0.492

Table 7-4: Average Stage 1 buffer level.

Average Buffer 1 level (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	3.687	20.924	3.388	3.461	3.832
	HighDemand	4.854	5.134	4.966	5.401	4.863
	HighDemandV2	7.166	7.045	7.041	7.537	7.117
	Slower Production	5.001	11.000	10.938	6.726	6.692
	EnergyCost	11.692	19.544	19.987	19.129	8.926

Table 7-5: Average Stage 2 buffer level.

Average Buffer 2 level (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	5.974	1.936	2.980	2.348	6.135
	HighDemand	6.419	3.832	3.924	3.485	6.309
	HighDemandV2	7.449	4.824	4.714	4.370	7.520
	Slower Production	26.089	25.232	25.434	21.025	26.872
	EnergyCost	2.970	1.985	1.875	1.421	2.949

Table 7-6: Maximum client queue spike.

Maximum client queue spike (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	7	16	12	15	7
	HighDemand	7	9	9	11	7
	HighDemandV2	7	9	9	12	7
	Slower Production	7	9	9	8	7
	EnergyCost	11	18	22	18	7

Table 7-7: Average client wait times.

Average client wait times (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	0.072	3.382	1.201	4.668	0.084
	HighDemand	0.057	0.557	0.448	1.876	0.061
	HighDemandV2	0.048	0.290	0.247	1.525	0.046
	Slower Production	0.045	0.098	0.108	0.191	0.050
	EnergyCost	0.493	3.378	3.420	8.424	0.457

Table 7-8: Average client queue length.

Average client queue length (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	0.018	1.124	0.399	1.161	0.021
	HighDemand	0.014	0.185	0.149	0.465	0.015
	HighDemandV2	0.012	0.097	0.082	0.378	0.011
	Slower Production	0.011	0.032	0.036	0.047	0.012
	EnergyCost	0.122	1.123	1.137	2.087	0.113

Table 7-9: Average Machine 1 energy state coefficient.

Average Machine 1 energy state coefficient (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	6.191	6.990	6.866	7.034	6.196
	HighDemand	6.199	6.885	6.896	7.093	6.198
	HighDemandV2	6.225	6.904	6.927	7.137	6.204
	Slower Production	6.576	7.674	7.657	7.648	6.692
	EnergyCost	6.276	6.999	7.007	7.269	6.270

Table 7-10: Average Machine 2 energy state coefficient.

Average Machine 2 energy state coefficient (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	6.332	7.108	7.358	7.718	6.319
	HighDemand	6.247	7.151	7.130	7.395	6.275
	HighDemandV2	6.248	7.092	7.115	7.333	6.230
	Slower Production	7.397	8.375	8.380	8.473	7.477
	EnergyCost	6.355	7.123	7.119	7.385	6.369

Table 7-11: Total clients served.

Clients served (20.000 steps)						
		Scenarios				
		Baseline	HighDemand	HighDemandV2	Slower Production	EnergyCost
Agents	Baseline	4955	6646	6648	4947	4955
	HighDemand	4955	6648	6648	4955	4955
	HighDemandV2	4955	6648	6648	4955	4955
	Slower Production	4955	6648	6648	4955	4955
	EnergyCost	4955	6647	6645	4954	4955

8. Bibliography

[1] https://blog.gieicom.com/english/cutting-edge-solutions-for-your-assembly-line.

[2] Bierbooms, R. (2012). Performance analysis of production lines: discrete and continuous flow models. [Phd Thesis 2 (Research NOT TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR734088

[3] H.T. Papadopoulos, C. Heavey, Queueing theory in manufacturing systems analysis and design: A classification of models for production and transfer lines, European Journal of Operational Research, Volume 92, Issue 1, 1996, Pages 1-27, ISSN 0377-2217, https://doi.org/10.1016/0377-2217(95)00378-9. (https://www.sciencedirect.com/science/article/pii/0377221795003789).

[4] Baicun Wang, Fei Tao, Xudong Fang, Chao Liu, Yufei Liu, Theodor Freiheit: Smart Manufacturing and Intelligent Manufacturing: A Comparative Review, Engineering, Volume 7, Issue 6, 2021, Pages 738-757, ISSN 2095-8099, https://doi.org/10.1016/j.eng.2020.07.017. (https://www.sciencedirect.com/science/article/pii/S2095809920302502).

[5] Ji Zhou, Peigen Li, Yanhong Zhou, Baicun Wang, Jiyuan Zang, Liu Meng, Toward New-Generation Intelligent Manufacturing, Engineering, Volume 4, Issue 1, 2018, Pages 11-20, ISSN 2095-8099, https://doi.org/10.1016/j.eng.2018.01.002. (https://www.sciencedirect.com/science/article/pii/S2095809917308652).

[6] Sabadka D. , Molnár V. , Fedorko G. , Jachowicz T., Optimization of production processes using the Yamazumi method, Advances in Science and Technology. Research Journal, Vol. 11, no. 4, 2017, pages 175-182, http://dx.doi.org/10.12913/22998624/80921.

[7] Xia, T., Sun, H., Ding, Y. *et al.* Digital twin-based real-time energy optimization method for production line considering fault disturbances. *J Intell Manuf* **36**, 569–593 (2025). https://doi.org/10.1007/s10845-023-02219-9.

[8] D. Meike, M. Pellicciari and G. Berselli, "Energy Efficient Use of Multirobot Production Lines in the Automotive Industry: Detailed System Modeling and Optimization," in *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 798-809, July 2014, doi: 10.1109/TASE.2013.2285813.

[9] Maccarthy, B. L., & Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, *31*(1), 59–79. https://doi.org/10.1080/00207549308956713.

[10] Hao Ding, Sebastian Kain, Frank Schiller, Olaf Stursberg, Increasing Reliability of Intelligent Manufacturing Systems by Adaptive Optimization and Safety Supervision, IFAC Proceedings Volumes, Volume 42, Issue 8, 2009, Pages 1533-1538, ISSN 1474-6670, ISBN 9783902661463, https://doi.org/10.3182/20090630-4-ES-2003.00250. (https://www.sciencedirect.com/science/article/pii/S1474667016359936).

[11] Jørn Vatn, Terje Aven, An approach to maintenance optimization where safety issues are important, Reliability Engineering & System Safety, Volume 95, Issue 1, 2010, Pages 58-63, ISSN 0951-8320, https://doi.org/10.1016/j.ress.2009.06.002. (https://www.sciencedirect.com/science/article/pii/S095183200900132X).

[12] Thomas Sobottka, Felix Kamhuber, Mohammadali Faezirad, Wilfried Sihn, Potential for Machine Learning in Optimized Production Planning with Hybrid Simulation, Procedia Manufacturing, Volume 39, 2019, Pages 1844-1853, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2020.01.254. (https://www.sciencedirect.com/science/article/pii/S2351978920303188).

[13] Tekin, E., & Sabuncuoglu, I. (2004). Simulation optimization: A comprehensive review on theory and applications. IIE Transactions, 36(11), 1067–1081. https://doi.org/10.1080/07408170490500654.

[14] Tsourveloudis, N., Doitsidis, L. & Ioannidis, S. Work-in-process scheduling by evolutionary tuned fuzzy controllers. *Int J Adv Manuf Technol* **34**, 748–761 (2007). https://doi.org/10.1007/s00170-006-0636-x.

[15] Nguyen, S., Mei, Y. & Zhang, M. Genetic programming for production scheduling: a survey with a unified framework. *Complex Intell. Syst.* **3**, 41–66 (2017). https://doi.org/10.1007/s40747-017-0036-x.

[16] Jianzhi Li, Miguel González, Yun Zhu, A hybrid simulation optimization method for production planning of dedicated remanufacturing, International Journal of Production Economics, Volume 117, Issue 2, 2009, Pages 286-301, ISSN 0925-5273, https://doi.org/10.1016/j.ijpe.2008.11.005. (https://www.sciencedirect.com/science/article/pii/S0925527308003642).

[17] Stuart Russell, Peter Norvig, Artificial Intelligence: A modern approach, 4th edition, ISBN 978-960-645-187-4 (Greek language edition).

[18] Fleury, G., Goujon, JY., Gourgand, M. *et al*. Multi-agent approach and stochastic optimization: random events in manufacturing systems. *Journal of Intelligent Manufacturing* **10**, 81–101 (1999). https://doi.org/10.1023/A:1008972615329.

[19] Iannino, V., Vannocci, M., Vannucci, M., Colla, V., Neuer, M., A Multi-Agent Approach for the Self-Optimization of Steel Production, International Journal of Simulation Systems, Science & Technology, DOI 10.5013/IJSSST, https://ijssst.info/Vol-19/No-5/paper20.pdf.

[20] Alkhalefah, H., Umer, U., Abidi, M. H., & Elkaseer, A. (2023). Development and Numerical Optimization of a System of Integrated Agents for Serial Production Lines. *Processes*, *11*(5), 1578. https://doi.org/10.3390/pr11051578.

[21] S.G. Li, Y.L. Rong, The reliable design of one-piece flow production system using fuzzy ant colony optimization, Computers & Operations Research, Volume 36, Issue 5, 2009, Pages 1656-1663, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2008.03.010. (https://www.sciencedirect.com/science/article/pii/S0305054808000737).

[22] Chen, Xiaobo, Yu, Fangfang, Zhou, Hengyu, Li, Zhengdao, Wu, Kuo-Jui, Qian, Xikun, Mixed Production Line Optimization of Industrialized Building Based on Ant Colony Optimization Algorithm, Advances in Civil Engineering, 2022, 2411458, 12 pages, 2022. https://doi.org/10.1155/2022/2411458.

[23] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, Andreas Kyek, Optimization of global production scheduling with deep reinforcement learning, Procedia CIRP, Volume 72, 2018, Pages 1264-1269, ISSN 2212-8271, https://doi.org/10.1016/j.procir.2018.03.212. (https://www.sciencedirect.com/science/article/pii/S221282711830372X).

[24] Waldomiro Alves Ferreira Neto, Cristiano Alexandre Virgínio Cavalcante, Phuc Do, Deep reinforcement learning for maintenance optimization of a scrap-based steel production line, Reliability Engineering & System Safety, Volume 249, 2024, 110199,ISSN 0951-8320, https://doi.org/10.1016/j.ress.2024.110199. (https://www.sciencedirect.com/science/article/pii/S0951832024002722).

[25] Wiering, M. A., & Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, *12*(3), 729.

[26] Klassen, T., Q., Valenzano, R., McIlraith, S., A., Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning, https://doi.org/10.1613/jair.1.12440.

[27] Eschmann, J. (2021). Reward Function Design in Reinforcement Learning. In: Belousov, B., Abdulsamad, H., Klink, P., Parisi, S., Peters, J. (eds) Reinforcement Learning Algorithms: Analysis and Applications. Studies in Computational Intelligence, vol 883. Springer, Cham. https://doi.org/10.1007/978-3-030-41188-6_3.

[28] Bagnell, J. Andrew; Schneider, Jeff (2018). Covariant Policy Search. Carnegie Mellon University. Journal contribution. https://doi.org/10.1184/R1/6552458.v1.

[29] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar and S. Levine, "Collective robot reinforcement learning with distributed asynchronous guided policy search," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 2017, pp. 79-86, doi: 10.1109/IROS.2017.8202141.

[30] Gabel, T., & Riedmiller, M. (2011). Distributed policy search reinforcement learning for job-shop scheduling tasks. *International Journal of Production Research*, *50*(1), 41–61. https://doi.org/10.1080/00207543.2011.571443.

[31] Esteso, A., Peidro, D., Mula, J., & Díaz-Madroñero, M. (2022). Reinforcement learning applied to production planning and control. *International Journal of Production Research*, *61*(16), 5772–5789. https://doi.org/10.1080/00207543.2022.2104180.

[32] Schulman, J., Wolski, F., Prafulla, D., Radford, A., Klimov, O. Proximal Policy Optimization Algorithms, 2017. https://doi.org/10.48550/arXiv.1707.06347, https://arxiv.org/abs/1707.06347.

[33] Bellemare, M., G., Naddaf, Y., Veness, J., Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents, Journal of Artificial Intelligence Research, 2013, https://doi.org/10.1613/jair.3912.

[34] Lim, H.-K., Kim, J.-B., Heo, J.-S., & Han, Y.-H. (2020). Federated Reinforcement Learning for Training Control Policies on Multiple IoT Devices. *Sensors*, *20*(5), 1359. https://doi.org/10.3390/s20051359.

[35] Lin Li , Qing Chang & Jun Ni (2009) Data driven bottleneck detection of manufacturing systems, International Journal of Production Research, 47:18, 5019-5036, DOI: 10.1080/00207540701881860.

[36] Y. C. Ho, M. A. Eyler & T. T. Chien (1979) A gradient technique for general buffer storage design in a production line, International Journal of Production Research, 17:6, 557-580, DOI: 10.1080/00207547908919637.

[37] https://www.mathworks.com/help/simevents/ug/discrete-event-systems-created-with-stateflow-charts.html.

[38] https://en.wikipedia.org/wiki/Continuous_uniform_distribution.

[39] https://www.math.net/uniform-distribution.

[40] https://en.wikipedia.org/wiki/Exponential_distribution.

[41] https://www.reliawiki.com/index.php/The_Exponential_Distribution.