NATIONAL TECHNICAL UNIVERSITY OF ATHENS

School of Mechanical Engineering

Manufacturing Technology Section

Investigation of Reinforcement Learning for Optimal Part Dispatching by Drones in Flexible Manufacturing Systems

Charikleia Angelidou

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

Master of Science in Automation Systems & Robotics

Supervisor

G. - C. Vosniakos
Professor, NTUA

Athens, April 2022

"Celebrate endings—for they precede new beginnings."

ACKNOWLEDGEMENTS

This project and the work that came with it wouldn't have been realized without the support and guidance of many great people.

First of all, I would like to thank Professor Georgios C. Vosniakos for giving me the opportunity to be part of his team and for his continuous support since my undergraduate years. He has served as both an advisor and mentor for me, guiding me all the way towards completion of this diploma thesis. I highly appreciate his trust in appointing me this project during my first steps in the world of 'Reinforcement Learning'.

Moreover, I am grateful to have met and closely worked with Manufacturing Technology Lab Postdoctoral researcher Manolis Stathatos, who was always keen on helping me overcome the difficulties that I encountered throughout the project. Collaborating with him was a great joy for me. The knowledge gained along his side has motivated me to pursue machine learning and RL further, diving deeper into this fascinating field. I would thus like to address a sincere thank you towards Dr. Stathatos for devoting his time to helping, advising, and supporting me throughout my work, even though he was not obliged to.

Beside the people in academia, I would like to wholeheartedly thank my friends who have been an integral part of my years in NTUA and who have always been supportive and helpful in their own unique way. I wouldn't have been the person I am without them.

Finally, I would like to express my gratitude to my parents and siblings for always believing in me and for providing me with their valuable support and love throughout my studies. My academic journey, as of now, would have not been realized without their contribution and non-stop motivation - and for this I will always be grateful.

ABSTRACT

The industrial environment of the past years has been characterized by the high rate of changes, pushing the industry to implement innovative technologies to satisfy the market needs. It is of high priority to modify existing production systems in order to adopt to the new era and optimize product control. Industry 4.0, the conjunction of Information Technology with well-established means of mass production, is gaining ground each passing day. Detailed simulations simplify the analysis processes and thus allow for a better understanding of the systems, while at the same time decreasing costs and leading to broad optimization possibilities. Unmanned aerial vehicles (UAVs) have also witnessed increasing use, as they provide flexibility and agility to the manufacturing processes, meeting the enhanced efficiency demands. The environmental impact and economic restraints pave the way towards on-demand manufacturing, so that the transportation scheme is also transforming to an efficient dispatching network, where dynamic and stochastic environments are required. Quick response to customer orders and the ability to work over a disparate set of floor settings is of high priority.

In this study, we address this problem of dispatching in manufacturing. A design to formulate the shop floor state as a virtual discrete events model using Reinforcement Learning (RL) is proposed. Machine processing times, transition times, queue warehouse condition and drone battery are incorporated in the RL state representation, while a reward function is defined in terms of production maximization. The influence of different model parameters and hyper-parameter optimization on the quality and stability of the results obtained is analyzed. A well-configured Proximal Policy Optimization (PPO) algorithm, that enables optimal part dispatching within the Flexible Manufacturing System is also discussed. The software package MATLAB SimEVents is used for modeling the manufacturing cell environment, interfaced with the MATLAB RL Toolbox for agent creation and training. The results obtained suggest a robust approach to optimizing part dispatching in production lines.

Table of Contents

Figure Inventory

Table Inventory

Diagram Inventory

1 INTRODUCTION

1.1 Flexible Manufacturing Systems (FMS)

A flexible manufacturing system is a manufacturing system that contains enough flexibility to allow the system to rapidly react to production changes. This flexibility is generally considered to fall into two categories. The first is machine flexibility. This allows the system to be changed to produce new product types and to change the order of operations executed on a part. The second is routing flexibility. This consists of the ability to use multiple machines to perform the same operation on a part as well as the system's ability to absorb large-scale changes, such as in volume, capacity, or capability [1].

Most FMS systems consist of three main systems: a material handling system to optimize the flow of parts, a central control computer that controls material movement, and the working machines, which are often automated machines or robots.

Control optimization, material flow efficiency, setup efficiency, and data flow efficiency can be achieved by using Flexible Manufacturing Systems. Economical machining is achieved by:

- Exploiting the flexibility and productivity of numerically controlled machine tools to produce smaller and medium sized lots,
- Utilizing costly production equipment more effectively through the reduction or elimination of setups,
- Changing parts, tools, and machining programs automatically [2].

1.2 Drones in Manufacturing

Recent advancement in technologies challenge the way companies manufacture and deliver products. For example, additive manufacturing technologies change the processes used to manufacture customized products, collaborative robot technologies enable new assembly processes, augmented reality technologies offer new ways to train operators and artificial intelligence replaces or assists human operators in customer service processes [3]. Among these promising technologies are also the unmanned aerial vehicles (UAVs), commonly known as drones. Over the past decade, the capability of drone technology has improved, its price has plummeted, and its availability has greatly increased, thus leading to its usage in a wide variety of practical and profitable applications.

Examples of such applications include support of first responders, surveillance, express delivery, aerial photography, agriculture, and manufacturing system operations, aiming to

contribute to further versatility and efficiency within FMS settings [4]. In the case of manufacturing applications specifically, drones have come to replace or supplement industrial robots, transportation systems, ground vehicles and humans working as operators in manufacturing plants. The main tasks to be executed by drones include pickup of a part or tool from a designated storage machine, takeoff, transportation from and to a workstation, landing and item dispatching [5].

This ever expanding technology comes with several advantages for its applications, such as flexible deployment on demand, large coverage, and stationary hovering at any time, which usually produces special effects when performing tasks [4] . With regards to manufacturing applications in particular, these potential benefits are related with improved throughput and higher system productivity, reduced work-in-process inventories, and increased capacity scalability of the production system. The latter is an inherent characteristic of agile manufacturing systems, associated with the possibility for rapidly and cost-effectively adjusting production capacity in discrete time intervals, entailing layout reconfiguration in response to dynamically changing market demands impacting on production volume and product variety [5].

However, there are also several drawbacks and limitations to drone capabilities and subsequent usage. The limited amount of flight time that can be achieved constitutes a major issue to be tackled. For small drones, the best power supply solution is through Lithium Polymer (LiPo) batteries with 3 - 6 cells in series, which not only increase drone efficiency, but also contribute to an overall low weight of the UAV system. Even so, 30 minutes of airtime is usually the limit. Additional disadvantages include the payload, gripping/placing movements and navigation of the drones [5] .

1.3 Digital Factory Technologies

Industry 4.0, also called “Smart Factory,” aims to increase factory productivity and the efficient utilization of resources in real time [6], [7]. These objectives are achieved via flexible event-driven reactions to changes in the factory environment, resource allocation, scheduling, optimization, and control in real time. Most of the “Smart Factory” concepts share the attributes of cyber-physical systems (CPS) for monitoring physical processes by creating a virtual copy of the physical world and making decentralized decisions [8]. CPS is particularly defined as a transformative technology for managing interconnected systems according to their physical assets and computational capabilities. Recent developments have improved the availability and affordability of sensors, data acquisition systems, and computer networks, making CPS a cost-effective concept widely used for simulations and dynamic environment digital recreation. The competitive nature of current industry is forcing more factories to implement high-tech methods. Thus, the increasing use of sensors, radio frequency identification (RFID), and networked machines has resulted in the continuous generation of high-volume data known as Big Data [9], [10]. In this environment, CPS can be developed further to manage Big Data and exploit the interconnectivity among machines to fulfill the goal of producing intelligent, resilient, and self-adaptable machines. Furthermore, by integrating CPS with production, logistics, and services in current industrial practices, it will be possible to transform current factories into Industry 4.0 factories with significant economic potential. Therefore, it is crucial to consider adaptive scheduling and control for dynamic manufacturing environments as key research issues.

1.4 Reinforcement Learning Applications

Reinforcement learning (RL) is one of the most remarkable branches of machine learning and attracts the attention of researchers from numerous fields. Especially in recent years, the RL methods have been applied to machine scheduling problems and are among the top five most encouraging methods for scheduling literature. Current state-of-the art literature on reinforcement learning applications in manufacturing primarily focuses on two fields, which are production scheduling and robotics.

An ever-growing tendency to employ RL algorithms is particularly noted in dynamic production scheduling. Research is concentrated on addressing flexible job-shop production [11] and adaptive order dispatching [12], while [13] studies the incorporation of real-time demand information with a manufacturer's resource information -including workforce data, machine capacity and condition information, among others - to optimally schedule manufacturing processes with multiple objectives. According to [14], introducing RL to the scheduling/dispatching decisions in production systems significantly reduces the average lead time of production orders in a dynamic environment. Moreover, it has been shown that as the complexity of the production environment increases, the application of RL for dynamic scheduling becomes more and more beneficial, making the future production systems more flexible and adaptive [12].

In the field of robotics applications of RL, [15] combines reinforcement and limitation learning for solving dexterous manipulation tasks from pixels, whereas [16] presents a high-performing RL algorithm for learning robot navigation policies. Optimization of a decentralized sensor level collision avoidance policy with RL has also been studied by [17].

RL seems to provide promising results in applications concerning the control of a single or a swarm of UAVs. The need for higher accuracy and better robustness led to the design of deep reinforcement learning (DRL) control methods. Therefore, DRL opens the door for new control tasks that were unachievable with the previous RL algorithms [18]. As the UAV needs to perform real-time tasks while working in a dynamic environment without centralized control, it needs to learn tasks according to real-time data. Reinforcement learning has the ability to carry out real-time learning and decision making based on the environment, which is an appropriate and feasible method for the task scheduling of UAV clusters [4].

1.5 Objectives and Goals of the Thesis Project

Current research constitutes a thorough investigation of reinforcement learning for optimal part dispatching within a flexible manufacturing system setting, which uses drones to service the individual workstations and complete work orders.

Specifically, in the frameworks of this study, the material covered includes:

- Study of Reinforcement Learning (RL) methods for application to machine scheduling and dispatching problems for selection of an appropriate RL algorithm approach.
- Modeling of the environment of the manufacturing cell, including the machines, products, workflow, and general workspace, using the SimEvents toolbox provided by Matlab as an interface with Simulink.
- Development of an agent-based method for independent learning and subsequent decision making of the drones with regards to servicing the manufacturing cell and covering their battery needs, when necessary.
- Training of the created agent under different manufacturing cell conditions and algorithm hyperparameters to test robustness.
- Simulation of different manufacturing cell workflows and assessment of the behavior of the RL methods employed with regards to production efficiency.

The goals of the study are:

- Simplification of production scheduling in dynamic environments by real-time learning and decision making based on the surrounding environment.
- Improving system adaptability to changing manufacturing system parameters by drifting from the conventional, rule-based approach of heuristics that becomes computationally inefficient and obscure for systems of increasing complexity.
- Reducing production bottlenecks and increasing manufacturing cell efficiency by training an agent in a virtual environment, where errors do not have physical consequences and can be corrected, saving time and preventing machine damage or costly failures during production.

2 REINFORCEMENT LEARNING

2.1 What is Reinforcement Learning?

Machine learning methods may be grouped into three categories: supervised learning, unsupervised learning, and reinforcement learning [21]. Supervised and unsupervised learning methods typically require large data sets, either to train the learning algorithm for the former method or to provide sufficient information for the algorithm to learn the pattern or structure in the data for the latter. In contrast, Reinforcement Learning (RL) does not require large data sets; instead, in RL the data is generated during the learning process in the form of rewards for actions taken, which serve as evaluative feedback to a learning agent [22]. RL has increased in importance in recent years because of its applicability in a wide range of disciplines for which identifying all possible examples to train the agent is difficult; those applications include robotics, gaming, computer vision, business, control system engineering, simulation-based optimization, networks, anti-crime operations [23] and cybersecurity [24].

The central idea of RL is that the learning agent learns, over time, what action to take in any given situation by a process analogous to methodical trial and error. The learning agent tries the different actions available to it in different situations and evaluates the outcome of each action, both in terms of the action's immediate effect on the environment and its long-term contribution to the learning agent's overall goals.

2.2 Components of Reinforcement Learning

In its basic form, RL can be understood in terms of five key elements, the relationship between which is described in **Figure 1**. These core elements can be summed as follows:

- **Environment:** The artificial space where the simulation is taking place.
- **Agent:** The learner, which in the case of DRL is a neural network, that accumulates experience by interacting with the environment.
- **Observation:** One particular instance of the environment which the agent is able to perceive.
- **Action:** The way that the agent interacts with its environment.
- **Reward:** A numeric signal produced by the environment after an action made by the agent.

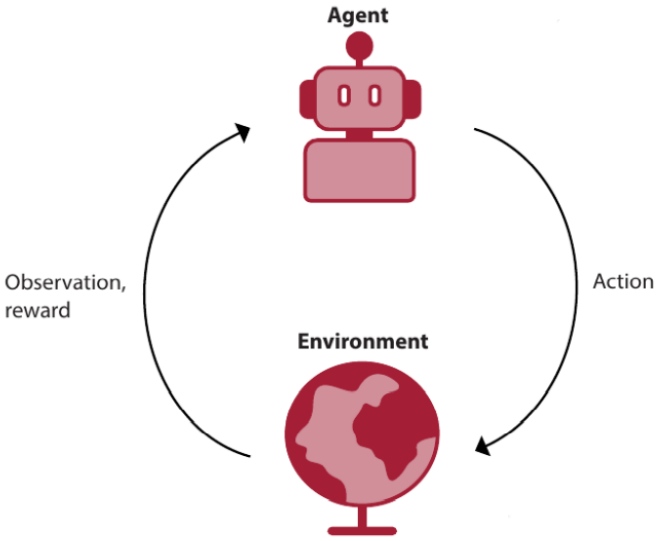

Figure 1: Components of Reinforcement Learning (RL)

2.2.1 Environment

An environment in RL is anything which is outside the learning agent in the RL problem. It is an entity that tries to encapsulate all the inner processes taking place during a decision-making problem. Reinforcement Learning environments are modelled by a mathematical framework known as Markov Decision Processes (MDPs) and this is a common core assumption that is made even if it is not explicitly stated.

In a reinforcement learning scenario, where an agent is trained to complete a task, the environment models the dynamics with which the agent interacts. It specifically receives actions from the agent, outputs observations in response to the actions and finally generates a reward measuring how well the action contributes to achieving the task. Usually, the environment is portrayed to include a well-defined task, the goal of which is defined through the reward signal.

2.2.2 Agent

The goal of reinforcement learning is to train an agent to complete a task within an uncertain environment. The agent is the learner, which learns to map states to actions by attempting actions and interacting with the environment. The agent may or may not have full access to the actual state of the environment, however it can observe a part of the environment. This component is the actual decision maker in a RL simulation.

The agent contains two components: a policy and a learning algorithm.

- The policy is a mapping that selects actions based on the observations from the environment. Typically, the policy is a function approximator with tunable parameters, such as a deep neural network.
- The learning algorithm continuously updates the policy parameters based on the actions, observations, and rewards. The goal of the learning algorithm is to find an optimal policy that maximizes the expected cumulative long-term reward received during the task.

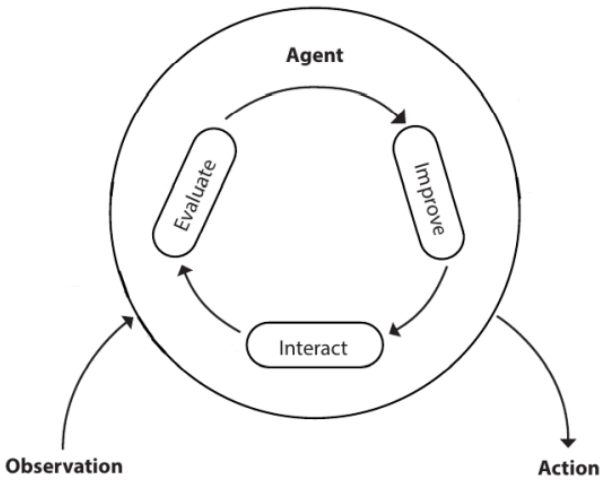

Figure 2: Characteristics of an agent

Depending on the learning algorithm, an agent maintains one or more parameterized function approximators for training the policy. Approximators can be used in two ways.

i. **Critic** — For a given observation and action, a critic returns as output the expected value of the cumulative long-term reward for the task.
ii. **Actor** — For a given observation, an actor returns as output the action that maximizes the expected cumulative long-term reward.

Agents that use only critics to select their actions rely on an indirect policy representation. These agents are also referred to as value-based, and they use an approximator to represent a value function or Q-value function. In general, these agents work better with discrete action spaces but can become computationally expensive for continuous action spaces.

Agents that use only actors to select their actions rely on a direct policy representation. These agents are also referred to as policy-based. The policy can be either deterministic or stochastic. In general, these agents are simpler and can handle continuous action spaces, though the training algorithm can be sensitive to noisy measurement and can converge on local minima.

Agents that use both an actor and a critic are referred to as actor-critic agents. Actor–critic algorithms learn both policies and value functions as depicted in Figure 3.

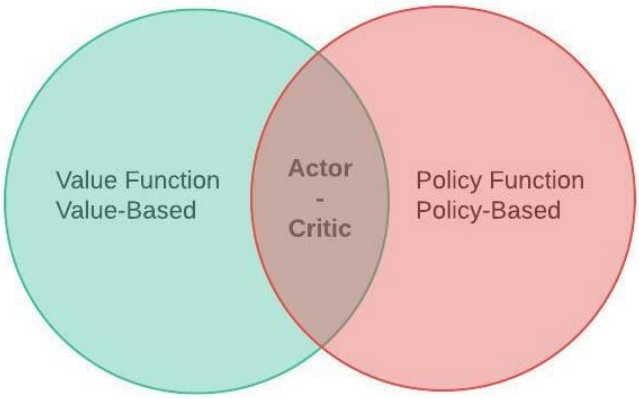

Figure 3: Value-based vs Policy-based agents

In these agents, during training, the actor learns the best action to take using feedback from the critic instead of using the reward directly. At the same time, the critic learns the value function from the rewards so that it can properly criticize the actor. Simply put, the 'actor' is the component that learns policies, and the 'critic' is the component that learns about whatever policy is currently being followed by the actor in order to 'criticize' the actor's action choices [22].

In general, these agents can handle both discrete and continuous action spaces.

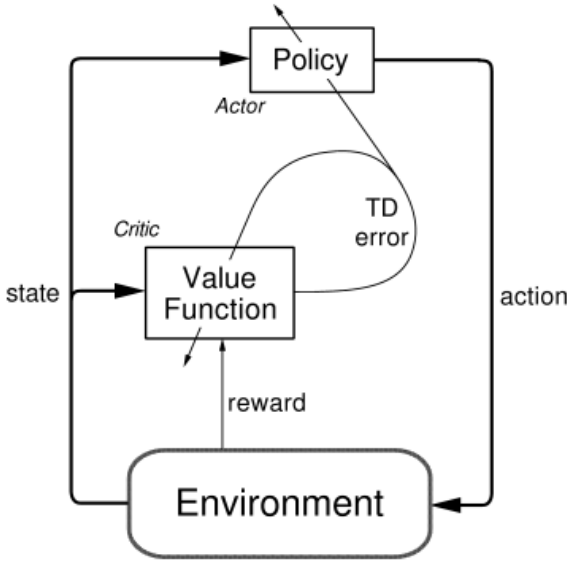

Figure 4: Representation of the actor-critic algorithm framework in RL

2.2.3 State/ Observation

The environment is represented by a set of variables that are related to a specific problem. The combination of all the possible values that this set of variables can take is called the state space, denoted as the set S. The state space can be finite or infinite. A single state however must always be finite, and its variables must be of a constant size from state to state.
The set of variables the agent can perceive at any given time step t is called an observation. The combination of all the values of the variables that the agent can observe is called the observation space. The observation is the agent's internal representation or abstraction of a particular situation or configuration of the environment as perceived by the agent.

In the case of MDPs the states are fully observable, meaning that the agent can see the internal state of the environment at each time step t. In this case states and observations are the same.

States should contain all the variables necessary to make them independent of all other states. This means that the probability of the next state, given the current state and action, is independent of the history of interactions of the agent with the environment. This is the memoryless property of MDPs also known as the Markov property and it is expressed as:

$$P\,(S_{t+1} \mid S_t\,, A_t) \;=\; P\,(S_{t+1} \mid S_t\,, A_t\,, S_{t-1}, A_{t-1}, \dots)$$

The above formula states that the probability of moving from one state S_t to another state S_{t+1} on two separate occasions given the same action A_t will be the same regardless of the previous states and actions the agent encountered up to that point. The set of all states in an

MDP is denoted as S^+ and the set of initial states as S^i . To begin interaction with an MDP, a state S^i must be drawn from a probability distribution which must be fixed throughout training, meaning that the probabilities of picking a particular initial state must be the same for all episodes.

2.2.4 Action

At every state, the environment allows a set of actions that the agent can take to interact with it. Usually, the set of actions is the same for all states, this is not however a requirement. The set of all possible actions that an agent can take across all the possible states of the environment is called the action space $A(s)$. Just as with the state the action space can be finite or infinite. The set of variables related to a given action can contain more than one element and must be finite. One distinction between the state and action variables is that unlike the number of states variables that should be kept constant at each time step, the number of action variables need not be constant. That means that the actions available in a state might change depending on that state.

The environment is responsible for making all the available actions for a given state known in advance. Agents might select actions either deterministically or stochastically. This means that agents can select actions either by looking up a specific action for a specific state or by selecting an action from a per state action probability distribution.

Through its actions, the agent attempts to influence the change of the environment, which will change states as a response to these actions. The function that is responsible for this change is called the transition function. After a transition is made, the environment produces a new observation, while it might also provide a reward signal as its response to the most recent action taken.

2.2.5 Reward and Discount

The reward function $R(s)$ is an external signal from the environment in response to an action which tells the agent how successful an action is with respect to completing the task goal. The reward is a method of communicating the objective of the RL problem to the agent. In general, the agent seeks to select actions that maximize its long-term cumulative reward, which is expressed as a scalar value.

Reward signals can be a) dense, meaning that they appear often during the agent-environment interaction or b) sparse meaning they seldom appear or something in between. More dense rewards lead to greater supervision on the agent and faster training but may inject bias to the agent's strategy. One the other hand, sparse rewards allow for greater exploration and possibly unexpected emerging behaviors but might lead to longer training duration.

Mathematically, the reward function can be described as a function that takes in a state-action pair and outputs the expectation of the reward at a time step t given the state-action pair in the previous time step $t-1$.

$$r(s,a) = E\left[R_t \mid S_{t-1} = s, A_{t-1} = a\right]$$

The reward at time step t comes from a set of all rewards R, which is a subset of all real numbers $R_t \in R \subset \mathbb{R}$.

Another critical component regarding rewarding the agent is the discount. Since there is a possibility of an episode being a very long sequence of time steps, a way is needed to inform the agent about the value of rewards overtime. Thus, a positive real number called discount factor or gamma (γ), which usually is less than one, is introduced. Gamma can control how the agent accounts for the importance of rewards over time. Essentially, the goal is that the agent understands that the later it receives rewards, the less attractive they are to present calculations. That means that the further into the future we receive a reward the less valuable it is in the present.

RL requires a balance between greedy selection of the action with the maximum reward, as far as is currently known, and non-greedy selection of actions with unknown or currently non-maximum rewards to learn their effectiveness. In the RL context, the former is called exploitation and the latter is called exploration. These aspects differentiate RL learning from supervised learning, in which the agent is explicitly taught.

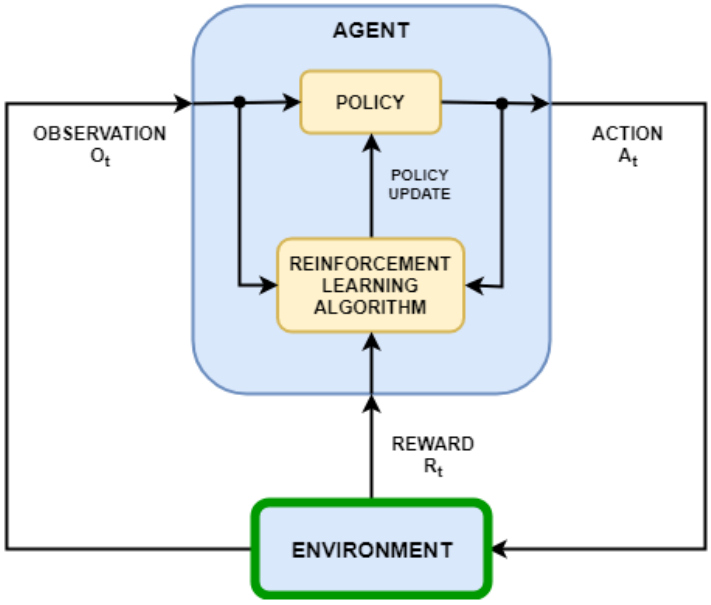

Figure 5: Schematic Interaction of components in Reinforcement Learning

2.3 The Agent-Environment Interaction Cycle

The interaction between the agent and the environment can go on for several cycles, which are called time steps, and which can vary from milliseconds to days or even years according to the application. At each time step the agent observes the environment, takes an action, and receives a new observation and reward. The task that the agent is trying to solve may or may not have a natural ending. Tasks that have a natural ending such as games are called episodic tasks whereas tasks that do not such as an agent learning to walk [25] are called continuing tasks. The sequence of time steps from the beginning to the end of an episodic task is called an episode. It is usual for agents to need several time steps and episodes to learn to solve a given task. The sum of rewards collected in a single episode is called the return. Agents are designed to maximize the return. However, there are other components that aid the learning procedure and on which the agent can often be designed. One such component is the mapping between observations and actions called policy. The agent may be also designed to learn mappings from observations to new observation or rewards called models. A final design principle is to create an agent to learn mappings from observations and possibly

actions to reward estimates which are considered part of the total episodic return. These mappings are called value functions.

Figure 6: The general workflow for training an agent using reinforcement learning

2.3.1 Policy

A policy defines the learning agent's way of behaving at a given time [26]. A policy, denoted as π, is a function that proposes actions to take for every possible given state s. It can be thought of as the universal action plan of an agent when encountering every different possible observation in the environment. A policy can be stochastic, meaning that it can return action-probability distributions or deterministic that return single actions for a given state. An effective policy is one that can adapt to the stochasticity present in an environment. This means that it is not enough for the agent to know how to act optimally at given states to obtain a maximum return. What the agent is really looking for is to maximize the expected return which is the return considering all the possible transitions due to the environment's stochasticity. So, it is obvious that methods are needed to automatically find optimal policies for a given environment. To do so, since it is not obvious at all what an optimal policy looks like, we should introduce tools used internally by agents that allow them to find optimal policies.

2.3.2 State-Value Function

The first mechanism is the state-value function denoted as V . The value function V answers the question of what the agent can expect from being in state s. Essentially, the state-value function introduces a way of comparing policies between them by assigning different numbers to states for a given policy. What is truly sought after is the expectation of returns if we follow a policy π. As the agent interacts with stochastic environments, we must account for all the possible ways the environment can react to our policy.

We shall now define the value of a state s when following a policy π as the expectation of returns if the agent follows policy π starting from state s. Calculating this for every state gives us the state-value function V . Mathematically we can write this as:

'The value of a state s under the policy π is the expectation over π of returns at time step t given that you select state s at time step t .'

$$v_\pi(s) = E_\pi [G_t \mid S_t = s]$$

2.3.3 Action-Value Function

The action-value represents the value of taking action α in a state s and it is utilized towards deciding between actions. Comparing different actions under the same policy may lead to selecting better actions, and thus acquiring an improved policy. The action-value function -

also known as Q-function - captures precisely the above. It represents the expected return when the agent follows a policy π after taking an action α in a state s. The Q-function essentially captures the dynamics of the environment, since it allows for the comparison of the results of the different possible actions that can be taken. Mathematically, the Q-function can be expressed as:

'The value of action α in a state *s under policy π is the expectation of returns given we select action α* in a state *s and follow policy π.'*

$$q_\pi(s,a) \;=\; E_\pi\,[G_t|\,S_t \;=\; s, A_t \;=\; a]$$

2.3.1 Optimality

All the components discussed so far are used to describe, evaluate, and improve the behavior of an agent. The situation where these components achieve their greatest values is called optimality.

An optimal policy can, for every possible state, obtain expected returns that are greater than or equal to any expected returns provided by any other policy.

An optimal state-value function is the one with the maximum value for all states across all policies. Mathematically, this can be expressed as:

$$v_*(s) = \max_\pi v_\pi(s), \forall\, s \;\in S$$

An optimal action-value function is the one with the maximum value across all policies for all state-action pairs. The mathematical interpretation can be written as:

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall\, s \;\in S\,, \forall\, a \;\in A(s)$$

2.4 Policy - Gradient Methods

Having defined all tools necessary for evaluating policies in the previous section we will now introduce the concept of Policy-Gradient Methods. These are the methods used to optimize the policies of agents by parametrizing them and iteratively adjusting them through learning procedures to maximize expected returns.

In policy-gradient methods we are trying to maximize a performance objective. This objective is to maximize the performance of a parametrized policy, so we are running gradient ascent or regular gradient descent on the negative performance. The performance discussed pertains to the expected total discounted reward from the initial state or in other words the expected state-value function from all initial states of a given policy. The objective is to maximize the expected value of the true value-function under the parametrized policy $\pi_{\theta i}$ from all initial states which can be defined as follows:

$$J_i(\theta_i) \;=\; E_{s_0 \sim p_0}\,[\,v_{\pi_{\theta i}}\,(s_0)\,]$$

Policy-gradient methods are a great tool for learning stochastic policies. Stochastic policies are policies that prescribe a probability distribution of different actions a for a particular state s. Moreover, learning stochastic policies efficiently allows us to have better performance under partially observable environments. The logic behind this is that since we can learn arbitrary action probability distributions, the agent is less dependent on the Markov

assumption. For example, if the agent cannot distinguish a handful of states from their emitted observations, the best strategy is often to act randomly with specific probabilities. Another feature of learning stochastic policies through policy-gradient methods is that the learning algorithm allows us to approach a deterministic policy for specific states. This phenomenon occurs since the exploration ability of the agent is embedded in the learning function and thus converging to a deterministic policy for a given state is possible while training.

When choosing an agent, a best practice is to start with a simpler and faster to train algorithm that is compatible with the action and observation spaces of the problem at hand, before progressively going to more complicated algorithms. Specifically, following cases can be distinguished:

i. ***Discrete action and observation spaces***

For environments with discrete action and observation spaces, the Q-learning [22, Ch. 6.5] and SARSA [22, Ch. 6.4] agents are the simplest compatible agent, followed by Deep Q -Network (DQN) [22], Proximal Policy Optimization (PPO) [27], and Trust Region Policy Optimization (TRPO) [27] (**Figure 7**).

Figure 7: Policy-gradient methods for discrete action and observation spaces

ii. ***Discrete action and continuous observation space***

For environments with a discrete action space and a continuous observation space, DQN is the simplest compatible agent followed by PPO and then TRPO (**Figure 8**).

Figure 8: Policy-gradient methods for discrete action and continuous observation space

iii. ***Continuous action space***

For environments with both a continuous action and observation space, Deep Deterministic Policy Gradient (DDPG) [28] is the simplest compatible agent, followed by Twin Delayed DDPG (TD3) [29], PPO, and Stochastic Actor-Critic (SAC)[30] , which are then followed by TRPO (**Figure 9**). In general:

- TD3 is an improved, more complex version of DDPG.
- PPO has more stable updates but requires more training.
- SAC is an improved, more complex version of DDPG that generates stochastic policies.
- TRPO is a more complex version of PPO that is more robust for deterministic environments with fewer observations.

Figure 9: Policy-gradient methods for continuous action space

Policy gradient methods are fundamental to recent breakthroughs in using deep neural networks for control. However, achieving successful results via policy gradient methods is often challenging, as they tend to be sensitive to the choice of step size — too small and progress is slow, while too large and the signal is overwhelmed by the noise. They also often have very poor sample efficiency, requiring millions of timesteps to learn simple tasks.

The central idea of RL is that the agent learns to influence or gain control of an environment over which it initially has no control by improving its policy. A policy which is better than or equal to all other policies for the same problem is called an optimal policy. Finding an optimal policy is difficult for most practical problems, usually due to the lack of computational resources to do an exhaustive search and unknown environment dynamics. Hence, for most practical purposes an improved or approximately optimal policy is the goal.

Details and extensive study of the separate policy methods fall beyond the scope of the current work. Therefore, attention will be given to PPO, the policy gradient method employed throughout the case study discussed in Chapter 4. Although TRPO tends to be more robust than PPO if the environment dynamics are deterministic and the number of observations is low, PPO is able to achieve a balance between ease of implementation and sample complexity, making it the most prominent method to use for the case of the manufacturing system studied.

2.4.1 Proximal Policy Optimization (PPO)

PPO is a policy gradient method that makes policy updates using a surrogate loss function to avoid catastrophic drops in performance [27]. This algorithm is a type of policy gradient training that alternates between sampling data through environmental interaction and optimizing a clipped surrogate objective function using stochastic gradient descent for computing policy updates. The surrogate objective regularizes large policy updates and essentially improves training stability by limiting the size of the policy change at each step [27]. In that way, each policy update step remains within a close neighborhood around the previous-iteration policy [31]. The validity of the surrogate objective, which is based on data collected from the previous-iteration policy, is thus increased.

To estimate the policy and value function, a PPO agent maintains two function approximators.

- **Actor $\boldsymbol{\pi(A|S;\theta)}$** — The actor, with parameters θ, outputs the conditional probability of taking each action A when in state S as one of the following:
    - Discrete action space — The probability of taking each discrete action. The sum of these probabilities across all actions is 1.
    - Continuous action space — The mean and standard deviation of the Gaussian probability distribution for each continuous action.

- **Critic $V(S;\phi)$** — The critic, with parameters ϕ, takes observation S and returns the corresponding expectation of the discounted long-term reward.

During training, the agent tunes the parameter values in θ. After training, the parameters remain at their tuned value and the trained actor function approximator is stored in $\pi(A|S)$.

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Figure 10: PPO-Clip pseudocode implementation [32]

PPO is a simplified version of TRPO, which is generally more computationally expensive than PPO. The PPO algorithm is robust in that hyperparameter initializations are not as sensitive, whereas it can work out of the box on a wide variety of RL tasks. It achieves a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. Even so, proper hyperparameter initialization and search can still lead to improved results. Hyperparameters dealing with experience collection are:

i. *Experience Horizon*
Number of steps the agent interacts with the environment before learning from its experience, specified as a positive integer.

ii. *Mini Batch Size*
Mini-batch size is treated as the training trajectory length. It is used for each learning epoch, specified as a positive integer. Its range can be smaller or equal to the Experience Horizon.

iii. *Number of Epochs*
Number of epochs for which the actor and critic networks learn from the current experience set, specified as a positive integer.

Other hyperparameters include:

iv. *Entropy loss weight*
To promote agent environment exploration, an entropy loss term $w\mathcal{H}_i(\theta, \mathrm{S}_i)$ can be added to the actor loss function [28] , where w is the entropy loss weight and $\mathcal{H}_i(\theta, \mathrm{S}_i)$ is the

entropy. Entropy loss weight is specified as a scalar value between 0 and 1. A higher entropy loss weight value promotes agent exploration by applying a penalty for being too certain about which action to take. Doing so can help the agent move out of local optima.

v. *Discount Factor*

Discount factor applied to future rewards during training, specified as a positive scalar less than or equal to 1.

vi. *Learning Rate*

It usually ranges between 0.003 to $5x10^{-6}$ and represents how fast the optimizer learns. The actor and the critic used for the algorithm may or may not have different learning rates, depending on the nature of the problem at hand.

3 MATLAB SIMEVENTS INTERFACE

3.1 Discrete Event Simulation (DES)

Discrete event simulation (DES) is a method of simulating the behavior and performance of a real-life process, facility, or system. DES models the system as a series of 'events' that occur over time, while assuming no change in the system between events [33]. Thus, the simulation time can directly jump to the occurrence time of the next event, which is called next-event time progression.

In addition to next-event time progression, there is also an alternative approach, called fixed-increment time progression, where time is broken up into small time slices and the system state is updated according to the set of events or activities happening in the time slice [34]. Because not every time slice has to be simulated, a next-event time simulation can typically run much faster than a corresponding fixed-increment time simulation.

Both forms of DES contrast with continuous simulation in which the system state is changed continuously over time on the basis of a set of differential equations defining the rates of change of state variables.

The assignment of tasks and the scheduling of a manufacturing system and its service by drones, which is the problem at hand, can be studied with the help of discrete events simulation, examining various combinations of heuristic rules.

3.2 MathWorks SimEvents

SimEvents is a discrete event simulation tool developed by MathWorks. It adds a library of graphical building blocks for modeling queuing systems to the Simulink® environment. It also adds an event-based simulation engine to the time-based simulation engine in Simulink® [35].

SimEvents provides a graphical drag-and-drop interface for building a discrete-event model. It provides libraries of entity generators, random number generators, queues, servers, graphical displays, and statistics reporting blocks. Integration with MATLAB allows customization of the process flow in a SimEvents model. A MATLAB function can be developed to represent a task-scheduling sequence, routing of parts, or production recipes in a process flow. Since the two programs are within the same tool environment, it is straightforward to generate custom random distributions of input tasks, optimize a process, as well as to generate custom statistics [36], while SimEvents and Simulink® can be used in the same simulation model to simulate hybrid or multi-domain systems that have both time-based and event-based components [37].

Some common uses of Simevents include *Process/Logistics* simulations. SimEvents is used to model process flows and logistics to understand resource availability, inventory management techniques, and the effects of arbitrary events on a mission critical network design. System capacity planning and production planning can be simulated while ascertaining its optimal performance. A SimEvents model can predict characteristics such as availability, serviceability and end-to-end latencies based on a specific maintenance schedule [38].

This work seeks to develop a Discrete Event Simulation (DES) process model which shows the internal behavior of processes in system model using Simevents.

A model is similar to but simpler than the system it represents. The major objective of a model is to enable an expert predict the effect of changes to the system. In this case, the model should be a close approximation to the real system and incorporate most of its salient features as well as not being unnecessarily complex such that it is impossible to understand and experiment with it. A good model is a meticulous tradeoff between objectivity and simplicity. Model validity techniques such as simulation allows for comparison between input conditions and model output states with respect to the system output. Basically, the model can be reconfigured and experimented with, but usually, this is impossible, too expensive or impractical to do in the system it represents owing to obvious factors. Besides, if the operation of the model can be clearly studied, the properties concerning the behavior of the actual system or its subsystem can be inferred.

In Simulink®, communication across blocks is based on signals only whereas in SimEvents, it is based on both signals and entities. Essentially, SimEvents consists of a number of libraries (**Figure 11**) containing blocks with different system functionalities.

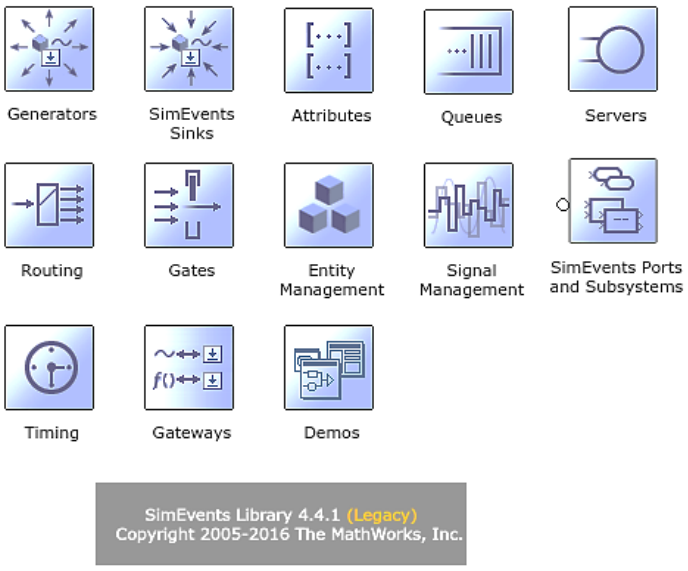

Figure 11: SimEvents libraries

The main libraries are the following:

i. Generators: Blocks which generate entities, or function calls (i.e., events that call Simulink® blocks), or random varieties.
ii. Attributes: Blocks that assign and modify data to entities. Various control actions are then made based on the values of these data, allowing blocks to differentiate between entities they process.

iii. Queues: Blocks where entities can be temporarily stored while waiting to access a resource.
iv. Servers: Blocks that model various types of resources.
v. Routing: Blocks that control the movement of entities as they access queues and servers.
vi. Gates: Blocks that control the flow of entities by enabling/disabling access of entities to certain blocks.
vii. Subsystems: These allow a combination of blocks to be executed upon occurrence of specific events; not upon Simulink® sample times.
viii. Timers and Counters: Blocks that measure event occurrence times or time elapsing between events, and blocks that count occurrences of particular event types. These data are supplied to standard display or scope blocks in Simulink® or specialized scopes designed specifically for SimEvents.

3.3 MATLAB® Reinforcement Learning (RL) Toolbox™

The Reinforcement Learning Toolbox™ provides an app, functions, and a Simulink® block for training policies using reinforcement learning algorithms, including DQN, PPO, SAC, and DDPG. These policies can be used towards implementing controllers and decision-making algorithms for complex applications such as resource allocation, robotics, and autonomous systems.

The toolbox allows for the representation of policies and value functions using deep neural networks or look-up tables and their raining through interactions with environments modeled in MATLAB® or Simulink®. A selection of single or multi-agent reinforcement learning algorithms is provided in the toolbox, while the user can also create their own. You can experiment with hyperparameter settings, monitor training progress, and simulate trained agents either interactively through the app or programmatically. To improve training performance, simulations can be run in parallel on multiple CPUs, GPUs, computer clusters, and the cloud with the Parallel Computing Toolbox™ and MATLAB® Parallel Server™.

The toolbox interfaces with Simulink®, creating a complete visual, virtual model environment and a single- or multi-agent reinforcement learning algorithm that can be controlled and trained via MATLAB®.

4 CASE STUDY: MANUFACTURING CELL MODEL

In accordance with the general workflow for training an agent using reinforcement learning presented in ***Figure 6***, we are going to gradually build up to the agent validation, starting with the formulation of the problem at hand.

4.1 Problem Statement

The problem at hand can be briefly summarized as a decision problem, pertaining to the use of reinforcement learning as a means to optimize part dispatching within an FMS that uses drones to service the different workstations and complete work-orders.

The assignment and scheduling of an FMS and its service by drones, whether considered as a single problem, or as two separate problems, can be studied with the help of discrete event simulation, examining different combinations of heuristic rules via RL implementation, and recording the result, such as the total production achieved. The goal is the optimal combination of rules to optimize decision making at each process time-step.

4.2 Creating the environment

The most important part of this work is related to the artificial environment that we created to run our simulations. The development of the environment was based on the MathWorks SimEvents library, which allows for direct interface with the MATLAB® RL toolbox. Our environment consisted of several functional components written that were responsible for depicting elements regarding our reinforcement learning simulation. These elements include components such as the observation space, the action space, and the formulation of the reward function, which will be analyzed in more detail for each case separately.

The manufacturing cell example used as inspiration for the environment of the current study is graphically presented in **Figure 12**, using the graphical interface of software ProModel. The manufacturing process presented constitutes a variation of the basic model described in Bateman et al. [39].

The plant is assumed to work on an 8-hour workday 5 days per week, producing shafts for lawnmower machines. Pieces of raw material in the form of bars are input in the system (*Stores),* where they are inspected by a dedicated operator, before being transferred to the first workstation (Saw). The Saw cuts the bars in appropriate lengths, producing 5 shafts out of each processed bar. The shafts arrive in a box queue (*SawOutQ*) with theoretically infinite capacity, where they await to be further processed.

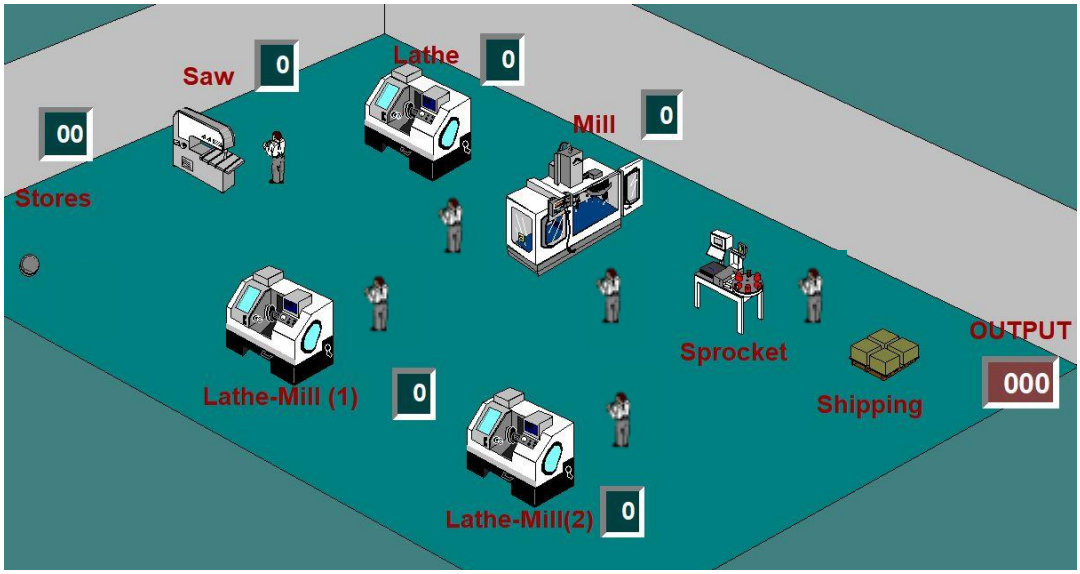

Figure 12: Conventional manufacturing cell layout as modelled in ProModel

Another operator is responsible for transferring the shafts, piece by piece, from the queue to the next workstation. The operator has overall three different workstation alternatives to choose from, depending on their availability.

i. the operator transfers a shaft to the *Lathe*, which has a shorter processing time compared to the other two existing workspaces. The *Lathe* is responsible for cutting and boring both ends of the shaft. The lathed items are then transferred in *LatheOutQ*, which is a box with a capacity of 9 items. In this case, the shafts are also transferred to the *Mill*, where they undergo some further processing.
ii. the operator transfers a shaft to the *Lathe-Mill (1)*, which has a longer processing time, but, besides cutting and boring, also includes the creation of a wedge's way on each of the shafts.
iii. the operator transfers a shaft to the Lathe-Mill (2), which has the same function as (ii), yet demands the longest processing time of all.

After completing either of the above scenarios, the processed shafts are transferred in a box with a capacity of 9 pieces (*MillOutQ*). Finally, another operator is responsible for placing a wedge in the wedge's way of each shaft from *MillOutQ* and ensuring an appropriate interlock by using a special machine (*Sprocket*).

Finally, the finished item is transferred to the packaging and shipping area (*Shipping*).

For the conventional model, the production flow happens on a ‘first-come, first-serve’ (FCFS) basis, meaning that operators are always responsible for transferring parts between immediately available workstations, irrespective of which path might be the most efficient one to increase production rates. The manufacturing cell presented above may be transformed by replacing the operators with a drone system consisting of 1 or more UAVs in the form of a swarm, as presented in **Figure 13**.

Based on the conventional model, we can recreate a new setup, closely resembling the so far know information and structure.

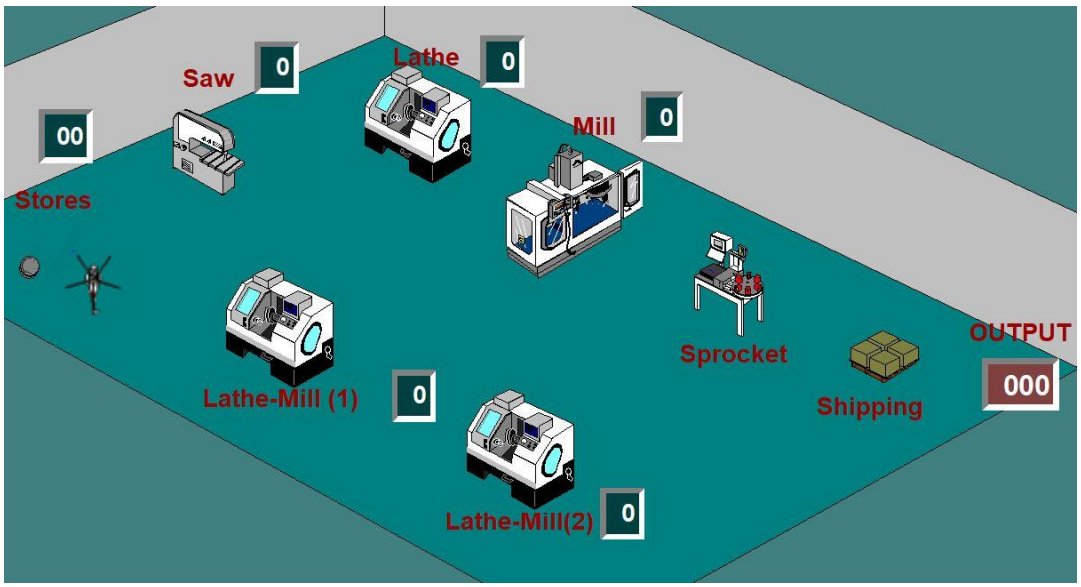

Figure 13: Proposed manufacturing cell layout as modelled in ProModel

For this case and due to the simplicity of the problem at hand, one drone is chosen to service the entire production plant, being responsible for transferring the material in between workstations and taking over the operators' duties. The choice of including 1 drone constitutes the more conservative scenario that can be tested and is meant to 'stretch' the drone capability to both cover and enhance the production rate within the manufacturing cell. Also, the available paths that the drone might follow in such systems are virtually infinite, since the drone has the capacity to move in between any of the workstations, the input and output areas in any order of moves. Some remarks to take into consideration before continuing with the modelling and the simulation of the system are the following:

1) The transition times represent the time units needed to complete a route from any point in the work cell to the workstation of interest. For simplification purposes, the transition time (TA) from and to each workstation is bidirectionally considered to be constant and workstation specific. The TA value is formed after averaging the transition times recorded for every workstation and scaling the value to match the simulation parameters.
2) The machine worktime (MA) represents the time units necessary for a specific workstation to complete the processing of a single item.
3) A metric that has to be taken into consideration is the drone battery life. The UAV(s) will need to be charged in between routes to remain functional.

The models presented below are the implementation of the proposed cell structure and are recreated with the help of the MathWorks SimEvents library. For each of the models and for visualization purposes, the color-coding legend provided in **Table 1** is used.

	Machine in Use (*1 =True / 0 = False*)
	Transit in Progress (*1 =True / 0 = False*)
	Queues Capacity (value >0 =True/0 =False)

Table 1: Color-coding legend for SimEvents models

4.2.1 FCFS model with 1 drone – No charging

The FCFS model follows the production flow imposed by Simulink® rules and gates that allow item flow via the pathway that is not blocked at the specific point in time, when the drone must make a routing decision. Therefore, all gates are set to always be 'open' and receive items whenever they are not blocked. This case does not take into account the battery charging needs of the drone, but rather assumes an ideal scenario. The cases where the charging is not involved serve merely as a proof of concept for the model configuration and the production capabilities of the manufacturing cell irrespective of relevant drone constraints.

An overview of the model is presented in **Figure 16**.

The 'Resource Pool' block in **Figure 14** represents the drones that are available at each simulation step. For the purposes of the current investigation, drone number is restricted to one (1) drone responsible to serve the whole manufacturing cell.

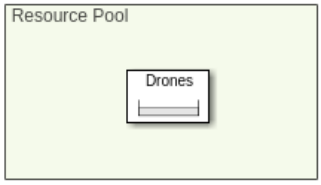

Figure 14: Resource pool representation in SimEvents

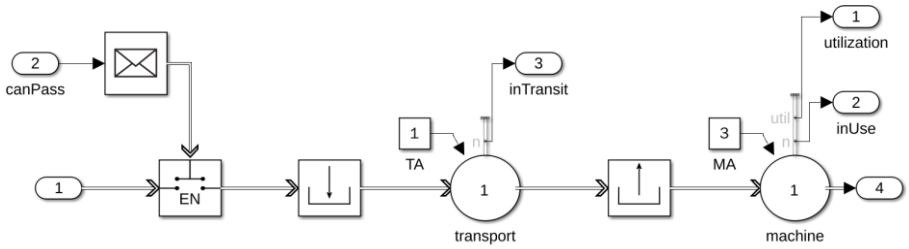

Figure 15: Configuration of representative workstation structure in SImEvents

Each block represents each of the workstations involved in the production line. All the blocks present a similar inner structure, which consists of the transition time (TA) that is necessary for the drone to move from its current position to the workstation (MA), where the production item is processed for a machine-specific time, before continuing towards the rest of the production line. For visualization purposes, the structure of block 'Machine A1' is presented in **Figure 15**, which adequately describes the structure of all the relevant workstation blocks.

The block referred to as the 'Observation/State' (**Figure 17**) monitors in terms of Boolean logic the vectors representing the workstations in use (*machineInUse*) and the drones in transit (*transitInUse*). Finally, the number of available items within each queue is represented in *queueOccupation*. The 'Observation/State' block merely serves as a monitoring of states for the FCFS model, but is a necessary component for the RL case, which is discussed later.

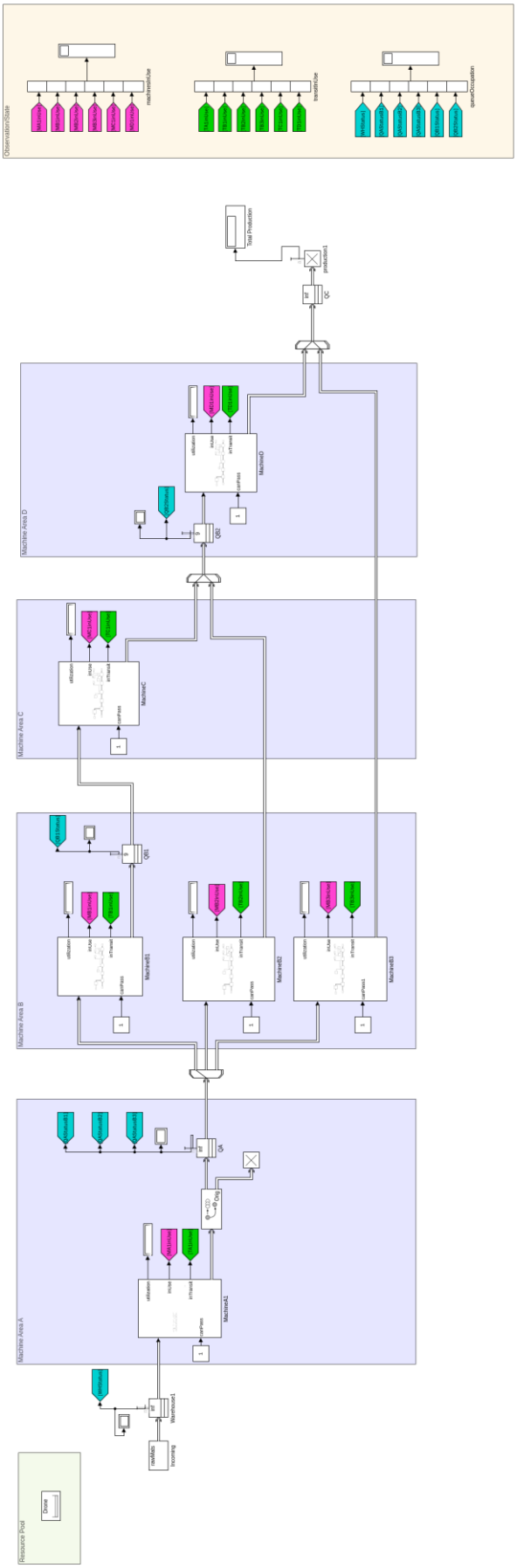

Figure 16: Structure of the FCFS model (no charging) in Matlab Simulink

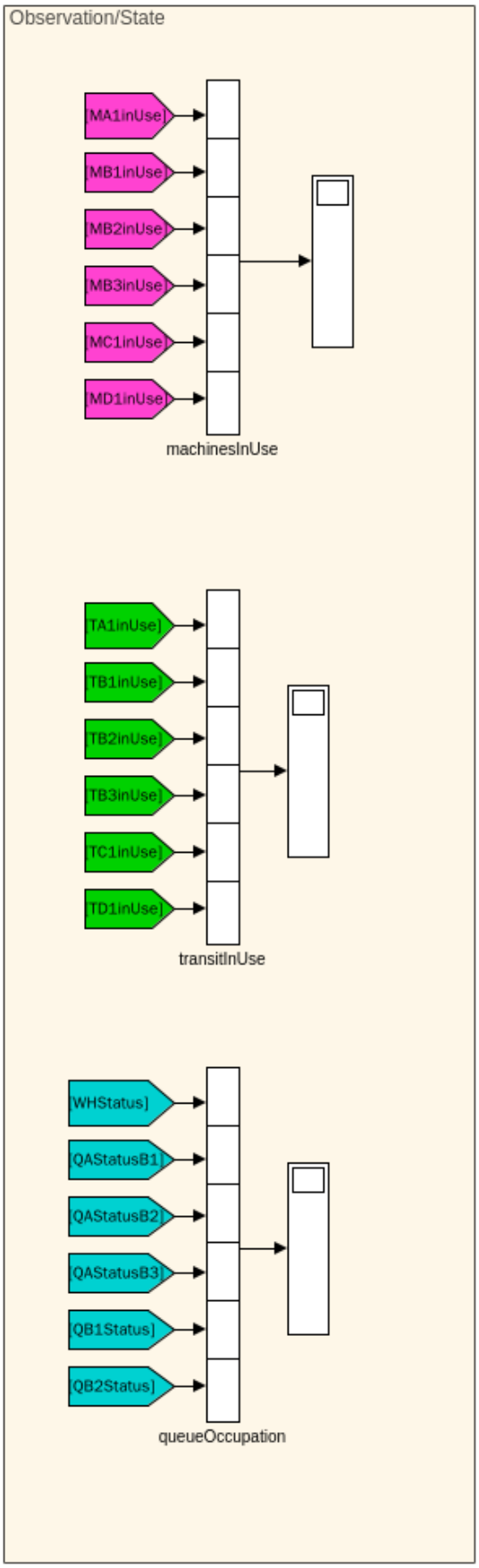

Figure 17: Observation space of the agent in SimEVents

4.2.2 FCFS model with 1 drone – With charging

An overview of the model is presented in **Figure 19**. The main model blocks remain the same as described in the previous section.

The function block now visible in **Figure 18** regulates the battery charging of the drone. Specifically for the case of the realistic representation of the FCFS model, heuristics, such as the battery charging threshold must be determined a priori, since the drone has to return to the charging station before the battery life drops to a level that does not allow for further transitions between the workstations or that could cause its depletion and interrupt production. After trial and error for testing a few hypothesis relating to battery life, the threshold was set to 50% of the drone's battery capacity. This allows for higher production rates, compared to a more stringent and conservative charging policy ($> 50\%$), while preventing the possibility of total battery drainage ($< 50\%$).

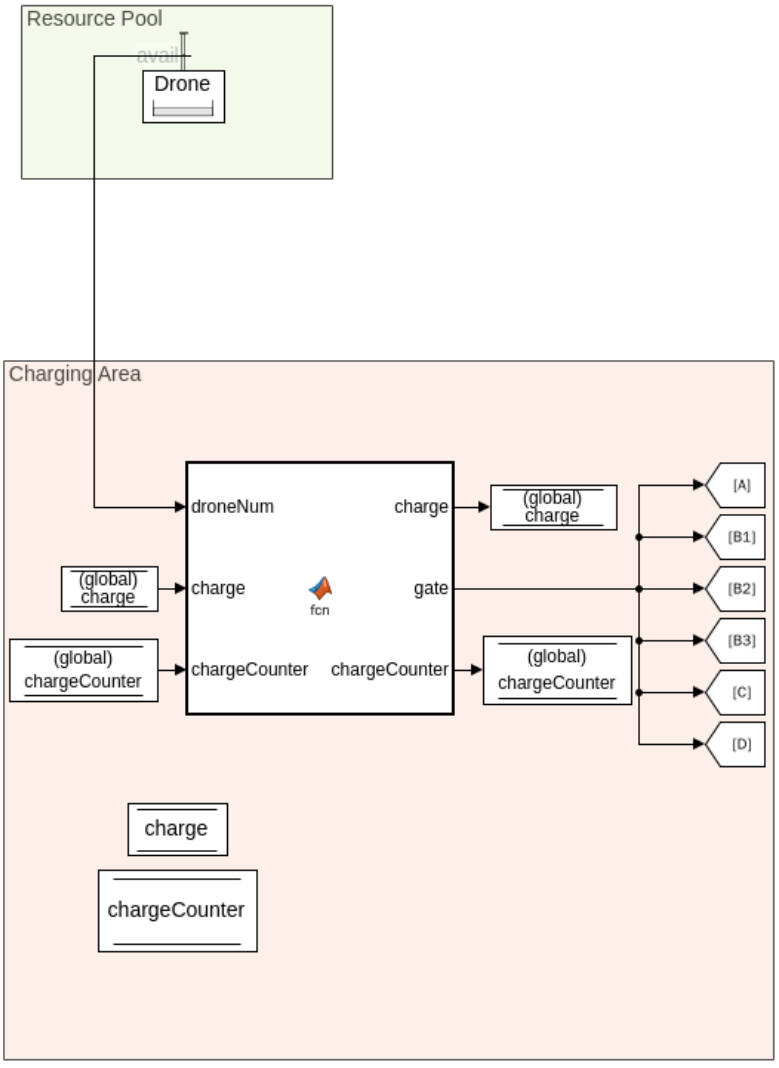

Figure 18: Implementation of a drone battery charging policy in SimEvents

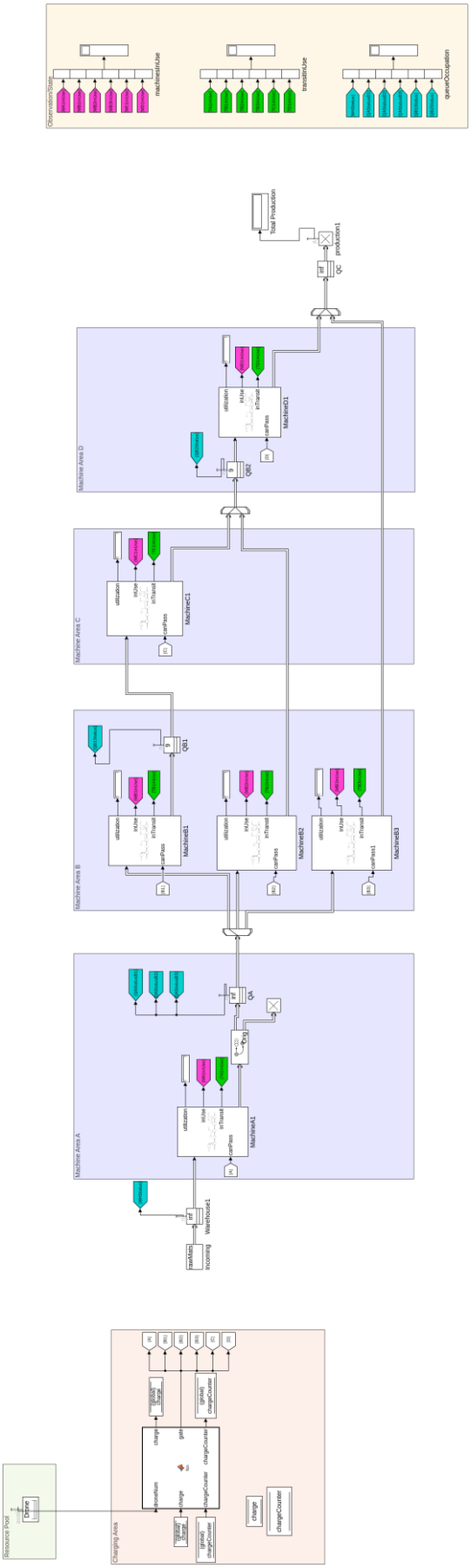

Figure 19: Structure of the FCFS model (with charging) in Matlab Simulink

4.2.3 RL model with 1 drone – No charging

The RL model follows the production flow imposed by the trained RL agent. The agent chooses its next action on the basis of maximizing the cell production. This case does not take into account the battery charging needs of the drone, but rather assumes an ideal scenario. This model again merely serve as a proof of concept for the model configuration and the production capabilities of the manufacturing cell irrespective of relevant drone constraints, when a reinforcement learning approach is employed.

An overview of the model is presented in **Figure 20** and **Figure 22**.

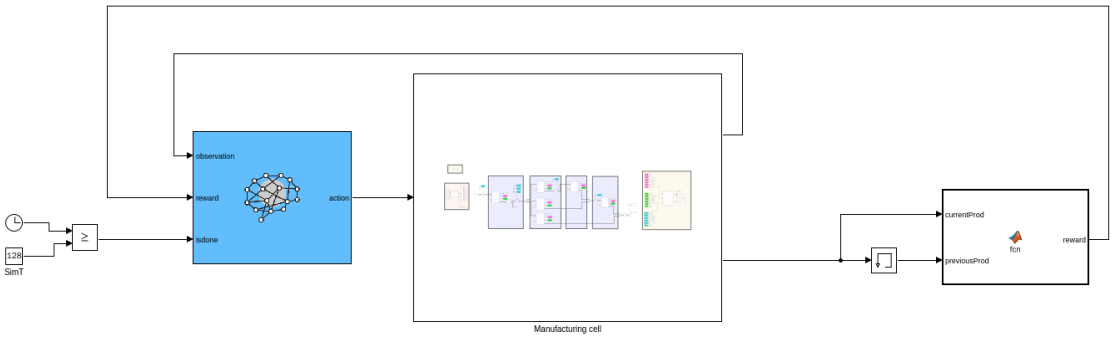

Figure 20: Interfacing of the RL agent with the the RL model (no charging) in Matlab Simulink

Training the agent for multiple episodes until it reaches a training stop criterion – usually pertaining to the stabilization of the production numbers to a desired average value – increases the confidence level that this particular agent will converge to the desired average production value. Here, in addition to the *average reward stopping criterion* of the RL agent, a stopping criterion regarding the simulation time of each training episode is introduced (**Figure 21**). Specifically, the training episode is terminated once the simulation time elapses and a new episode commences as part of the agent learning process.

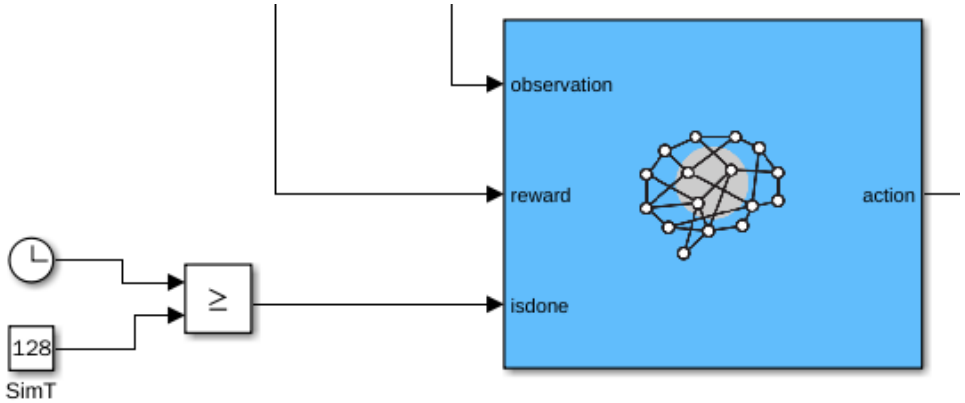

Figure 21: Detailed view of simulation stopping criteria for RL model

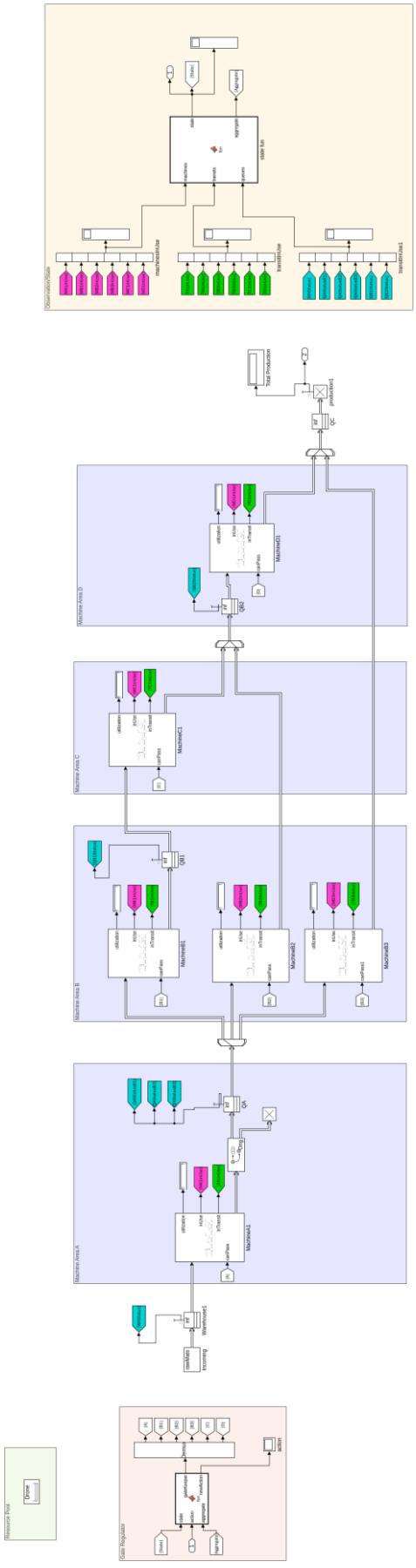

Figure 22: Structure of the RL model (no charging) in Matlab Simulink

The function block of **Figure 23** functions as 'Gate Regulator' and sends a signal to the respective workstation to be ready to be serviced by the available drone, a process modeled by opening the corresponding workstation gate.

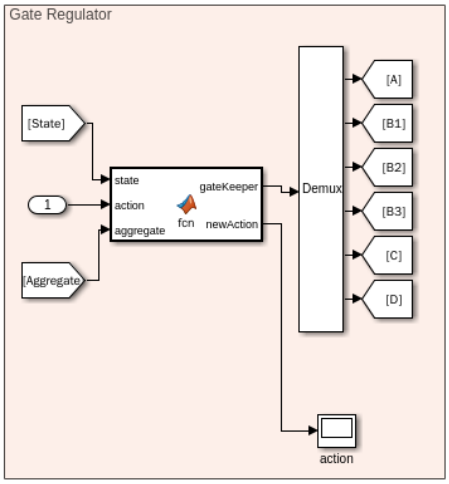

Figure 23: Implementation of a 'gate regulator' in SimEvents

For the 'Gate Keeper' function, two approaches are also carried out and compared in the results discussed in Chapter 5.

i. <u>RL-based (unconstrained) actions</u>
   The agent is left to train and decide the best course of action on its own. This means that in the case when more than one actions from the available action space are possible, the agent chooses one of them with the aim to increase its final reward (production).

ii. <u>Hard-coded (constrained/ 'illegal') actions</u>
   In that case, the user manually interferes to the final action choice of the agent by imposing a set of 'illegal actions' with direct respect to the current observation/state. Therefore, if the selected action is 'illegal', the 'Gate Keeper' decides the next action on a circular shift basis, by observing both the RL generated action and the complete observation/state of the system.

For the RL case, the 'Observation/State' (**Figure 24**) block serves as input for the RL training, following the schemes mentioned in the previous chapters. The state for this case is formed as a $1x6$ binary vector. Particularly, each vector element represents the value acquired from a logical *AND* operation between the *state (in use or not)*, the *transit (in transit or not)* and the *request (requesting or not)* conditions of each workstation involved (6 workstations). The action space of the RL model is thus also the $1x6$ vector $[1\ 2\ 3\ 4\ 5\ 6]$, each element of which represents the servicing of each of the 6 workstations in the order [A1 B1 B2 B3 C1 D1].

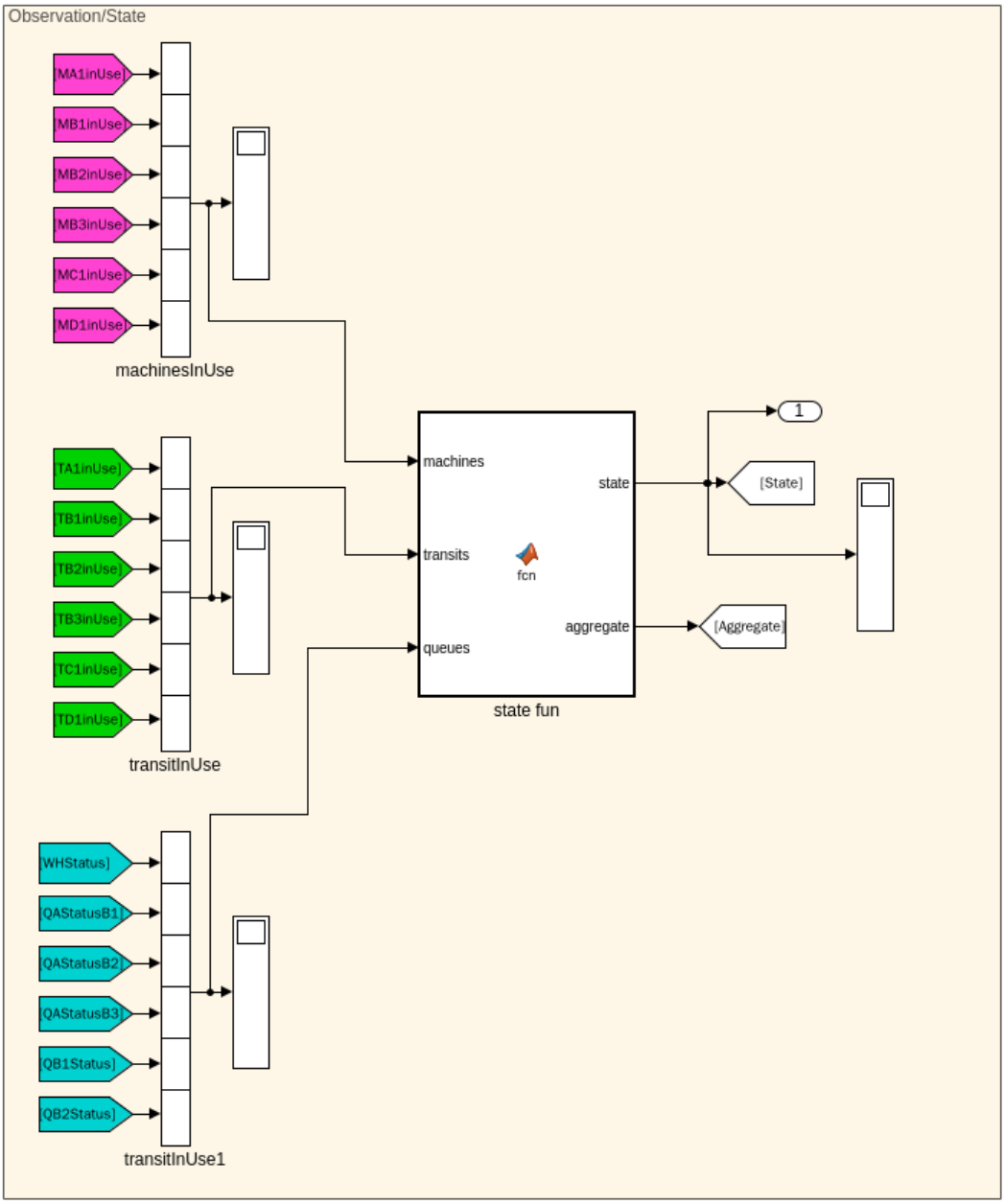

Figure 24: Observation space and state of the agent in SimEVents for the RL model (no charging)

4.2.4 RL model with 1 drone –With charging

The RL model involving the battery charging is presented in **Figure 25** and **Figure 29**.

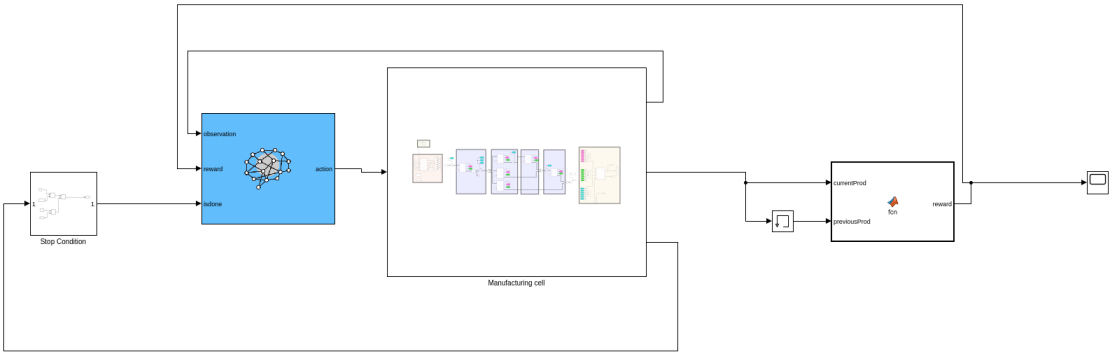

Figure 25: Interfacing of the RL agent with the the RL model (with charging) in Matlab Simulink

The Stop Criteria block added in **Figure 25** represents the model related criteria according to which the RL training is terminated. As seen in **Figure 26**, besides the simulation time stopping criteria discussed before, a training episode is also terminated immediately if the battery reaches 0% and the drone becomes inactive.

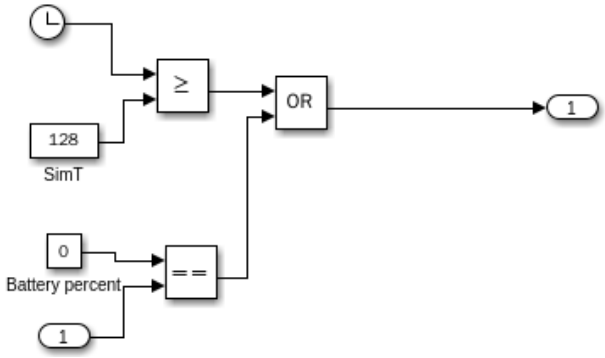

Figure 26: Simulation stopping criteria for the RL model (with charging) in Matlab Simulink

The function block visible in **Figure 27** functions as both 'Gate Keeper' and a regulator for the battery charging of the drone. In contrary to FCFS model previously presented, the agent is now responsible for optimizing drone behavior by balancing between increasing production and appropriately charging the battery. Specifically, no battery threshold is set for the RL case, but the RL agent rather learns to charge drone battery in a theoretically optimal way, while selecting between the best suited action to increase its reward every time. The cases of the *RL-based (unconstrained) actions* and the *Hard-coded (constrained/ 'illegal') actions* are examined again.

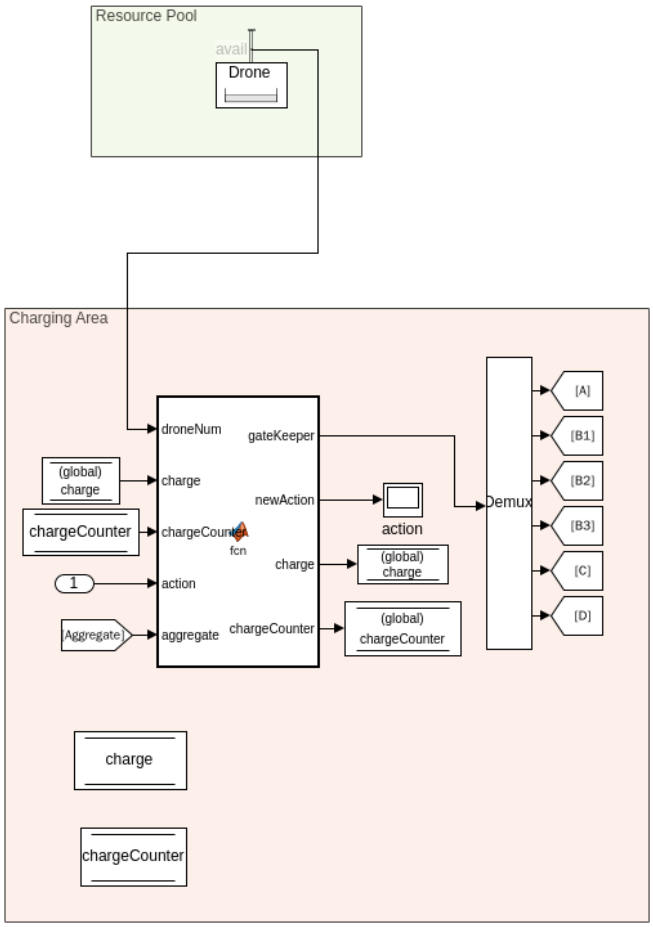

Figure 27: Implementation of a 'gate regulator' and battery charging policy in SimEvents

With respect to the 'Observation/State' (**Figure 28**), the new model now includes the drone battery charging as part of its state space. The state for this case is formed by a $1x7$ vector, where the 6 first elements are binary and remain as described previously for the no charging case, while 7th element is a real value, representing the drone battery percent at each time. The action space of the RL model thus forms into a $1x7$ vector $[1\ 2\ 3\ 4\ 5\ 6\ 7]$, each element of which represents the servicing of each of the 6 workstations and the drone charging station' in the order [A1 B1 B2 B3 C1 D1 ChargingStation].

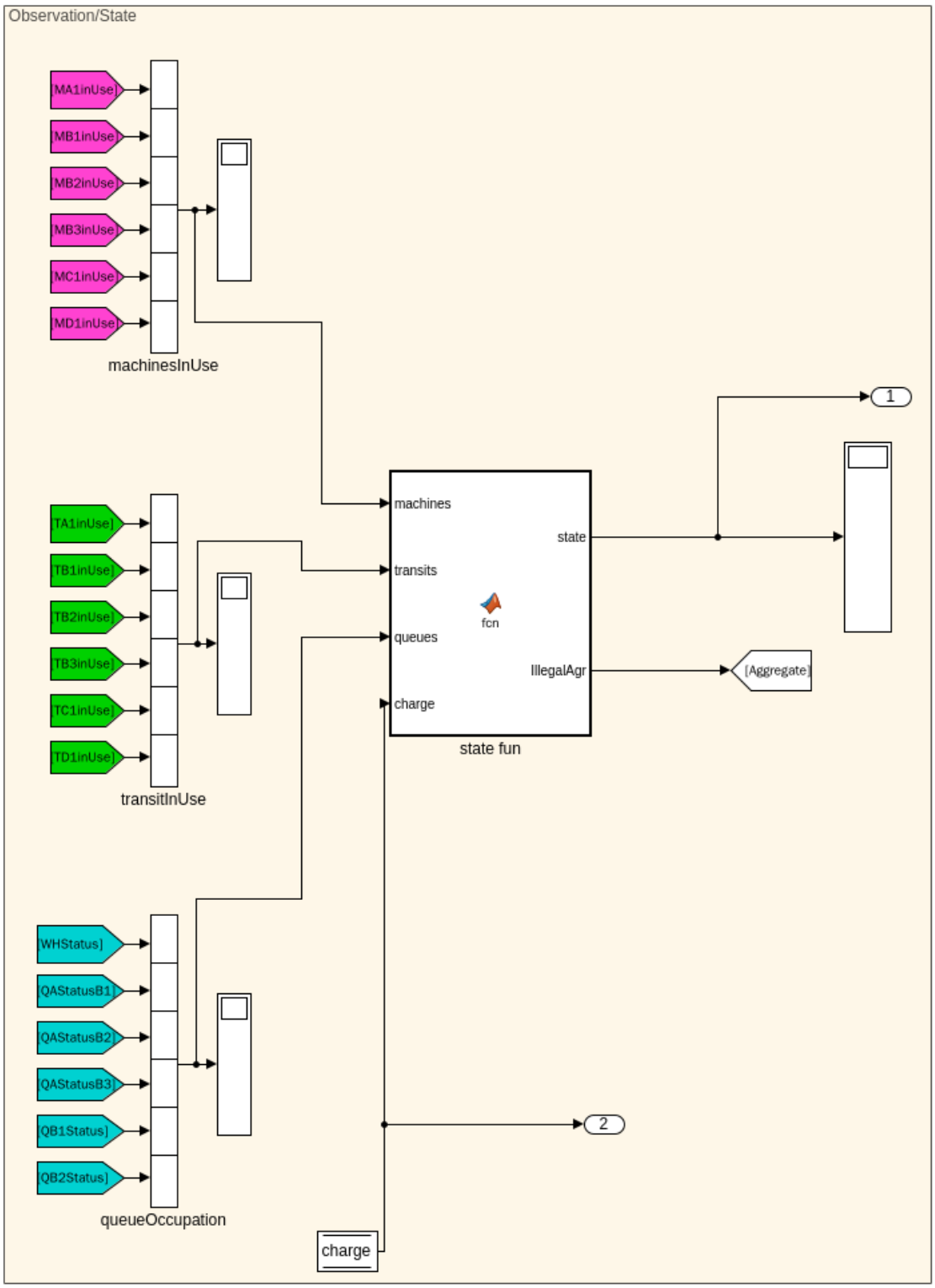

Figure 28: Observation space of the agent in SimEVents for the RL model (with charging)

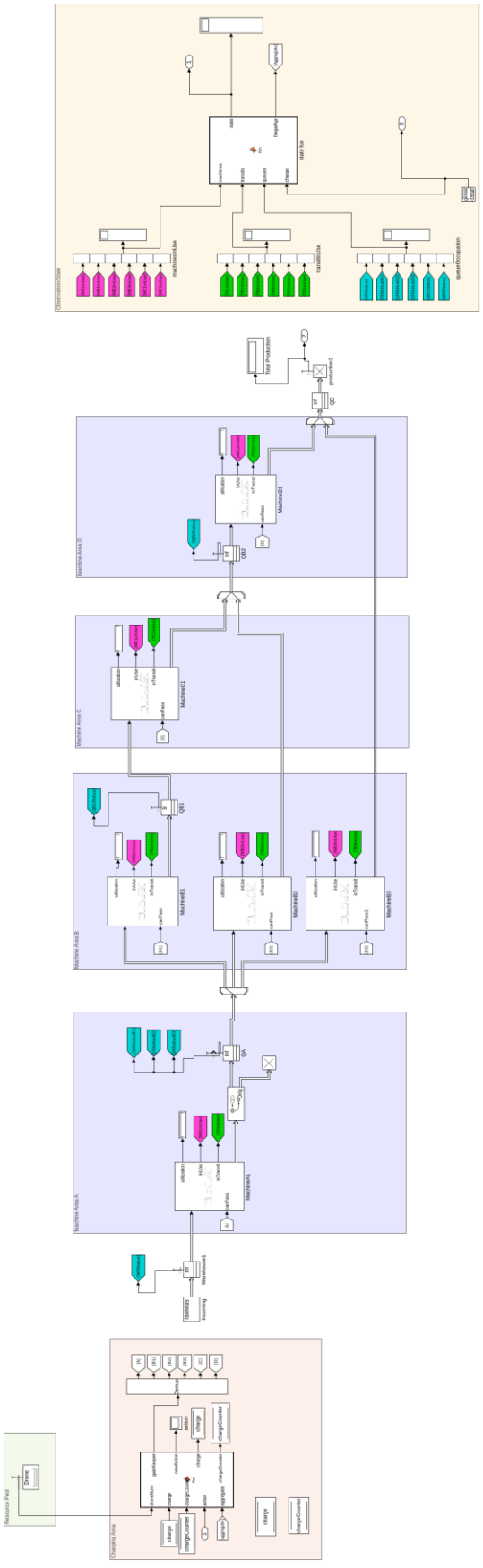

Figure 29: Structure of the FCFS model (with charging) in Matlab Simulink

4.1 Defining the reward

Reward is the short-term return for each scheduling step. When the agent chooses an action on a specific state s, the environment will give a reward r. Although a training set for learning is not provided, the system goal is expressed by reward. In the Case Study at hand, completing a product is the goal. In every episode of RL, scheduling steps are finite. The agent accumulates reward in limited steps, which show the behavior of its policy. Generally, setting of the reward determines both the way of learning and the convergence of the RL policy [40].

Referring back to the RL reward definition that *'the agent seeks to select actions that maximize its long-term cumulative reward',* the choice to represent the reward in terms of production maximization is an intuitively adequate and suitable metric to train the agent and get the desired result. For the case studies conducted on the basis of this investigation, the reward formulation, following the definition expression, can be simply represented as:

$$R(s,a) = prod_t - prod_{t-1}$$

With above formulation, the long-term reward is equated with the total production at each completed RL episode. As the agent's goal is to maximize the long-term reward, production is in this way also maximized.

4.2 Creating the agent

As agent to our problem, we use the built-in Matlab PPO agent. It uses two models, both Deep Neural Nets, one called the Actor and the other called the Critic.

4.2.1 The actor model

The Actor model performs the task of learning what action to take under a particular observed state of the environment. In our case, it takes the observation vector regarding the machine availability, transition progress, queue state and – for the charging case – the drone battery state as input and gives a particular action like the number of the workstation which should be serviced.

```
7×1 Layer array with layers:

   1   'input_1'       Feature Input     7 features
   2   'fc_1'          Fully Connected   128 fully connected layer
   3   'relu_body'     ReLU              ReLU
   4   'fc_body'       Fully Connected   128 fully connected layer
   5   'body_output'   ReLU              ReLU
   6   'fc_action'     Fully Connected   7 fully connected layer
   7   'output'        Softmax           softmax
```

Figure 30: Neural Net (NN) information of the actor model

4.2.2 The critic model

We send the action predicted by the Actor to the model environment and observe what happens in the production. If something positive happens as a result of our action, like production increase, then the environment sends back a positive response in the form of a reward. This reward is taken in by the Critic model. The job of the Critic model is to learn to evaluate if the action taken by the Actor led our environment to be in a better state or not and give its feedback to the Actor, hence its name. It outputs a real number indicating a rating (Q-value) of the action taken in the previous state. By comparing this rating obtained from the Critic, the Actor can compare its current policy with a new policy and decide how it wants to improve itself to take better actions.

```
6×1 Layer array with layers:

   1   'input_1'       Feature Input     7 features
   2   'fc_1'          Fully Connected   128 fully connected layer
   3   'relu_body'     ReLU              ReLU
   4   'fc_body'       Fully Connected   128 fully connected layer
   5   'body_output'   ReLU              ReLU
   6   'output'        Fully Connected   1 fully connected layer
```

Figure 31*:* Neural Net (NN) information of the critic model

4.3 Training and validating the agent

Once the environment and reinforcement learning agent are created, the agent training can commence using the train function. To configure the training options, MATLAB uses a built-in function. The train updates the agent as training progresses. This is possible because the agent is a handle object. Training terminates automatically when the conditions specified as stopping criteria are satisfied. When training terminates the training statistics and results are stored.

5 RESULTS

5.1 General Remarks

Two different scenarios (*Scenario 1* & *Scenario 2*), which differ from each other in terms of transit and processing times, as well as slightly in terms of model configuration will be examined. Both scenarios are described in detail below. The formulation and presentation of the results is conducted as follows:

Scenario 1

i. Both the 'FCFS Model' and the 'RL Model' are examined. Firstly, the no-charging cases are presented as proof of concept showing the increase in production using the RL approach, when compared to the rule-based FCFS case.
ii. Later, the charging cases are presented for both the 'FCFS Model' and the 'RL Model', reinforcing the power of the RL approach, especially for models of increasing complexity, where rule-based systems start getting even more convoluted.
iii. The 'RL Model' is run for 2 different value sets of the hyperparameter 'learning rate' to present the effect that hyperparameter tuning has in agent training. Specifically, the sets that will be examined are:
    - same learning rate between actor-critic ($\varepsilon_{actor} = \varepsilon_{critic} = 0.001$)
    - different learning rates between actor-critic ($\varepsilon_{actor} = 0.008$, $\varepsilon_{critic} = 0.005$)
iv. For the 'RL Model' and for each of the 2 different hyperparameter tunings, the cases of the 'unconstrained' and 'illegal' actions mentioned and discussed before are examined. The goal is to show that heuristics and hard-coded interference with the RL training can negatively affect the final production or cause slower convergence to the algorithm by manually leading the agent to take actions different from the one it explores on its own.
v. For each of the runs mentioned in the previous steps, the metrics regarding average total production, machine utilization, queue waiting time, drone battery life (if applicable) and agent training progress are used to quantitatively compare the models. For simplification purposes and best result presentation, most of the diagrams and data pertaining to these metrics are provided in the Appendix.
vi. For the cases where the RL approach is followed, a bar chart showing the robustness of the trained agent in depth of 100 simulations is presented. Specifically, after agent training, each model is simulated consecutively 100 times with the trained agent and the consistency of the production number between the trials is monitored.

The comparison of the simulation runs will always take place between the FCFS and RL models that share the same properties (no charge / with charge) and only for the same scenarios and simulation parameters.

Description

Scenario 1 contains an "artificial trap" after the departure of the product from workstation A1. Specifically, the drone has the choice to serve three different workstations, however there is no knowledge of the next steps at this particular time step. Machine B1 (**Table *2***) seemingly has the least cumulative transition and processing time, so intuitively this would probably be a good step towards maximizing the cell production, whereas workstation B3 seems to be the slowest overall, so avoiding it would lead to better results. However, the steps following show that this is a trap, as machine B1 leads to another workstation that increases the processing time necessary if this path is followed, while B3 is in fact faster.

The goal is to test whether this trap can intelligently be avoided by the RL agent that is supposed to learn on its own by exploring and exploiting different paths according to the average reward it accumulates in each case.

Workstation	Transit time (TA)	Machining time (MA)
Machine A (Saw)	1	3
Machine B1 (Lathe)	1	1
Machine B2 (Lathe/Mill(1))	1	2
Machine B3 (Lathe/Mill(2))	3	6
Machine C1 (Mill)	1	4
Machine D (Sprocket)	1	2

Table 2: Workstation transit times (TA) and processing times (MA) for simulation Scenario 1

Simulation parameter	Value
Simulation time (time units)	128
Raw material input [pcs]	15

Table 3: Simulation parameters for FCFS and RL models (no charging) in Mathworks SimEvents for simulation Scenario 1

Simulation parameter	Value
Simulation time (time units)	128
Raw material input [pcs]	10

Table 4: Simulation parameters for FCFS and RL models (with charging) in Mathworks SimEvents for simulation Scenario 1

Scenario 2

After Scenario 1 is run, the goal is to be able to extract a series of model parameters and simulation hyperparameters for testing and employment in a different manufacturing setting, which is presented in detail below. A new RL agent is trained using a combination of suitable parameters, as a proof of robustness and consistency of the RL approach in dynamic environments.

Description

Scenario 2 constitutes a modification of the system as we have seen it so far. The configuration is specifically presented in **Figure 32**, corresponding to the machine interconnection for both the FCFS and the RL model. The TA and MA transit are formed in a way that creates no 'traps' for the system. The workstations are approximately equal in terms of cumulative transit and processing time.

The goal is to test, whether the RL approach will be able to find a better solution in a system where the first-come first-serve basis feels like the most intuitive way of servicing the cell.

Workstation	Transit time (TA)	Machining time (MA)
Machine A (Saw)	1.5	2.5
Machine B1 (Lathe)	0.5	2
Machine B2 (Lathe/Mill(1))	0.5	4
Machine B3 (Lathe/Mill(2))	0.5	4
Machine C1 (Mill)	1.5	1.5
Machine D (Sprocket)	1.5	1.5

Table 5: Workstation transit times (TA) and processing times (MA) for simulation Scenario 2

Simulation parameter	Value
Simulation time (time units)	128
Raw material input [pcs]	15

Table 6: Simulation parameters for FCFS and RL models (with charging) in Mathworks SimEvents for simulation Scenario 2

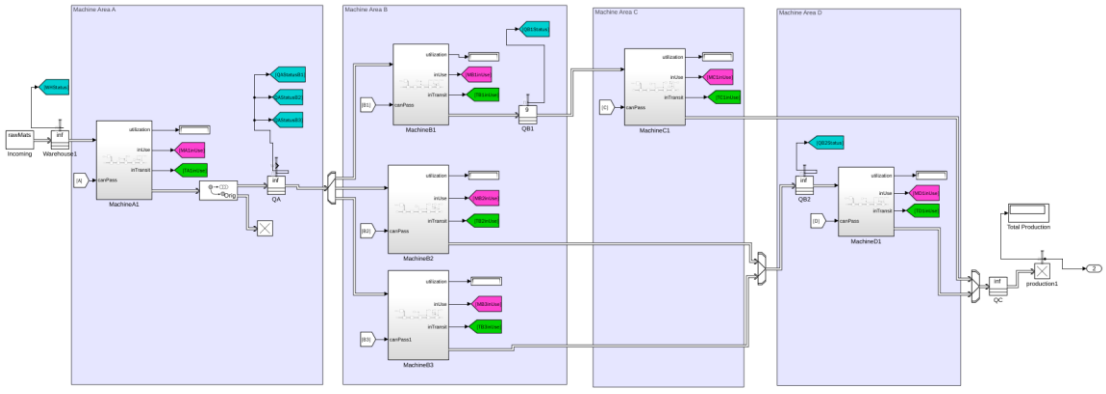

Figure 32: Model machine configuration for FCFS and RL model for Scenario 2

5.2 Scenario 1

5.2.1 FCFS model with 1 drone – No charging

Average Production Achieved (APA)	**28**

The FCFS model is able to produce approximately 62% of the theoretical maximum cell capacity for Scenario 1, which is the number of raw material pieces (15) multiplied by the number of bars in which the raw material is cut (each piece is cut into 3 bars).

From the diagrams provided in Appendix A for this case, it is evident that all workstations are treated as equal by the system (FCFS), so a distributed utilization time is observed across all workstations, while the graph profiles are coherent with each other presenting no distinct differences. Machine utilization also seems to be well-tuned for this case in terms of idle time of workstations. Specifically, all workstations seem to share the workload and the system shows no preference when it comes to selecting a production flow path. All paths here are equiprobable and the selection is made upon the condition of whether the machine is blocked or not. Furthermore, average wait time of the entrance queue seems to increase with time. The items are being processed by the workstations, but apparently the workload is higher than the cell can handle with the simple heuristics used by the FCFS model. The average waiting time remains high up to the end of the simulation.

As the system goal is production maximization, the ideal scenario would be to acquire a total production of 45 pieces if that value is indeed attainable by the system. The RL approach studied below endeavors to explore the maximum attainable production that the system can support via maximizing model reward and thus system production.

5.2.2 RL model with 1 drone – No charging

i. Same learning rate between actor-critic ($\varepsilon_{actor} = \varepsilon_{critic} = 0.001$)

a. *'Unconstrained' actions*

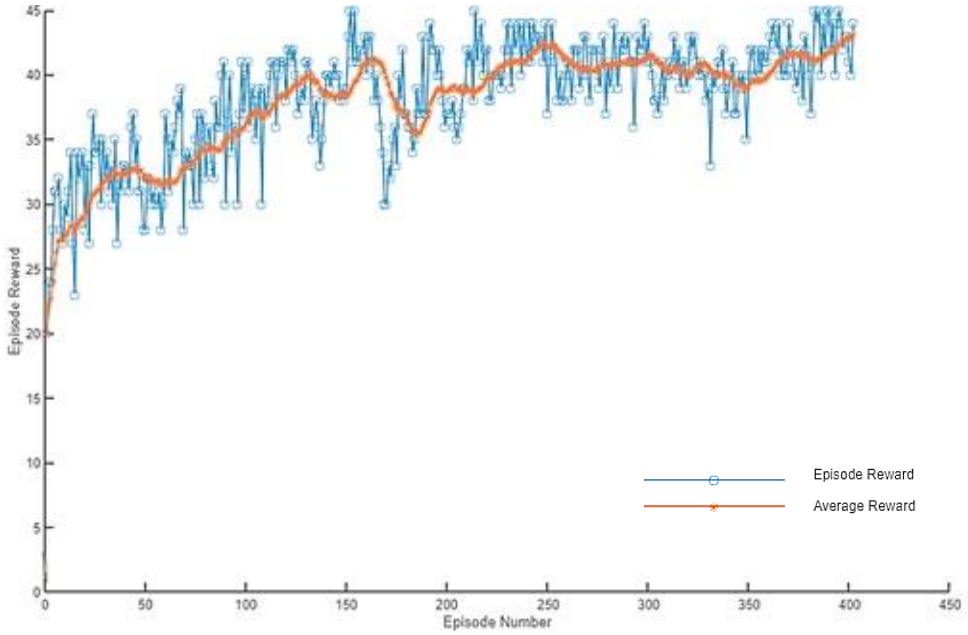

Diagram 1: Training progress of RL model (no charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

Average Production Achieved (APA)	**42.52**
Difference from FCFS APA	**+51.86%**

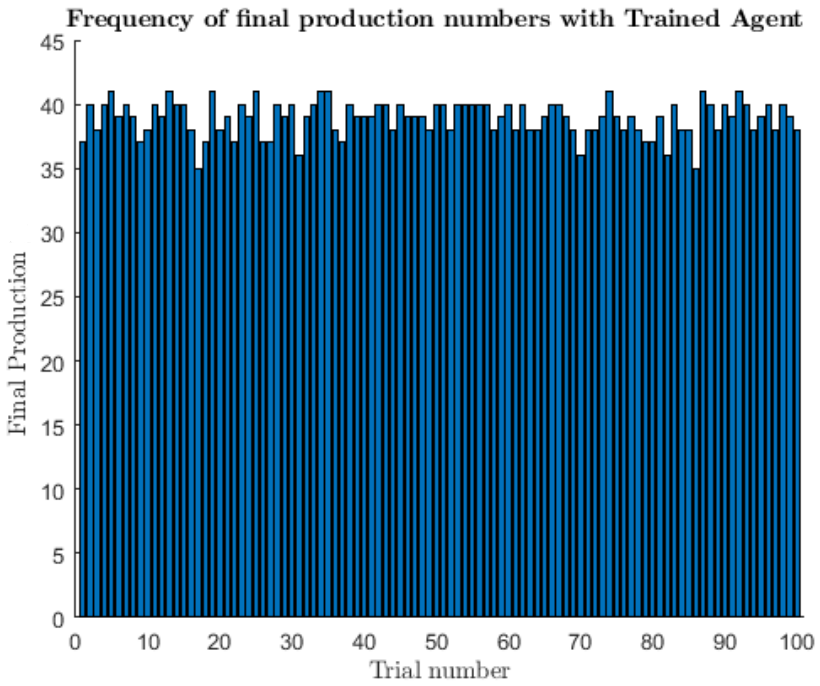

Diagram 2: Total Production of trained agent for RL model (no charging) for same actor-critic learning rate and 'unconstrained' case for 100 consecutive simulations (Scenario 1)

b. 'Illegal' actions

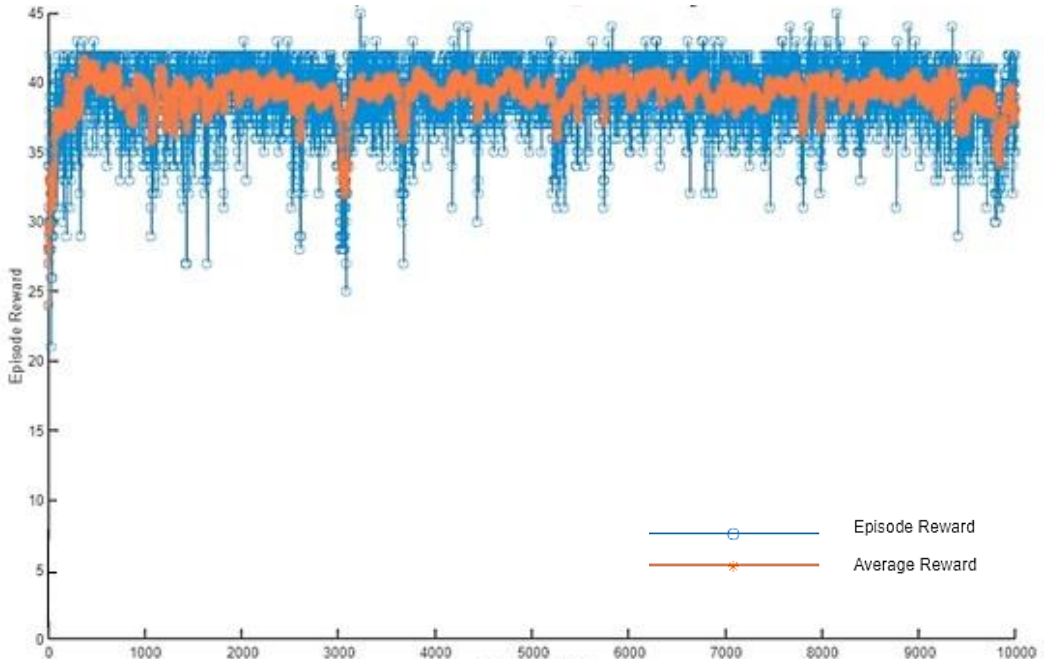

Diagram 3: Training progress of RL model (no charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

Average Production Achieved (APA)	**38.91**
Difference from FCFS APA	**+38.96%**
Difference from 'unconstrained' APA	**-8.49%**

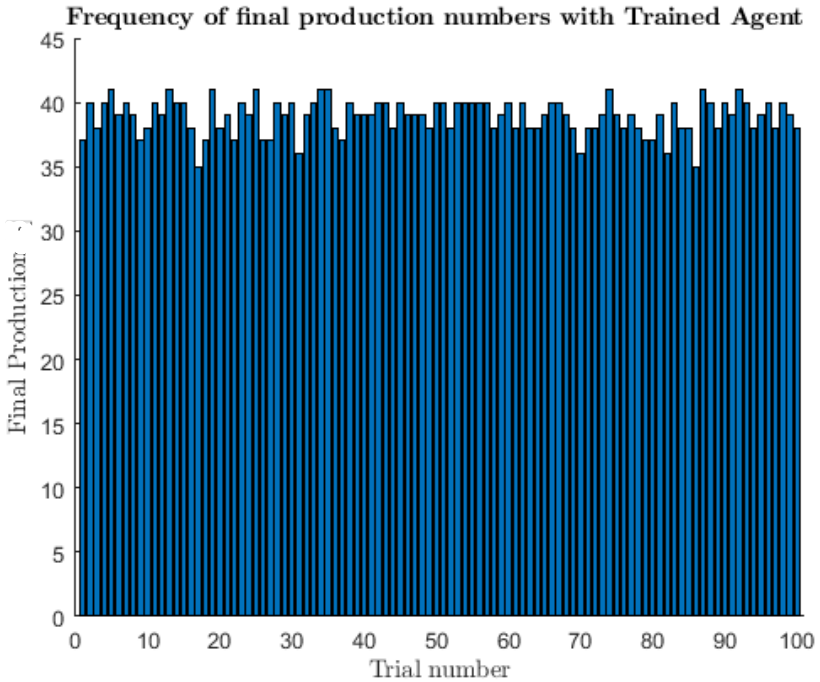

Diagram 4: Total Production of trained agent for RL model (no charging) for same actor-critic learning rate and 'illegal' case for 100 consecutive simulations (Scenario 1)

ii. different learning rates between actor-critic ($\varepsilon_{actor} = 0.008, \varepsilon_{critic} = 0.005$)

a. *'Unconstrained' actions*

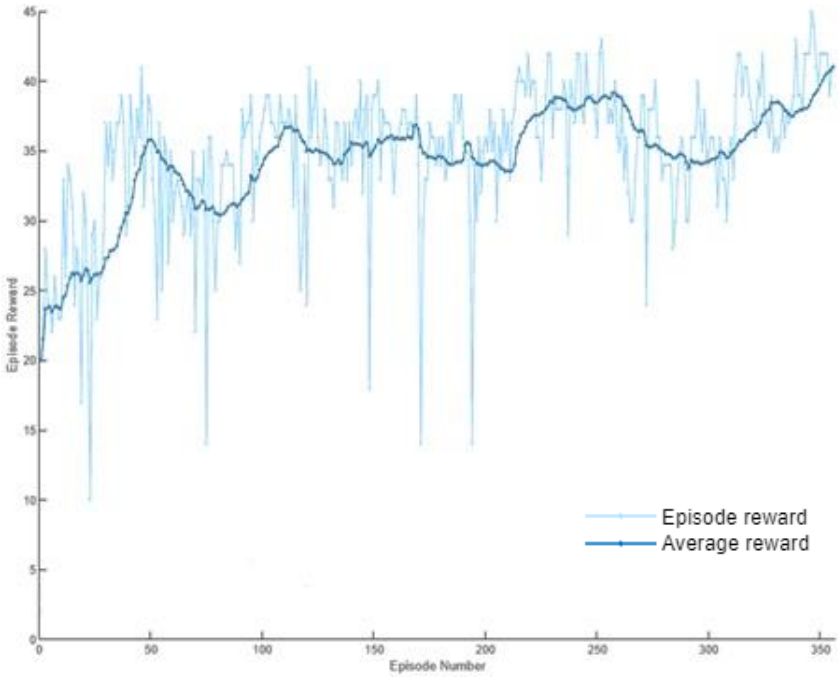

Diagram 5: Training progress of RL model(no charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

Average Production Achieved (APA)	**40.19**
Difference from FCFS APA	**+43.53%**

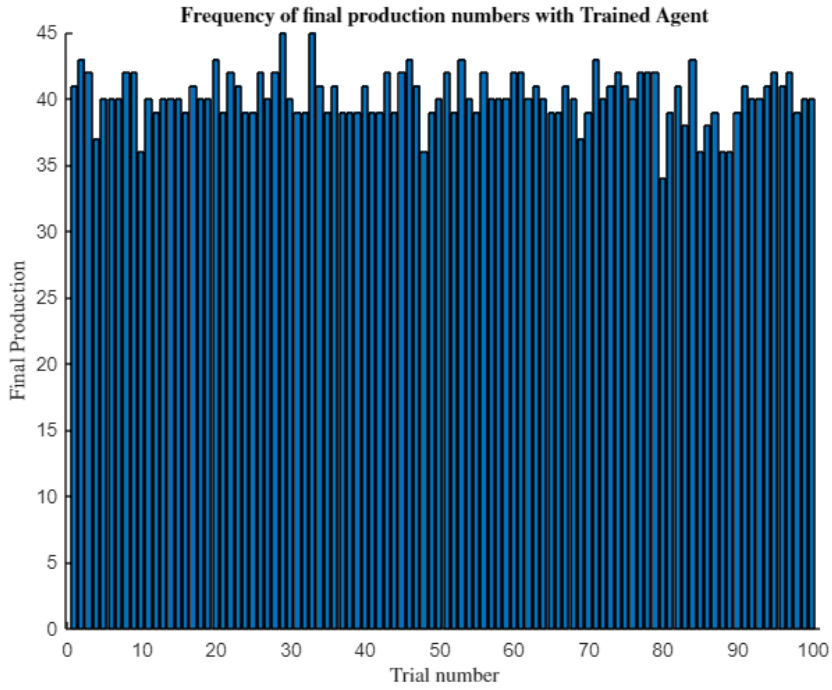

Diagram 6: Total Production of trained agent for RL model (no charging) for different actor-critic learning rate and 'unconstrained' case for 100 consecutive simulations (Scenario 1)

b. 'Illegal' actions

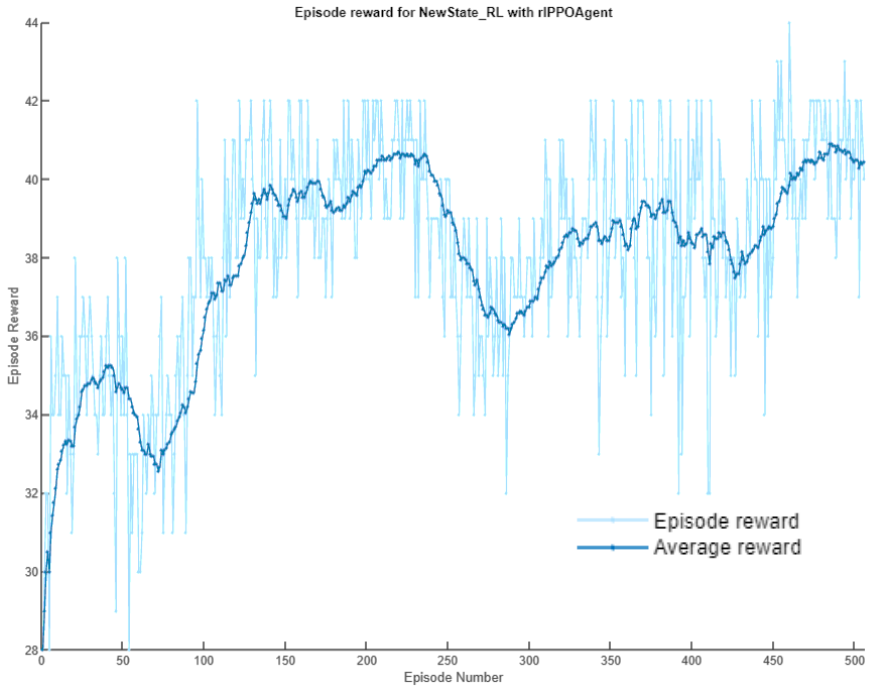

Diagram 7: Training progress of RL model (no charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

Average Production Achieved (APA)	**40.74**
Difference from FCFS APA	**+45.50%**
Difference from 'unconstrained' APA	**+1.36%**

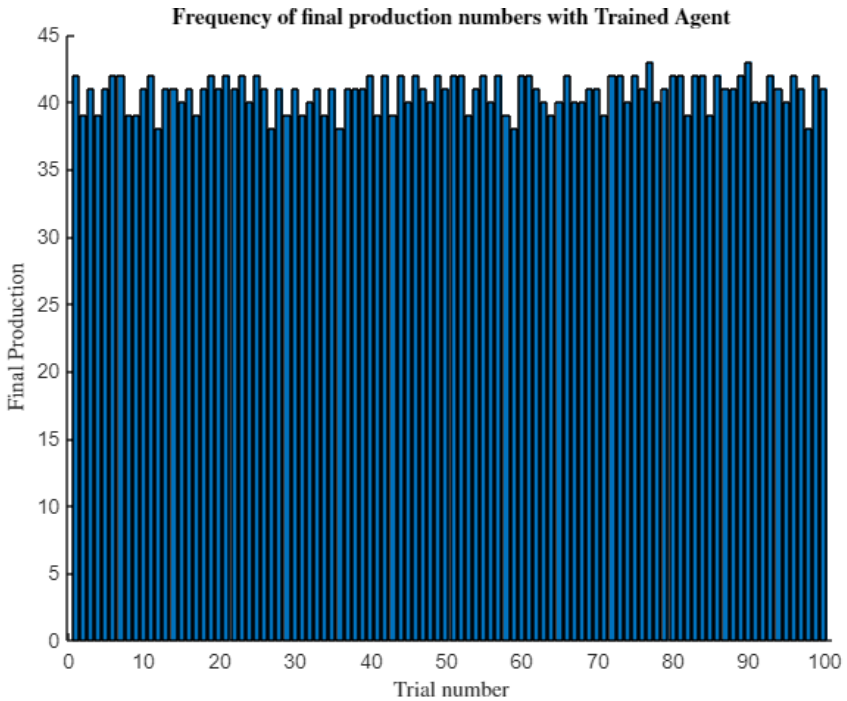

Diagram 8: Total Production of trained agent for RL model (no charging) for different actor-critic learning rate and 'illegal' case for 100 consecutive simulations (Scenario 1)

5.2.3 FCFS model with 1 drone – With charging

Average Production Achieved (APA)	**12**

The behavior of the system with the addition of charging is similar as before, however it is evident that production has now decreased as the battery needs of the drone need to be satisfied during production times.

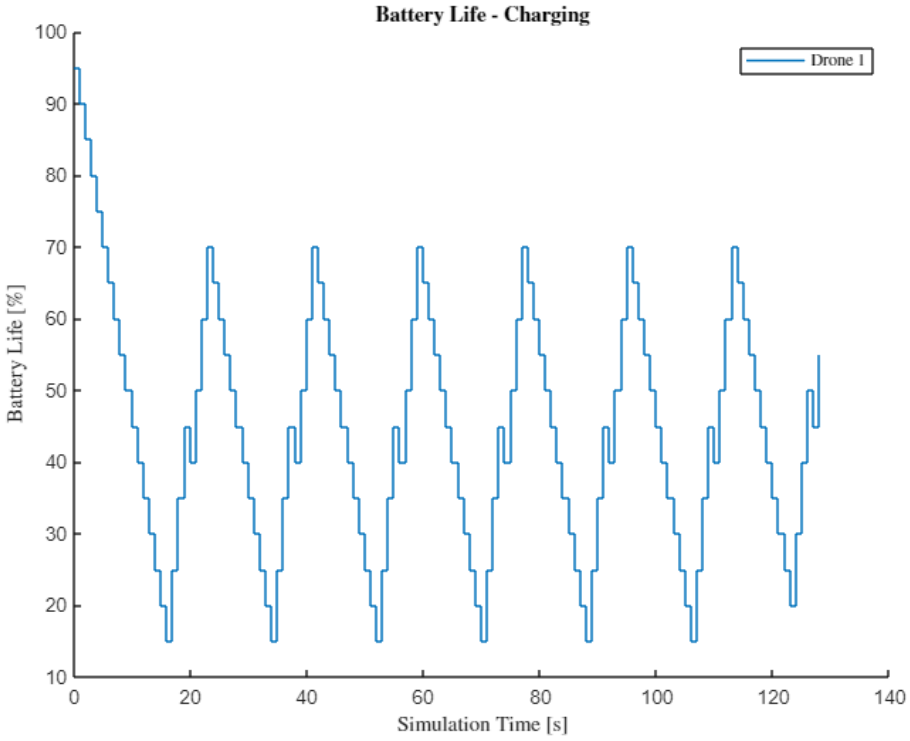

Diagram 9: Drone battery life and charging progress for the FCFS model (with charging) (Scenario 1)

It is important to note that even though the battery threshold is set to 50%, the drone reaches lower battery loads. Apparently, the battery level reaches the set threshold, while the drone finds itself mid-operation. Therefore, the battery continues to discharge until the drone is available again to return to the charging station. This heuristic seems to be safe since the battery never falls under 10%. However, for bigger and more complex systems, it becomes more difficult to find a good estimate for a battery threshold that is equally safe for the battery to not be drained and for the system to maximize its production.

As the system goal is production maximization and maintenance of the drone battery at a sufficient level to avoid drainage, the RL approach studied below endeavors to explore the maximum attainable production that the system can support, while balancing between the tasks of machine servicing and drone battery charging.

5.2.4 RL model with 1 drone – With charging

i. Same learning rate between actor-critic ($\varepsilon_{actor} = \varepsilon_{critic} = 0.001$)

a. *'Unconstrained' actions*

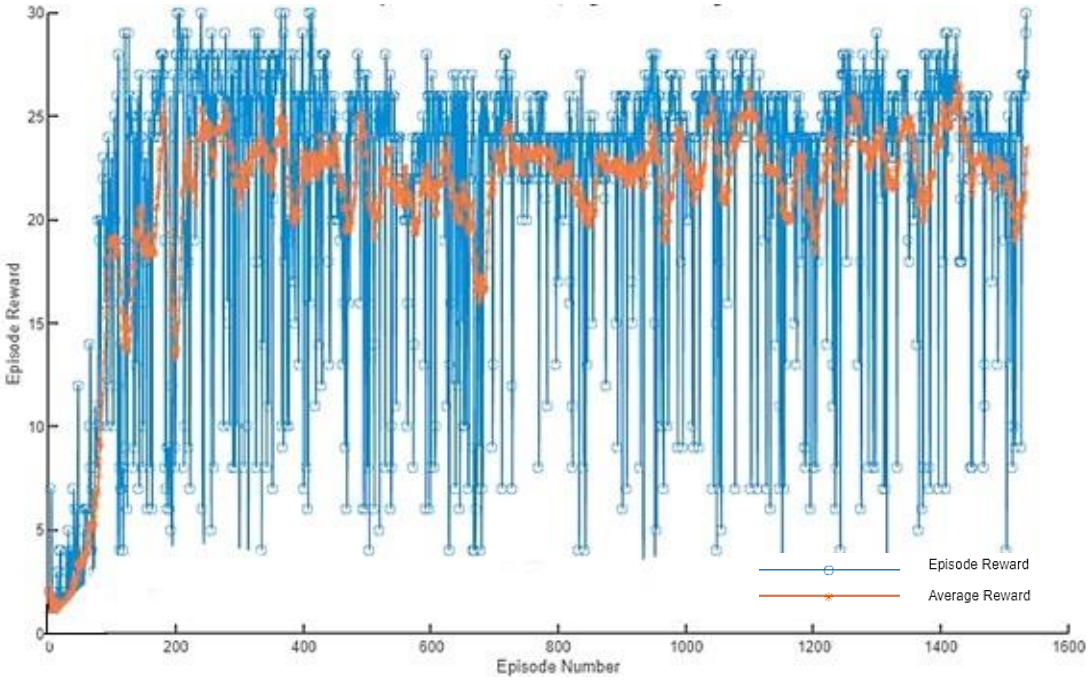

Diagram 10*:* Training progress of RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

Average Production Achieved (APA)	**28.57**
Difference from FCFS APA	**+138.08%**

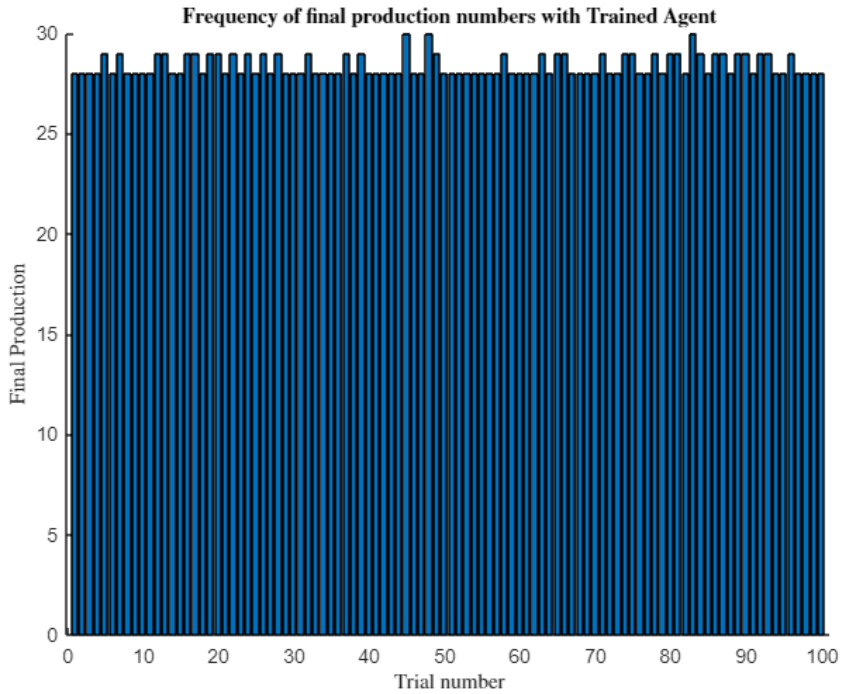

Diagram 11: Total Production of trained agent for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case for 100 consecutive simulations (Scenario 1)

b. *'Illegal' actions*

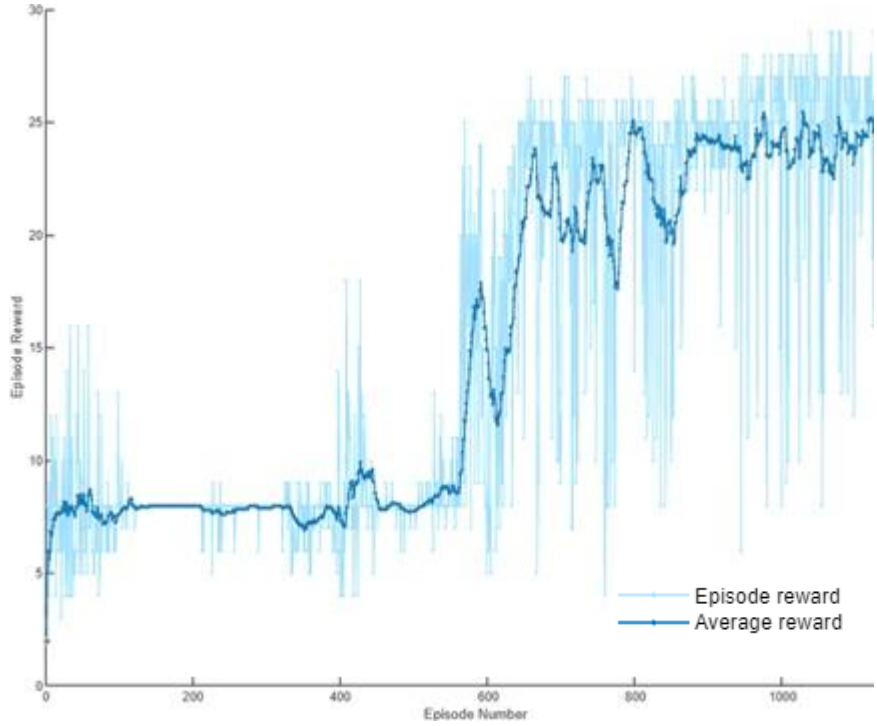

Diagram 12*:* Training progress of RL model (with charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

Average Production Achieved (APA)	**19.08**
Difference from FCFS APA	**+59.00%**
Difference from 'unconstrained' APA	**-17.58%**

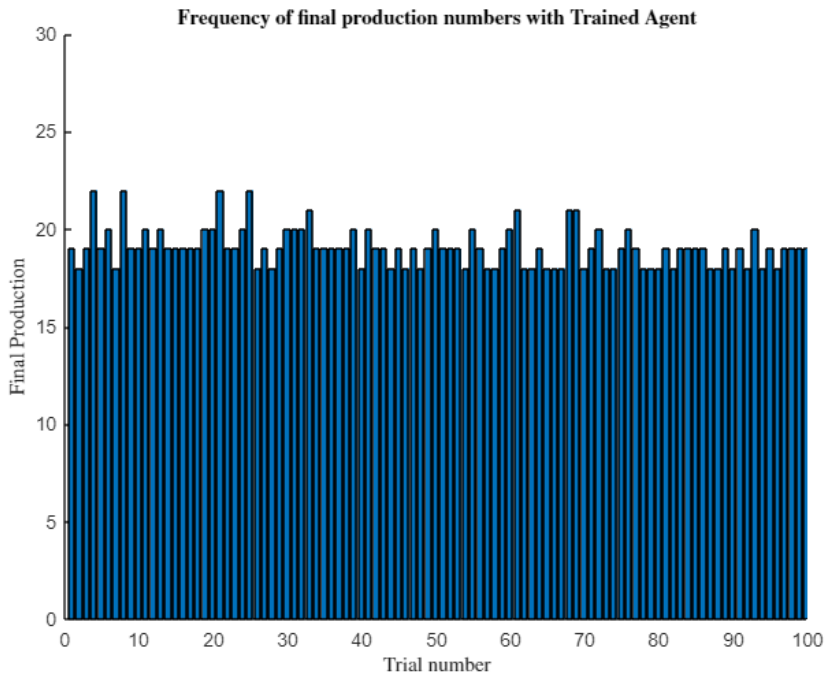

Diagram 13: Total Production of trained agent for RL model (with charging) for different actor-critic learning rate and 'illegal' case for 100 consecutive simulations (Scenario 1)

ii. Different learning rate between actor-critic ($\varepsilon_{actor} = 0.008, \varepsilon_{critic} = 0.005$)

a. 'Unconstrained' *actions*

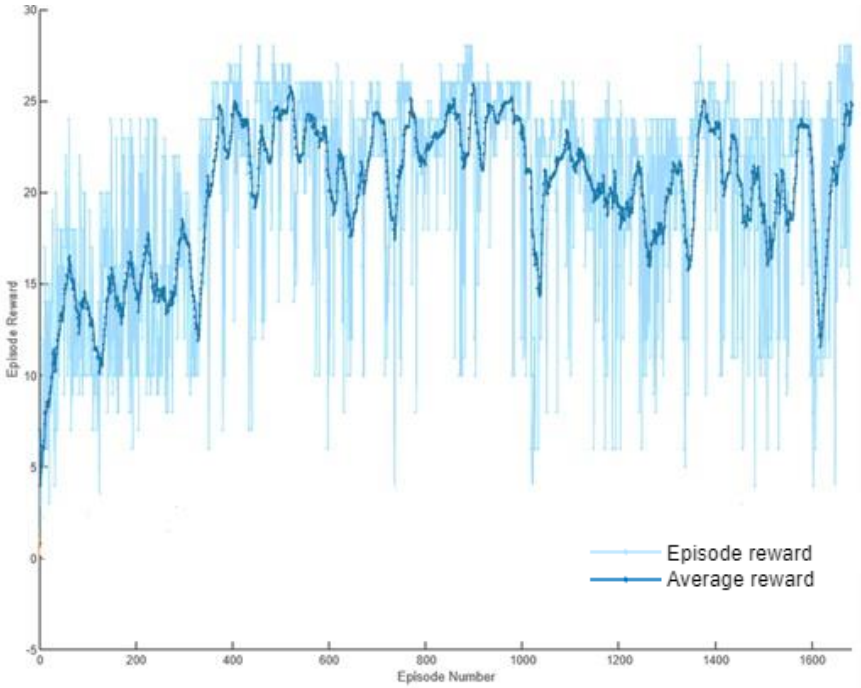

Diagram 14: Training progress of RL model (with charging) for different actor-critic learning rate and 'legal' case (Scenario 1)

Average Production Achieved (APA)	**29.42**
Difference from FCFS APA	**+145.16%**

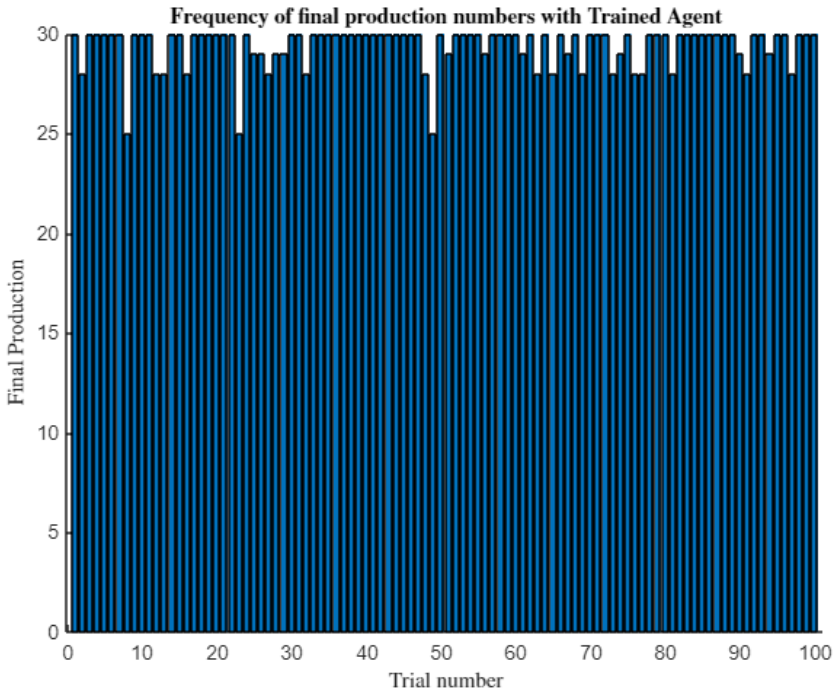

Diagram 15: Total Production of trained agent for RL model (with charging) for different actor-critic learning rate and 'unconstrained' case for 100 consecutive simulations (Scenario 1)

b. *'Illegal' actions*

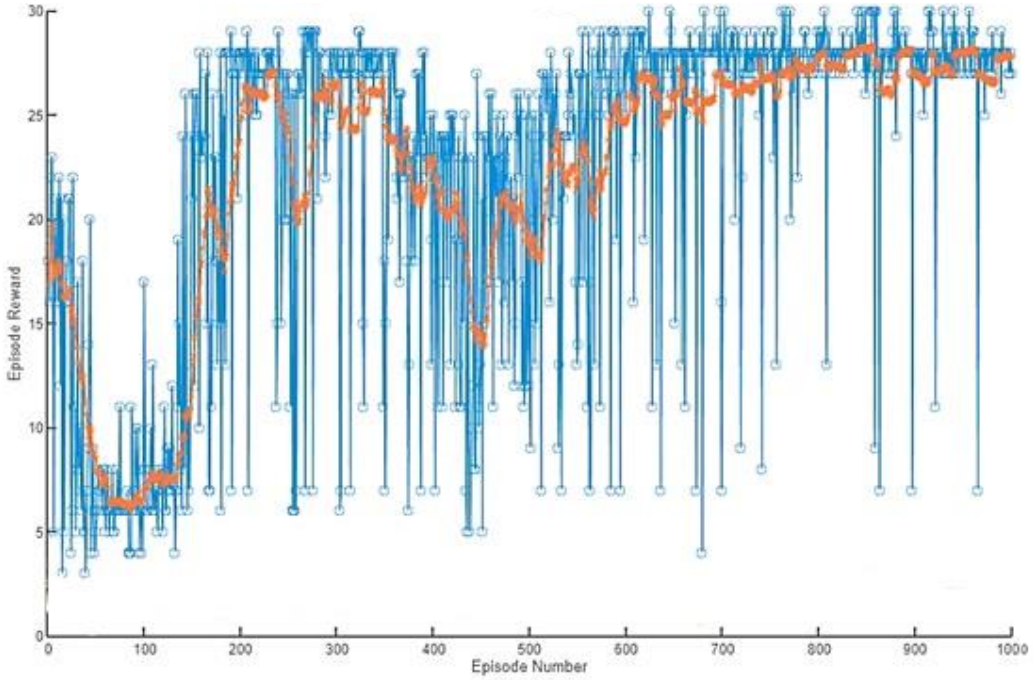

Diagram 16: Training progress of RL model (with charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

Average Production Achieved (APA)	**21.43**
Difference from FCFS APA	**+78.58%**
Difference from 'unconstrained' APA	**-27.15%**

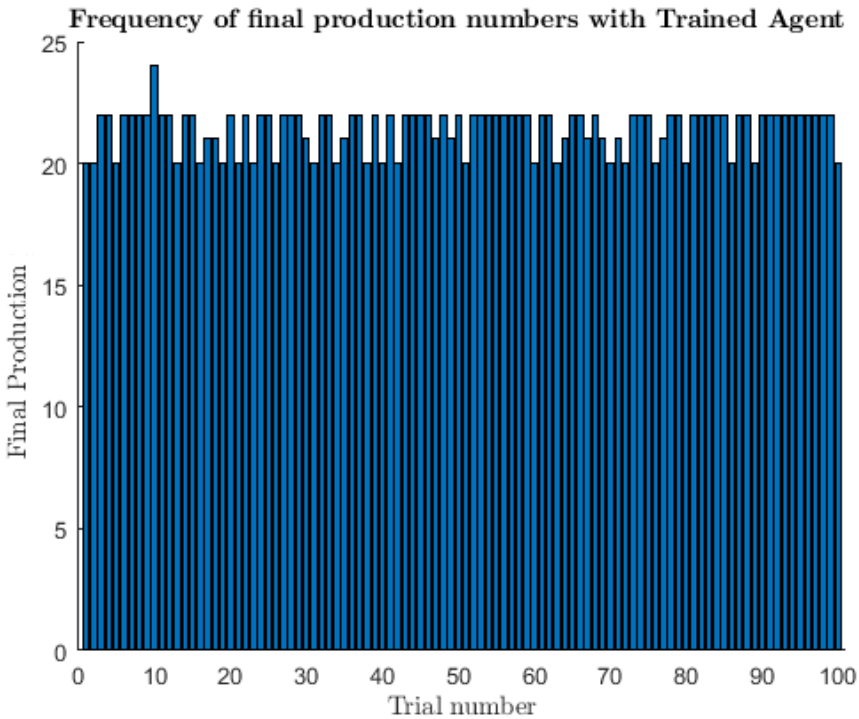

Diagram 17: Total Production of trained agent for RL model (with charging) for different actor-critic learning rate and 'unconstrained' case for 100 consecutive simulations (Scenario 1)

5.3 Scenario 2

After successfully completing the runs for Scenario 1, it is evident that lower learning rates tend to perform better in terms of agent training speed and convergence, while the 'unconstrained' case, which purely consists of the agent's action decisions, also performs better than when manually interfered with. The formulation and presentation of the results is thus conducted as follows:

i. Both the 'FCFS Model' and the 'RL Model' are examined for the cases including battery charging.
ii. The 'RL Model' is run for 1 value set of the hyperparameter 'learning rate' and specifically for:
    - the same learning rate between actor-critic ($\varepsilon_{actor} = \varepsilon_{critic} = 0.001$)
iii. For the 'RL Model' the case of the 'unconstrained' actions mentioned and discussed before is examined.
iv. For each of the runs mentioned in the previous steps, the metrics regarding average total production, machine utilization, queue waiting time, drone battery life and agent training progress are used to quantitatively compare the models. For simplification purposes and best result presentation, most of the diagrams and data pertaining to these metrics are provided in the Appendix.

For the RL approach, a bar chart showing the robustness of the trained agent in depth of 100 simulations is presented. Specifically, after agent training, the RL model is simulated consecutively 100 times with the trained agent and the consistency of the production number between the trials is monitored.

5.3.1 FCFS model with 1 drone – With charging

Average Production Achieved (APA)	**22**

The FCFS model is able to produce approximately 50% of the theoretical maximum cell capacity for Scenario 2, which is the number of raw material pieces (15) multiplied by the number of bars in which the raw material is cut (each piece is cut into 3 bars).

From the diagrams provided in Appendix A for this case, it is evident that all workstations are again treated as equal by the system (FCFS), so a distributed utilization time is observed across all workstations, while the graph profiles are coherent with each other presenting no distinct differences. Machine utilization also seems to be well-tuned for this case in terms of idle time of workstations. Specifically, all workstations seem to share the workload and the system shows no preference when it comes to selecting a production flow path. All paths here are equiprobable and the selection is made upon the condition of whether the machine is blocked or not. Furthermore, average wait time of the entrance queue seems to increase with time, an observation also made for the case of Scenario 1.

As the system goal is production maximization, the ideal scenario would be to acquire a total production of 45 pieces if that value is indeed attainable by the system. The RL approach studied below endeavors to explore the maximum attainable production that the system can support via maximizing model reward and thus system production by utilizing the parameters selected after examining Scenario 1 cases.

5.3.2 RL model with 1 drone –With charging

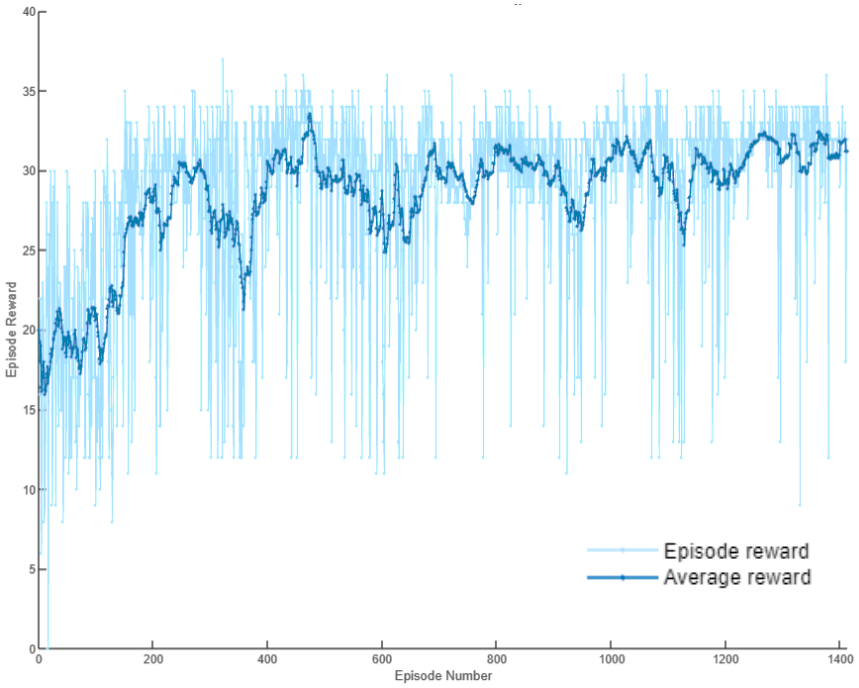

Diagram 18: Training progress of RL model with charging (Scenario 2)

Average Production Achieved (APA)	**24.71**
Difference from FCFS APA	**+12.31%**

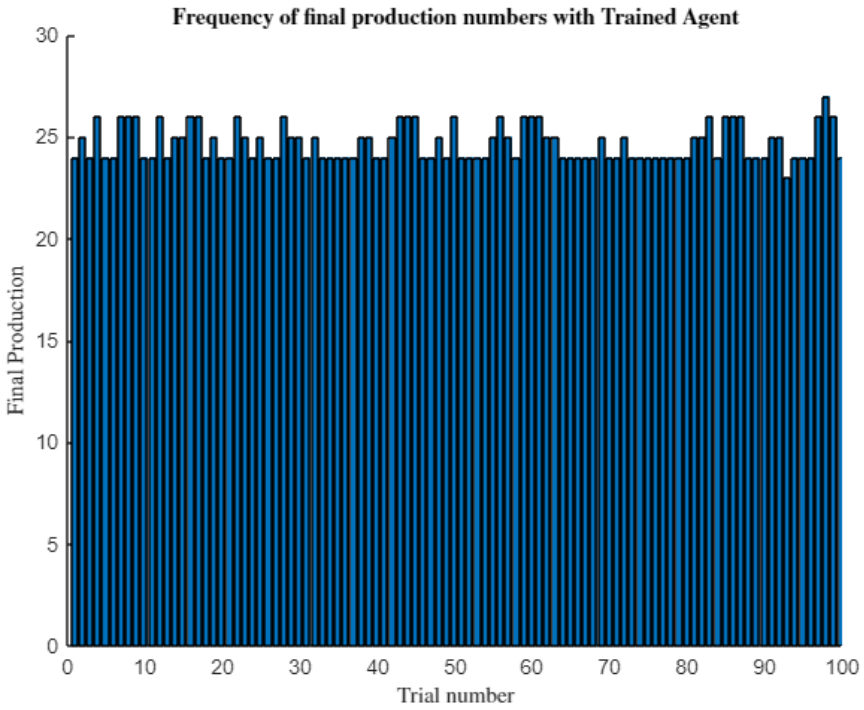

Diagram 19: Total Production of trained agent for RL model (with charging) for 100 consecutive simulations (Scenario 2)

5.4 *Concluding remarks*

The simulations conducted for both Scenario 1 and Scenario 2 consistently show that the RL outperforms the FCFS model for every case tested.

The ideal models, where the battery charging is not included, serve as base of our hypothesis that an RL approach would add more value in the production cell in terms of production maximization. The RL model reaches the theoretical maximum of the cell's capacity in a very consistent way, which presents only small divergence in a depth of 100 consecutive simulations. This proves the robustness of the approach and rejects the possibility of a randomly generated production maximization.

Getting above results for the simple, ideal case, the goal was to implement this in a generally more complex problem, where the drone battery charge would be included. Although a virtually simple addition, charging interferes with the system to a great extent, as the system must now cope with periodic interruptions that could be thought of as charging breaks. Incorporating this parameter as a heuristic to a rule-based manufacturing system constitutes a challenge for the system designers, since it is of paramount importance to strike a balance between maximum production and adequate drone charging. Specifically, a very stringent charging policy that requires the drone to return to its charging station very often would be highly uneconomical, as well as problematic for the production flow. On the other hand, a very relaxed charging condition entails the risk of battery drainage in the middle of the production process.

The models including the charging of the drone battery seem to follow the behavior observed in the ideal models. In more detail, the RL approach outperforms the FCFS model by a surprising 145% increase in terms of production. The RL models examined are capable of consistently producing an increased number of final products that approximate the cell's theoretical maximum capacity, while at the same time maintaining a healthy battery life. The RL agents successfully 'learn' to balance between the system requirements they are presented with.

As far as the RL models are concerned, it is evident that lower learning rates tend to perform better in terms of agent training speed and convergence, while the 'unconstrained' case, which purely consists of the agent's action decisions also performs better than when manually interfered with. This can be seen by comparing the production numbers between the 'unconstrained' and 'illegal' cases, with the latter tending to be either approximately the same or less than the former.

Overall, the case study conducted as part of this investigation shows that the RL approach constitutes a feasible and intelligent production scheduling and dispatching method for employment in flexible manufacturing systems. The challenge of utilizing unmanned aerial vehicles to service a manufacturing system is tackled successfully and the results are very promising for extension in more complicated systems, where heuristics become increasingly hectic.

6 CONCLUSIONS AND FUTURE WORK

6.1 Contribution

This master thesis investigates RL as a method to optimize part dispatching in a manufacturing cell serviced by drones. The results obtained by the constructed model confirm the initial assumption that RL works efficiently for this optimization problem. More specifically, results show an up to 145% increase of the production efficiency for the RL model, compared to the conventional rule-based models. Moreover, even in the case of systems where all possible production paths are approximately equivalent, we notice that reinforcement learning again outperforms the First-Come First-Served (FCFS) model, proving that it is indeed an intelligent approach. By choosing a modular type of modeling in MATLAB's SimEvents toolbox, we achieved an easy implementation of changes in the manufacturing cell environment. Taking into consideration the above, the preliminary investigation conducted in the framework of the current master thesis is considered successful, as the objective of training an agent to intelligently select actions to maximize productivity was accomplished.

Specifically covered were:

1. Literature review of previous works on RL for dispatching-scheduling in manufacturing.
2. Thorough review of current Reinforcement Learning theory and algorithms used for tackling scheduling and production issues.
3. Recreation of a model manufacturing system, including the workstations, drones, charging station, products and their topology, aiming to investigate the use of RL within a simulated manufacturing cell setting.
4. Modelling and simulation of the manufacturing system created using MATLAB SimEvents for DES.
5. Application of simple heuristics and rules to test system behavior on a 'First-Come First-Served' basis.
6. RL implementation to find the optimal part dispatching combinations in a system with drone battery charging needs via agent training.

6.2 Limitations and Challenges

This master thesis constitutes a preliminary investigation aiming to study the optimization of part dispatching within a drone-serviced manufacturing cell via the use of an RL approach. Throughout the study, both system design and modeling assumptions were made on the basis of simplification and to achieve a more RL concentrated focus. The aim of this investigation was thus not to study a complex system, but rather an RL algorithm implementation on a realistic model that could be further extended to models of increasing complexity. Given these simplifications and model assumption, some limitations to this study are introduced and which include:

- the use of one drone to service the manufacturing cell,
- the modeling of an FMS with six workstations,
- defining transition times from and to machines as an average value of all the route paths leaving from and leading to each workstation,
- production of one family of products within the FMS, meaning that one type of workorder was available for the plant.

Above limitations however do not compromise the validity of the results obtained in this investigation, as they constitute parameters of dynamic environments, which will always be different or changing depending on the manufacturing requirements and conditions.

Deploying the trained RL agent to actual production settings entails a series of challenges that should be taken into consideration a priori, including, but not limited to:

- planning of the drone trajectories in terms of altitude to avoid collisions in the case of a UAV swarm,
- online control of the drone(s) concurrently with the RL decision making algorithm, so interfacing control and learning data of the UAV,
- the inherit battery limitation of the drones, which tend to have a limited capacity in terms of operation times, leading to disruptions in production.

6.3 Future work

Future elaboration on the thesis' objective may include further expansion on the limitations previously mentioned to extend research to more complex manufacturing systems Specifically, future work could include:

- Investigation of the use of a UAV swarm for systems of increasing complexity and for FMS required to respond to multiple workorders within the same time window.
- Study of the optimal number of drones to use in a manufacturing cell with respect to the system complexity. This is rather challenging as a small number of UAVs would lead to an underserviced cell, while an abundance of drones would result in little to no utilization of the drones
- Investigation of drone battery life in an effort to maximize the battery usage, while minimizing the charging frequency.
- Simulation of a more complex model producing multiple families of products in a manufacturing cell and deployment of the RL trained agent in real-life manufacturing settings.

7 BIBLIOGRAPHY

[1] P. Kosky, R. Balmer, W. Keat, and G. Wise, “Manufacturing Engineering,” in *Exploring Engineering*, 2013, pp. 205–235.

[2] V. Manthou and M. Vlachopoulou, “Agile Manufacturing Strategic Options,” in *Agile Manufacturing: The 21st Century Competitive Strategy*, Elsevier, 2001, pp. 685–702.

[3] O. Maghazei and T. Netland, “Drones in manufacturing: exploring opportunities for research and practice,” doi: 10.1108/JMTM-03-2019-0099.

[4] J. Yang, X. You, G. Wu, M. M. Hassan, A. Almogren, and J. Guna, “Application of reinforcement learning in UAV cluster task scheduling,” *Futur. Gener. Comput. Syst.*, vol. 95, pp. 140–148, Jun. 2019, doi: 10.1016/J.FUTURE.2018.11.014.

[5] M. Deja, M. S. Siemitkowski, G. C. Vosniakos, and G. Maltezos, “Opportunities and challenges for exploiting drones in agile manufacturing systems,” *Procedia Manuf.*, vol. 51, pp. 527–534, 2020, doi: 10.1016/J.PROMFG.2020.10.074.

[6] M. Hermann, T. Pentek, and B. Otto, “Design principles for industrie 4.0 scenarios,” *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, vol. 2016-March, pp. 3928–3937, Mar. 2016, doi: 10.1109/HICSS.2016.488.

[7] S. Wang, J. Wan, D. Li, and C. Zhang, “Implementing Smart Factory of Industrie 4.0: An Outlook,” *Int. J. Distrib. Sens. Networks*, vol. 2016, 2016, doi: 10.1155/2016/3159805.

[8] J. Lee, B. Bagheri, and H. A. Kao, “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems,” *Manuf. Lett.*, vol. 3, pp. 18–23, Jan. 2015, doi: 10.1016/J.MFGLET.2014.12.001.

[9] J. Lee, E. Lapira, B. Bagheri, and H. an Kao, “Recent advances and trends in predictive manufacturing systems in big data environment,” *Manuf. Lett.*, vol. 1, no. 1, pp. 38–41, 2013, doi: 10.1016/J.MFGLET.2013.09.005.

[10] J. Lee, H. A. Kao, and S. Yang, “Service Innovation and Smart Analytics for Industry 4.0 and Big Data Environment,” *Procedia CIRP*, vol. 16, pp. 3–8, Jan. 2014, doi: 10.1016/J.PROCIR.2014.02.001.

[11] W. Bouazza, Y. Sallez, and B. Beldjilali, “A distributed approach solving partially flexible job-shop scheduling problem with a Q-learning effect,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15890–15895, Jul. 2017, doi: 10.1016/J.IFACOL.2017.08.2354.

[12] A. Kuhnle, L. Schäfer, N. Stricker, and G. Lanza, “Design, Implementation and Evaluation of Reinforcement Learning for an Adaptive Order Dispatching in Job Shop Manufacturing Systems,” *Procedia CIRP*, vol. 81, pp. 234–239, Jan. 2019, doi: 10.1016/J.PROCIR.2019.03.041.

[13] S. Qu, J. Wang, S. Govil, and J. O. Leckie, “Optimized Adaptive Scheduling of a

Manufacturing Process System with Multi-skill Workforce and Multiple Machine Types: An Ontology-based, Multi-agent Reinforcement Learning Approach," *Procedia CIRP*, vol. 57, pp. 55–60, Jan. 2016, doi: 10.1016/J.PROCIR.2016.11.011.

[14] C. Kardos, V. Gallina, W. Sihn, and C. Laflamme, "Dynamic scheduling in a job-shop production system with reinforcement learning Digitalization of Production Processes View project MMassist II-Assistenzsysteme in der Produktion im Kontext Mensch-Maschine Kooperation View project Dynamic scheduling in a job-shop production system with reinforcement learning," *Procedia CIRP*, vol. 97, pp. 104–109, 2020, doi: 10.1016/j.procir.2020.05.210.

[15] Y. Zhu *et al.*, "Reinforcement and Imitation Learning for Diverse Visuomotor Skills," Feb. 2018, doi: 10.48550/arxiv.1802.09564.

[16] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-Supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 5129–5136, Sep. 2018, doi: 10.1109/ICRA.2018.8460655.

[17] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 6252–6259, Sep. 2018, doi: 10.1109/ICRA.2018.8461113.

[18] A. T. Azar *et al.*, "Drone Deep Reinforcement Learning: A Review," *Electron. 2021, Vol. 10, Page 999*, vol. 10, no. 9, p. 999, Apr. 2021, doi: 10.3390/ELECTRONICS10090999.

[19] A. Kusiak, "Editorial: Intelligent manufacturing: bridging two centuries," *J. Intell. Manuf. 2018 301*, vol. 30, no. 1, pp. 1–2, Dec. 2018, doi: 10.1007/S10845-018-1455-2.

[20] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *http://dx.doi.org/10.1080/00207548208947745*, vol. 20, no. 1, pp. 27–45, 2007, doi: 10.1080/00207548208947745.

[21] C. Wendt, U. İlişkilerde, S. İnşacılık, and A. Kıran, "Pattern recognition and machine learning phat le Journal of Electronic Imaging."

[22] R. S. Sutton and A. G. Barto, "Reinforcement Learning, Second Edition: An Introduction - Complete Draft," *MIT Press*, pp. 1–3, 2018, Accessed: Mar. 26, 2022. [Online]. Available: https://mitpress.mit.edu/books/reinforcement-learning-second-edition.

[23] S. E. Barbosa and M. D. Petty, "Exploiting spatio-temporal patterns using partial-state reinforcement learning in a synthetically augmented environment," *Prog. Artif. Intell. 2014 32*, vol. 3, no. 2, pp. 55–71, Aug. 2014, doi: 10.1007/S13748-014-0057-2.

[24] J. A. Bland, M. D. Petty, T. S. Whitaker, K. P. Maxwell, and W. A. Cantrell, "Machine Learning Cyberattack and Defense Strategies," *Comput. Secur.*, vol. 92, p. 101738, May 2020, doi: 10.1016/J.COSE.2020.101738.

[25] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 4906–4913, 2012, doi: 10.1109/IROS.2012.6386025.

[26] R. Sutton and A. Barto, "Reinforcement learning: an introduction." MIT Press, 2018.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, “Proximal Policy Optimization Algorithms,” Jul. 2017, doi: 10.48550/arxiv.1707.06347.

[28] T. P. Lillicrap *et al.*, “CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING.” [Online]. Available: https://goo.gl/J4PIAz.

[29] S. Dankwa and W. Zheng, “Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent,” *ACM Int. Conf. Proceeding Ser.*, Aug. 2019, doi: 10.1145/3387168.3387199.

[30] D. Silver, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” 2014.

[31] C. C.-Y. Hsu, C. Mendler-Dünner, and M. Hardt, “Revisiting Design Choices in Proximal Policy Optimization,” Sep. 2020, doi: 10.48550/arxiv.2009.10897.

[32] “Proximal Policy Optimization — Spinning Up documentation.” https://spinningup.openai.com/en/latest/algorithms/ppo.html (accessed Mar. 26, 2022).

[33] M. Allen *et al.*, “HEALTH SERVICES AND DELIVERY RESEARCH Right cot, right place, right time: improving the design and organisation of neonatal care networks-a computer simulation study,” vol. 3, 2015, doi: 10.3310/hsdr03200.

[34] N. Matloff, “Introduction to Discrete-Event Simulation and the SimPy Language,” 2008.

[35] C. G. Cassandras and S. Lafortune, “Introduction to Discrete Event Systems,” *Introd. to Discret. Event Syst.*, 2021, doi: 10.1007/978-3-030-72274-6.

[36] M. A. Gray, “Discrete event simulation: A review of simevents,” *Comput. Sci. Eng.*, vol. 9, no. 6, pp. 62–66, Nov. 2007, doi: 10.1109/MCSE.2007.112.

[37] C. G. Cassandras and S. Lafortune, “Introduction to Discrete Event Systems Second Edition A BC,” 2008.

[38] A. Bender, A. H. Pincombe, and G. D. Sherman, “Effects of decay uncertainty in the prediction of life-cycle costing for large scale military capability projects,” *18 th World IMACS / MODSIM Congr.*, pp. 13–17, 2009, Accessed: Mar. 13, 2022. [Online]. Available: http://mssanz.org.au/modsim09.

[39] R. Bateman, R. Bowden, T. Gogg, C. Harrell, and J. Mott, *System improvement using simulation*, 5th ed. Orem, UT: Promodel Corporation, 1997.

[40] D. Shi, W. Fan, Y. Xiao, T. Lin, and C. Xing, “Intelligent scheduling of discrete automated production line via deep reinforcement learning,” *Int. J. Prod. Res.*, vol. 58, no. 11, pp. 3362–3380, Jun. 2020, doi: 10.1080/00207543.2020.1717008.

APPENDIX 𝔸

Simulation Parameters & Metrics

Scenario 1

A. FCFS model with 1 drone – No charging

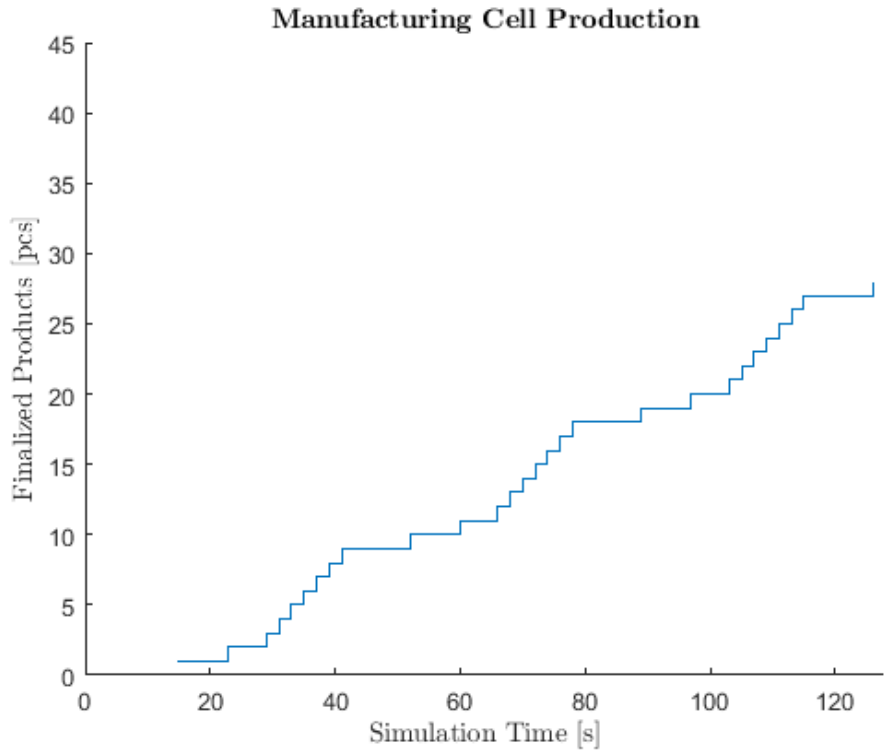

Diagram 20: Progress of manufacturing cell production for FCFS model (no charging) (Scenario 1)

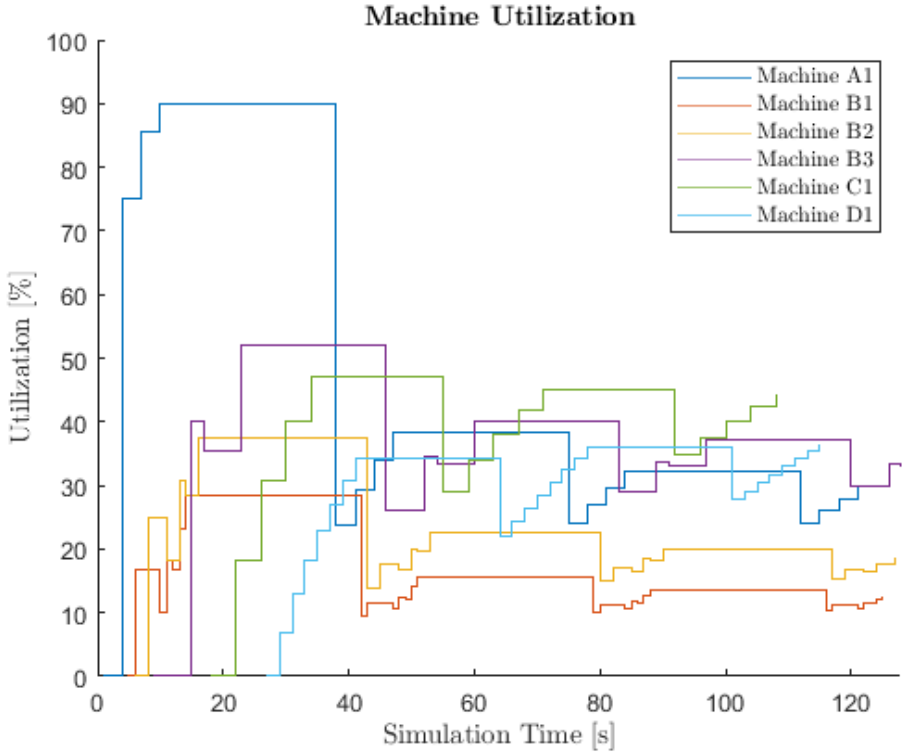

Diagram 21: Machine utilization for FCFS model (no charging) (Scenario 1)

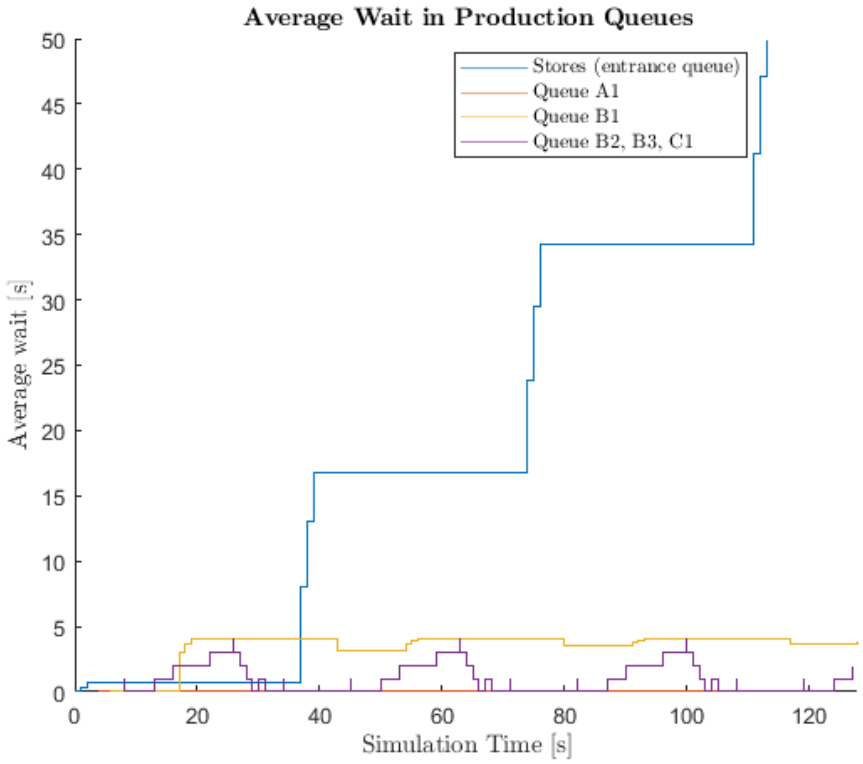

Diagram 22: Queue average waiting time for FCFS model (no charging) (Scenario 1)

B. RL model with 1 drone – No charging

Hyperparameter	Value
Entropy Loss Weight	0.01
Discount Factor	0.95
Number of Epochs	3
Experience Horizon	128
Mini Batch Size	16 (x 8)

Table 7: RL hyperparameters for agent training for RL model (no charging)

i. Same learning rate between actor-critic ($\varepsilon_{actor} = \varepsilon_{critic} = 0.001$)

(a) 'Unconstrained' actions

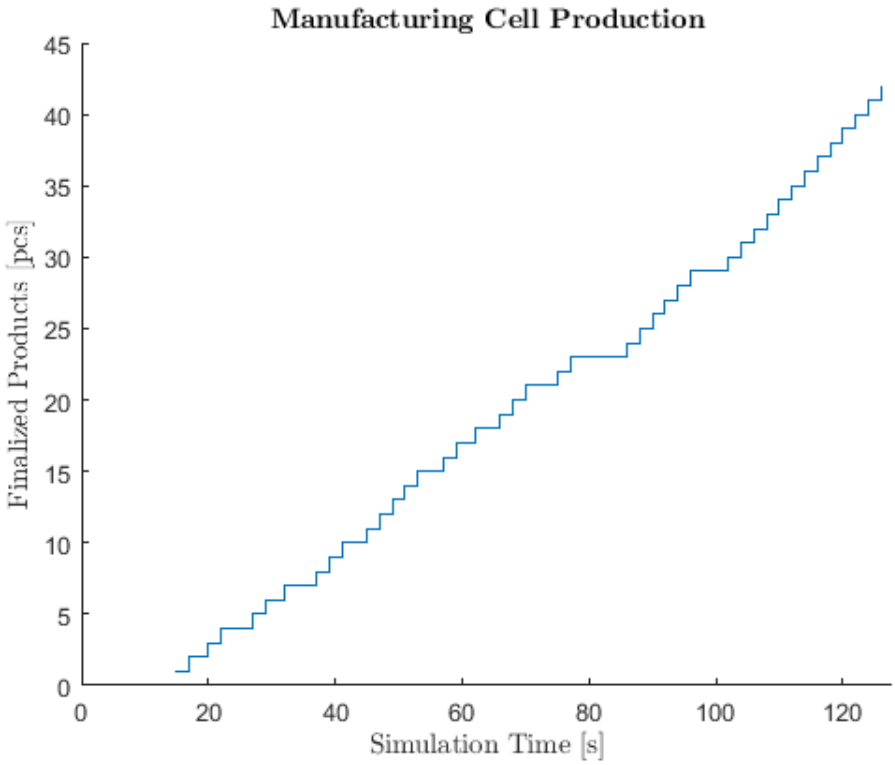

Diagram 23: Progress of manufacturing cell production for RL model (no charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

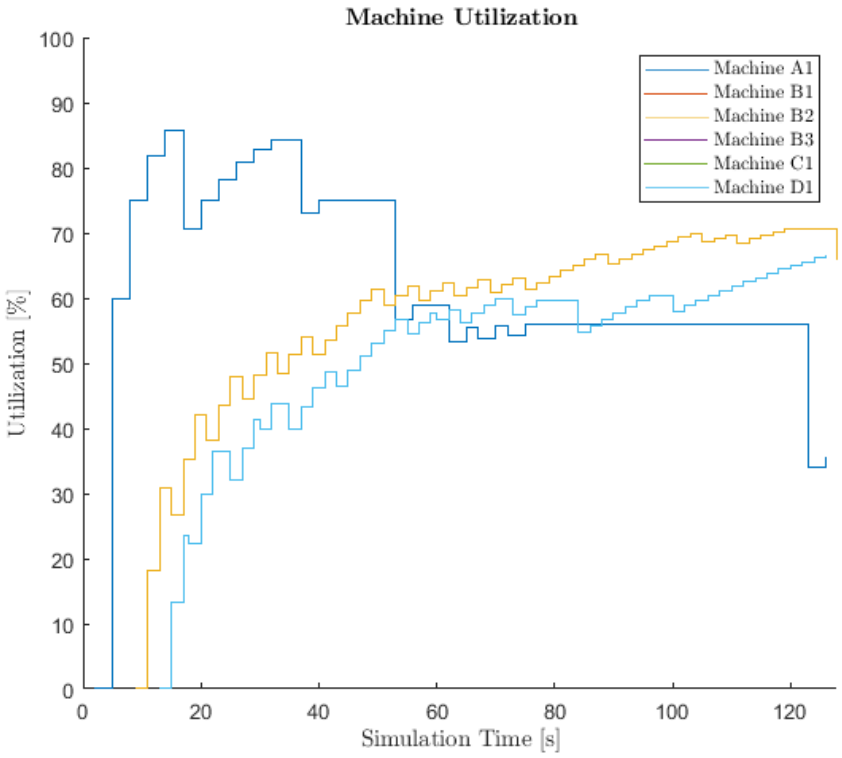

Diagram 24: Machine utilization for RL model (no charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

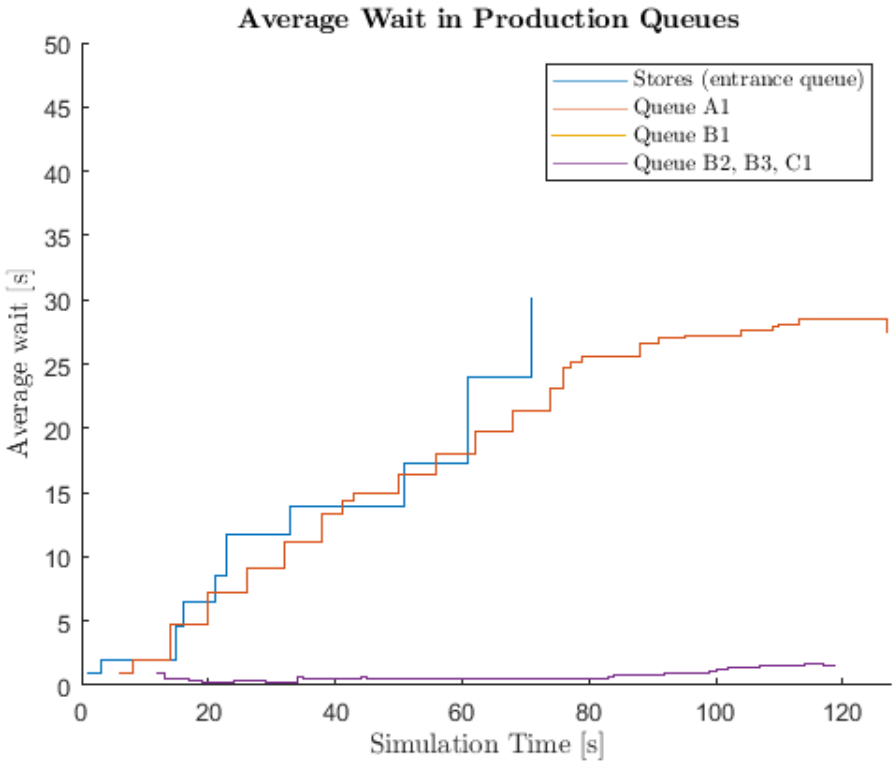

Diagram 25: Queue average waiting time for RL model (no charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

(b) 'Illegal' actions

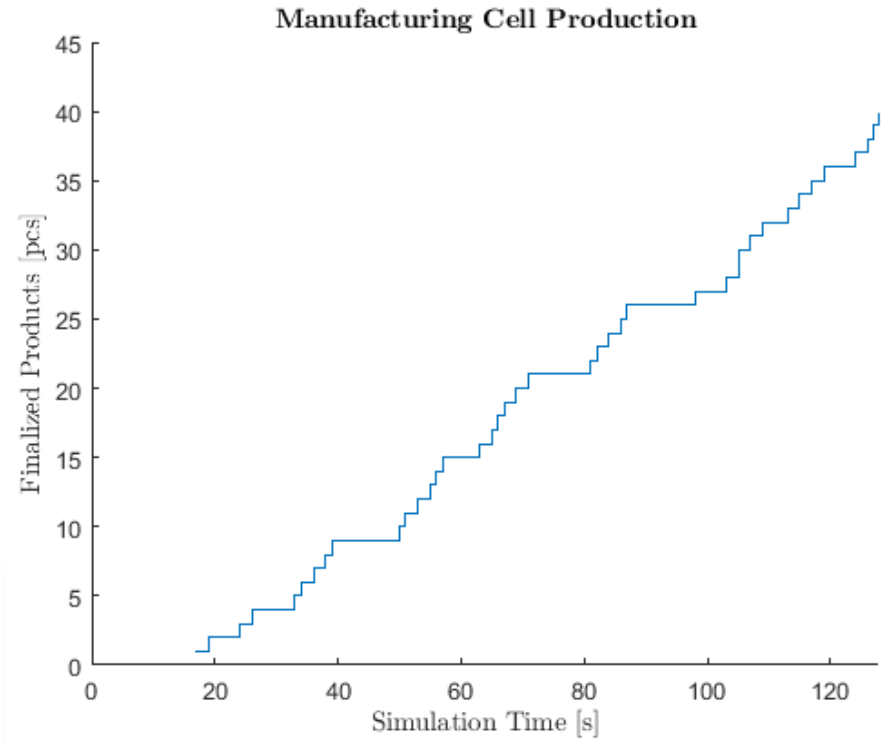

Diagram 26: Progress of manufacturing cell production for RL model (no charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

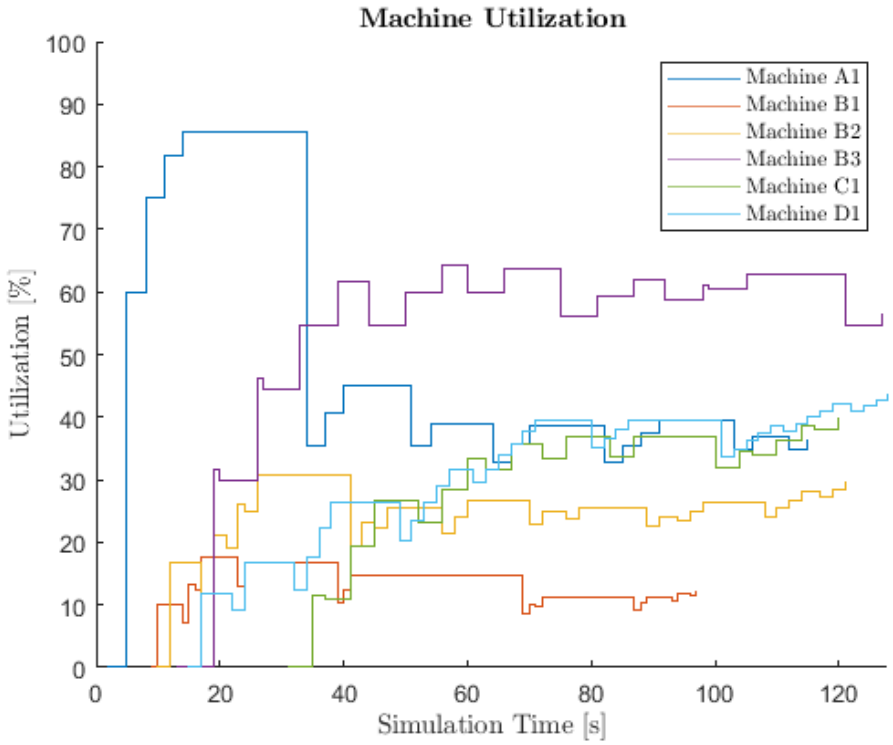

Diagram 27: Machine utilization for RL model (no charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

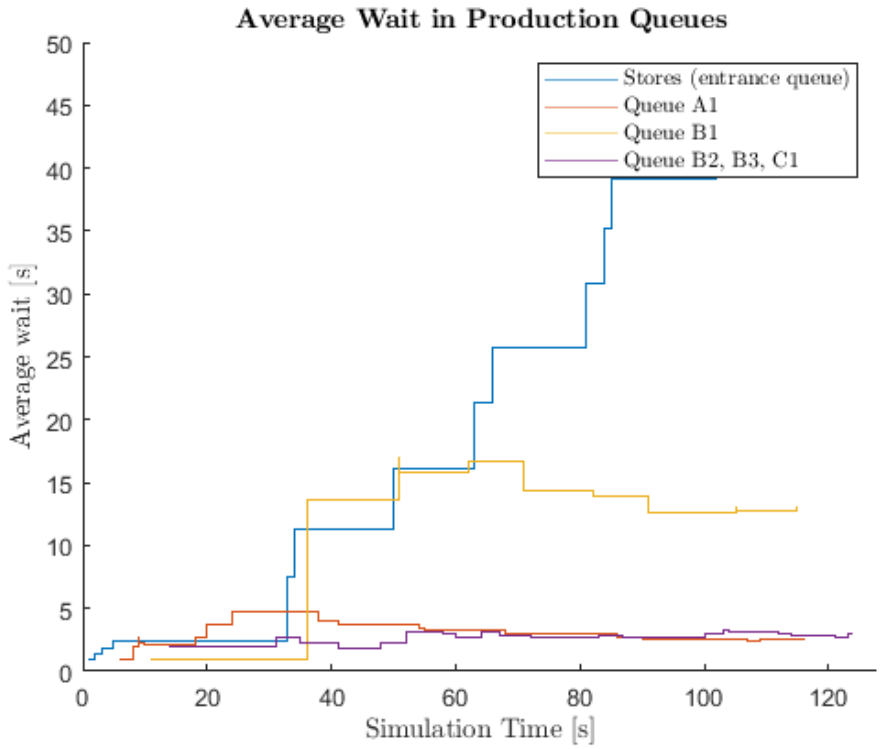

Diagram 28: Queue average waiting time for RL model (no charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

ii. Different learning rate between actor-critic ($\varepsilon_{actor} = 0.008$, $\varepsilon_{critic} = 0.005$)

(a) 'Unconstrained' actions

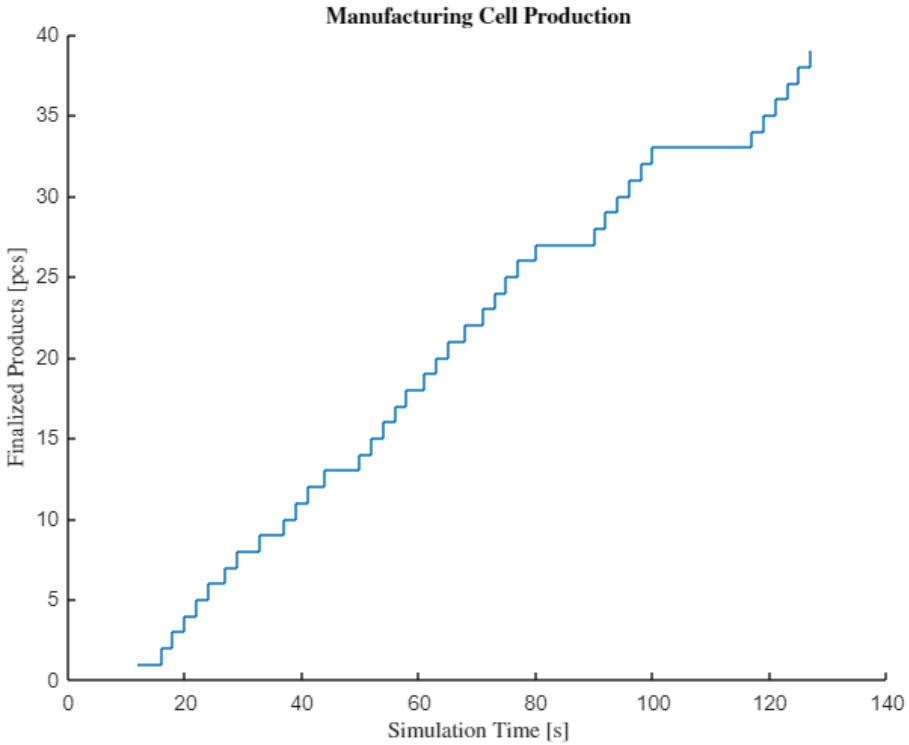

Diagram 29: Progress of manufacturing cell production for RL model (no charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

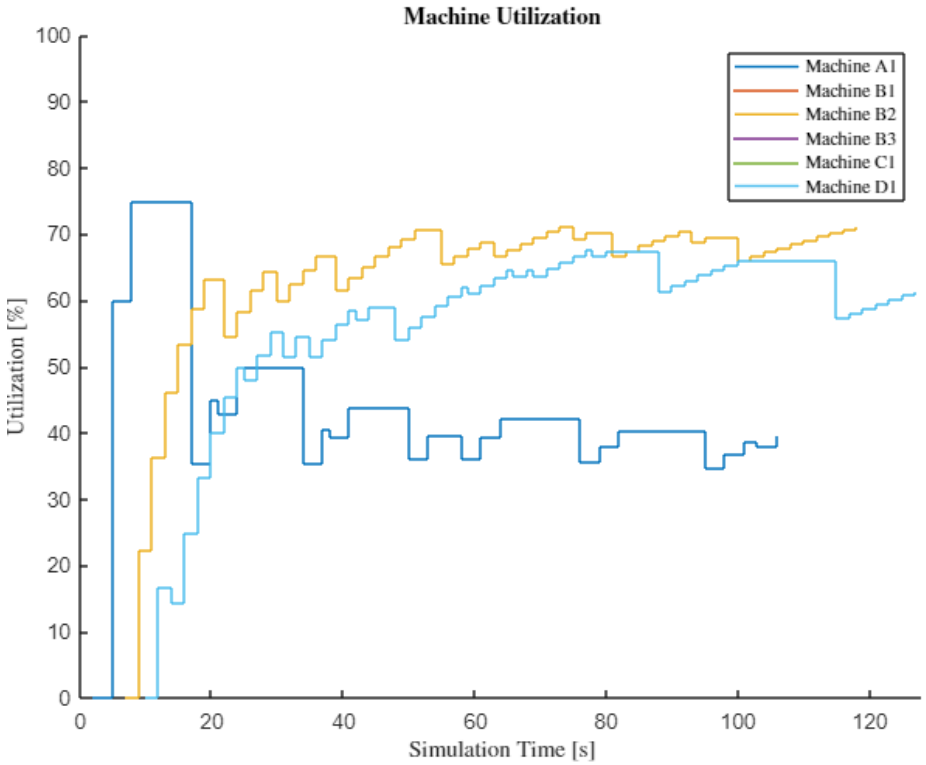

Diagram 30: Machine utilization for RL model (no charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

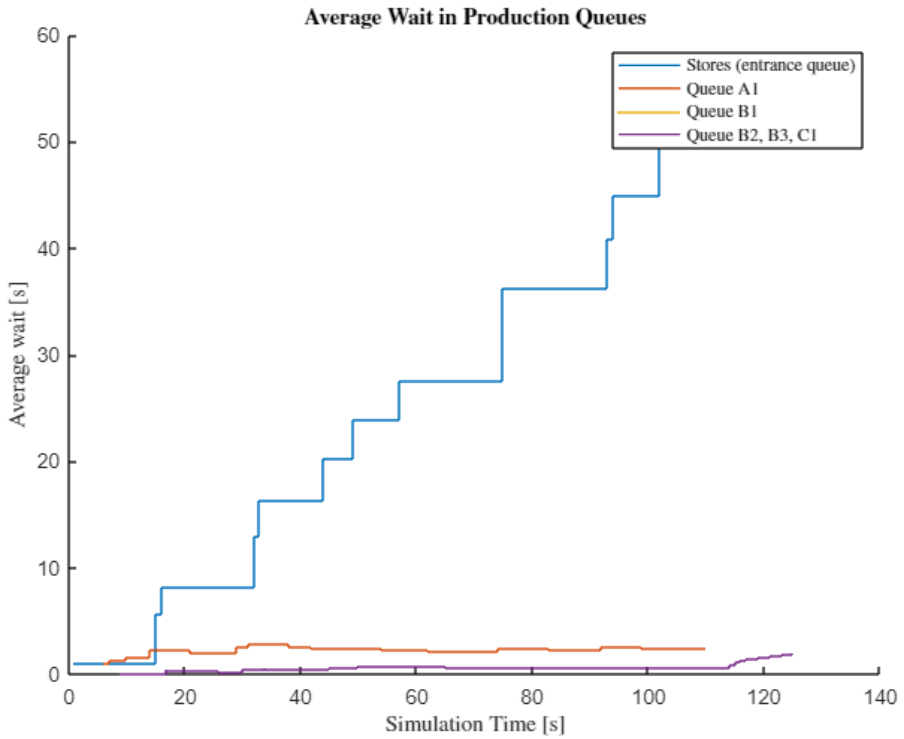

Diagram 31: Queue average waiting time for RL model (no charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

(b) 'Illegal' actions

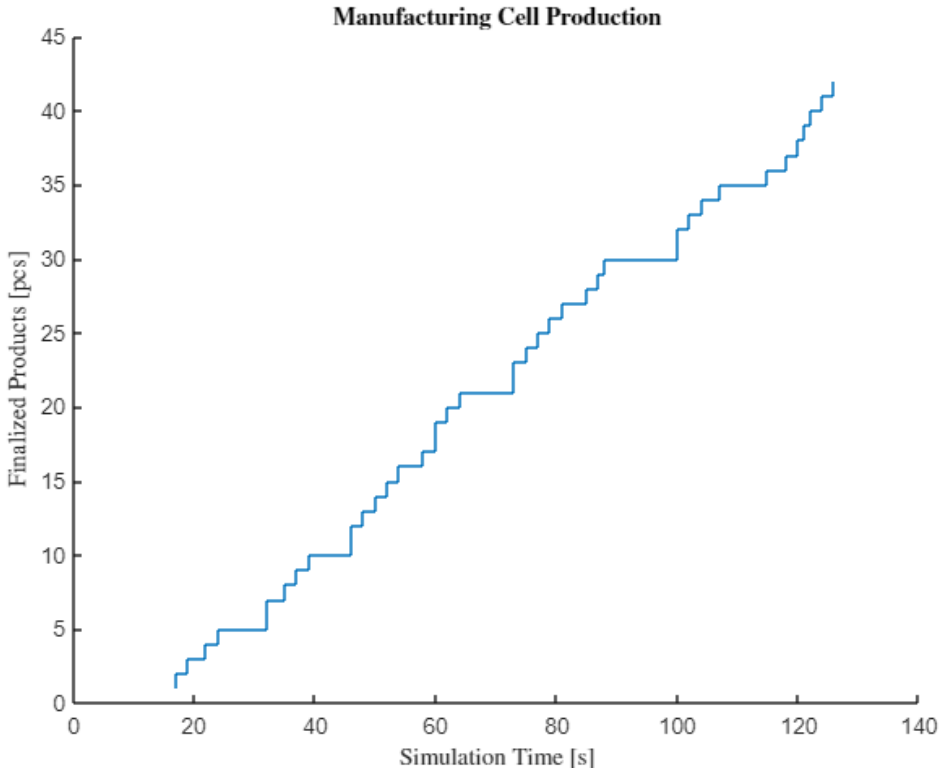

Diagram 32: Progress of manufacturing cell production for RL model (no charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

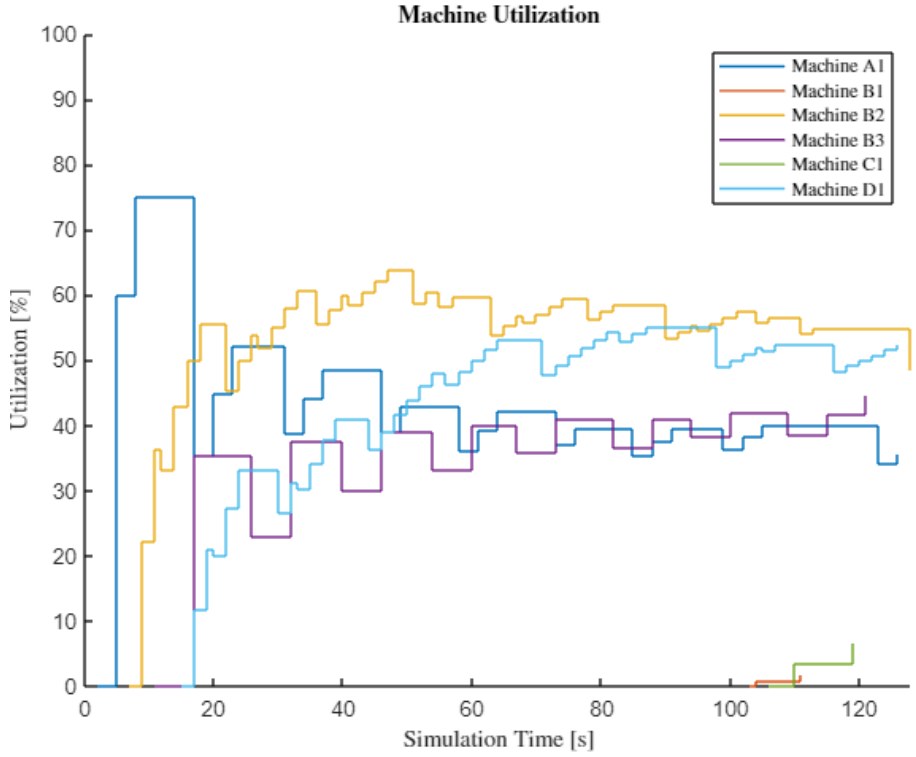

Diagram 33 : Machine utilization for RL model (no charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

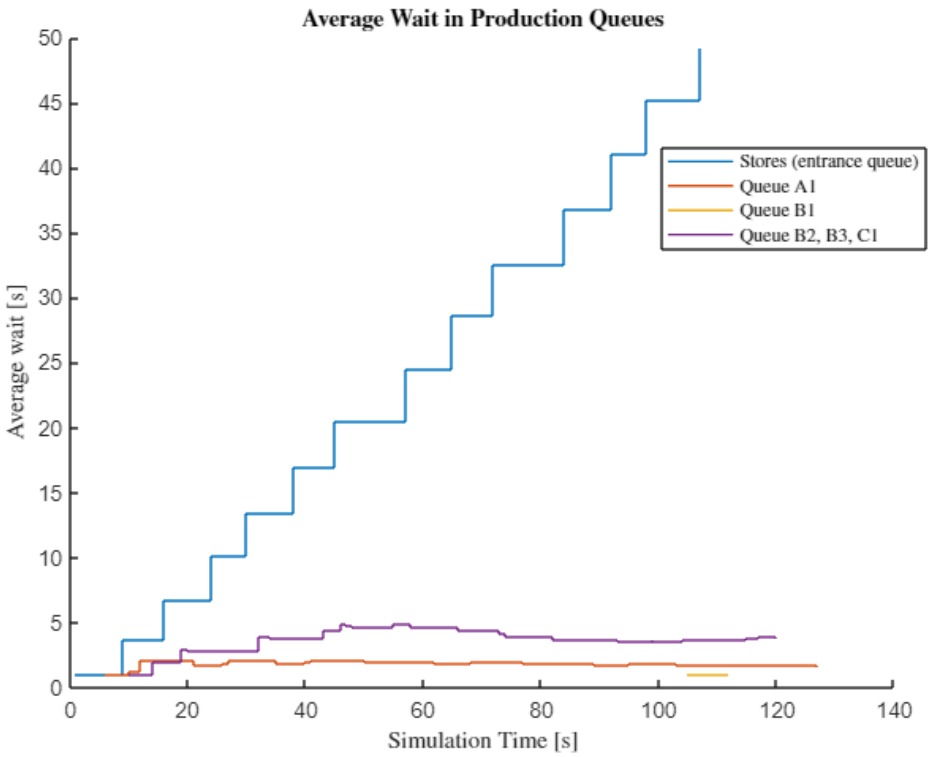

Diagram 34: Queue average waiting time for RL model (no charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

C. *FCFS model with 1 drone – With charging*

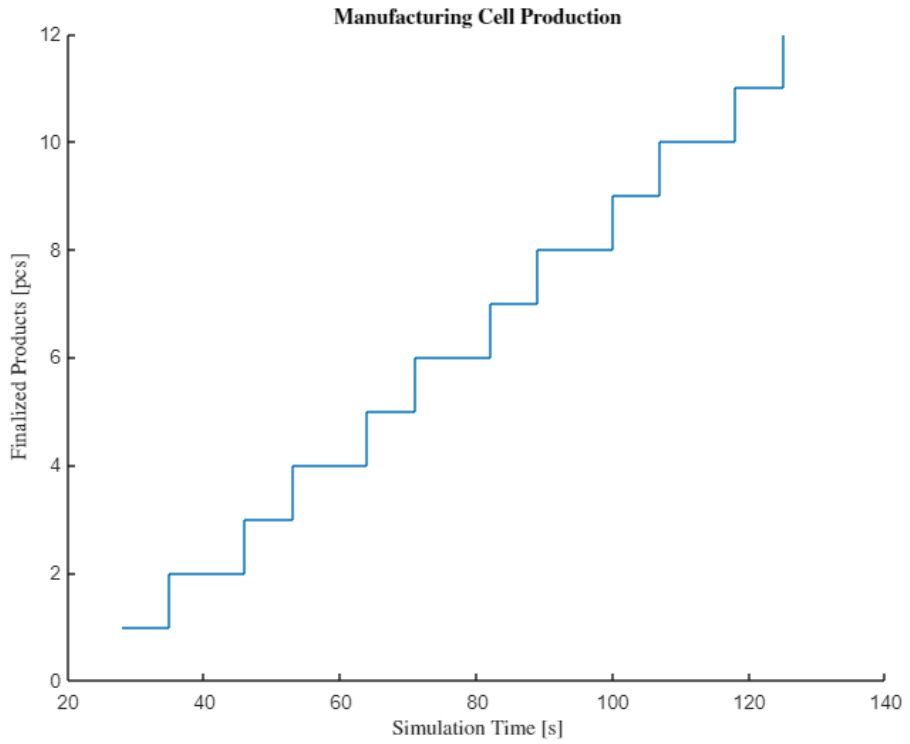

Diagram 35: Progress of manufacturing cell production for FCFS model (with charging) (Scenario 1)

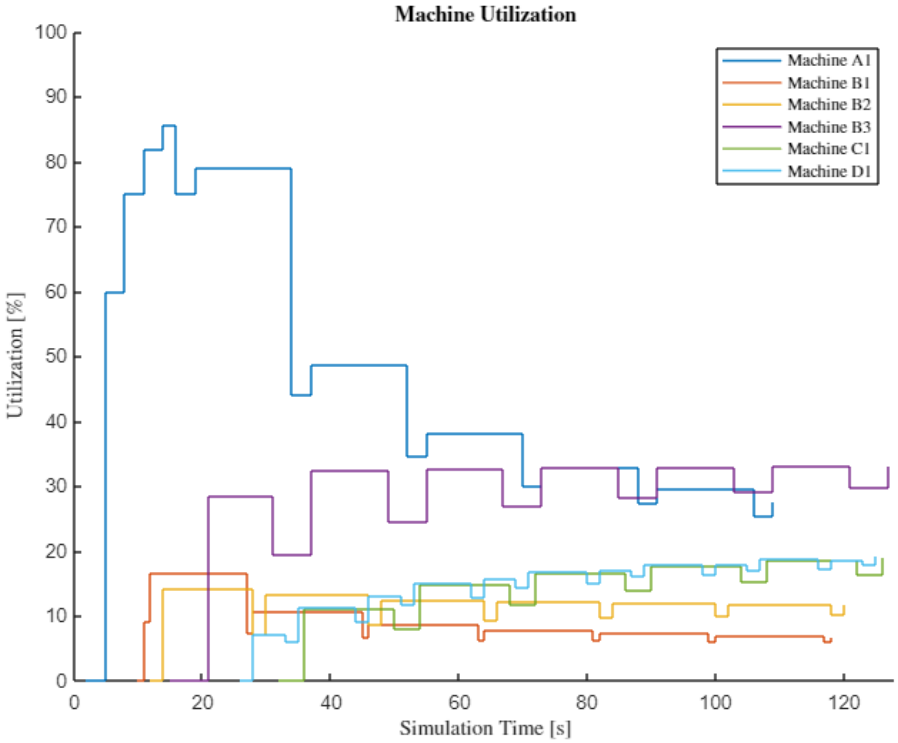

Diagram 36: Machine utilization for FCFS model (with charging) (Scenario 1)

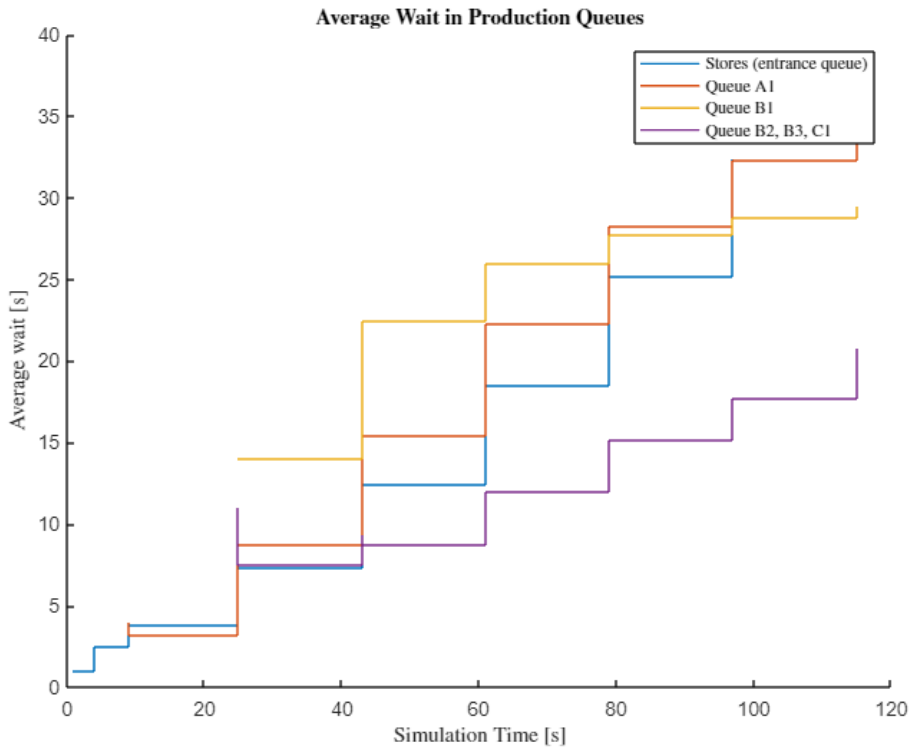

Diagram 37: Queue average waiting time for FCFS model (with charging) (Scenario 1)

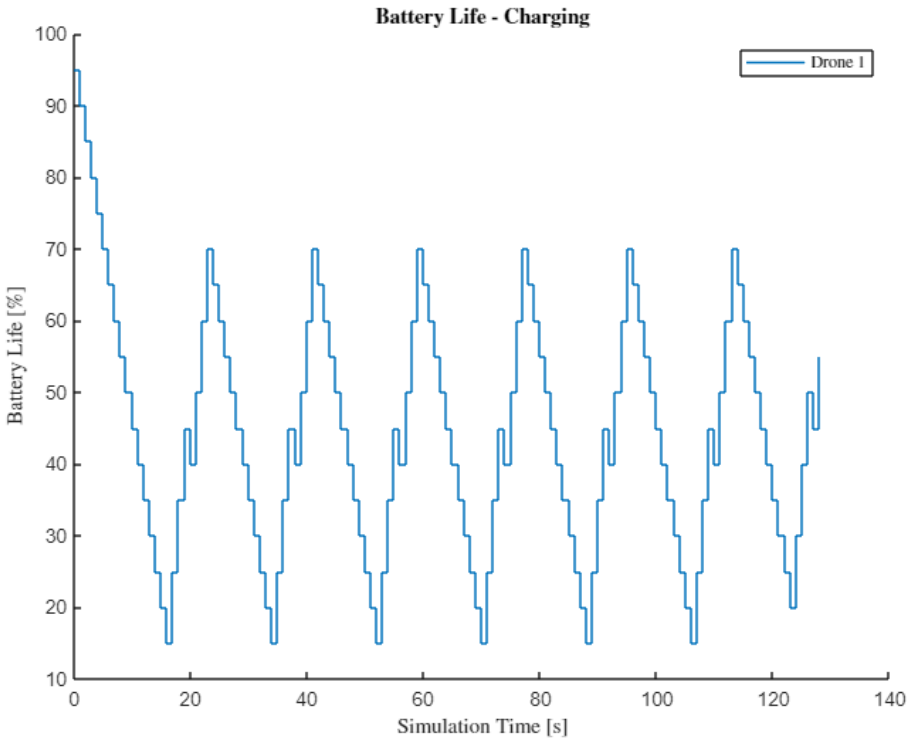

Diagram 38: Drone battery charging progress for FCFS model (with charging) (Scenario 1)

D. RL model with 1 drone – With charging

Hyperparameter	Value
Entropy Loss Weight	0.01
Discount Factor	0.95
Number of Epochs	3
Experience Horizon	128
Mini Batch Size	16 (x 8)

Table 8: RL hyperparameters for agent training for RL model (with charging)

Battery Parameters	Value
Drain Average	2%
Drain Per Tick	2%
Charge Per Tick	10%

Table 9: Drone battery charging parameters

i. Same learning rate between actor-critic ($\varepsilon_{actor} = \varepsilon_{critic} = 0.001$)

(a) 'Unconstrained' actions

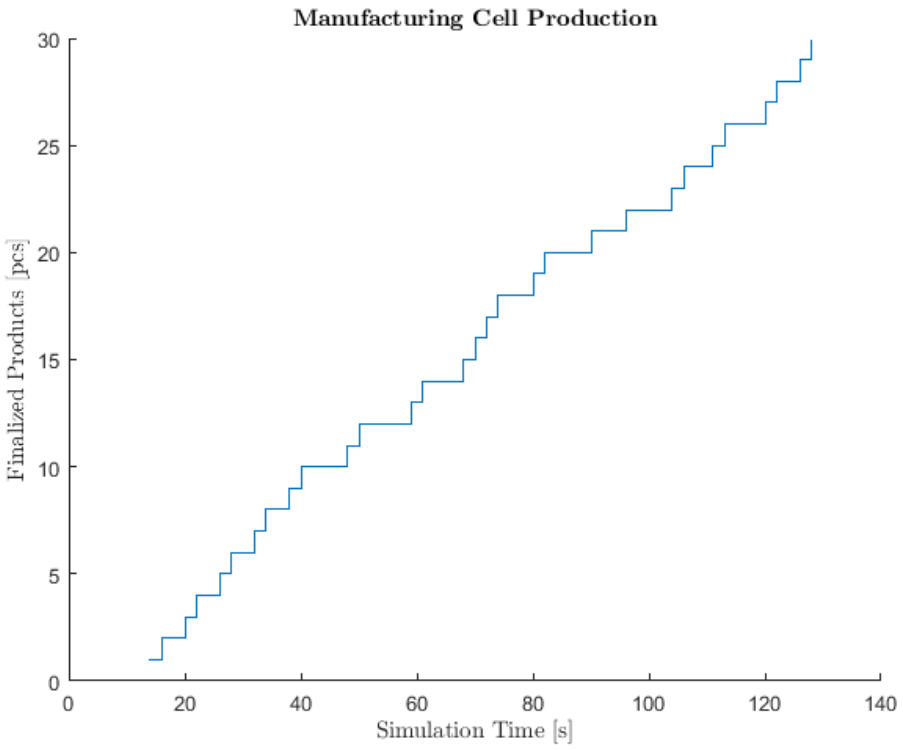

Diagram 39: Progress of manufacturing cell production for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

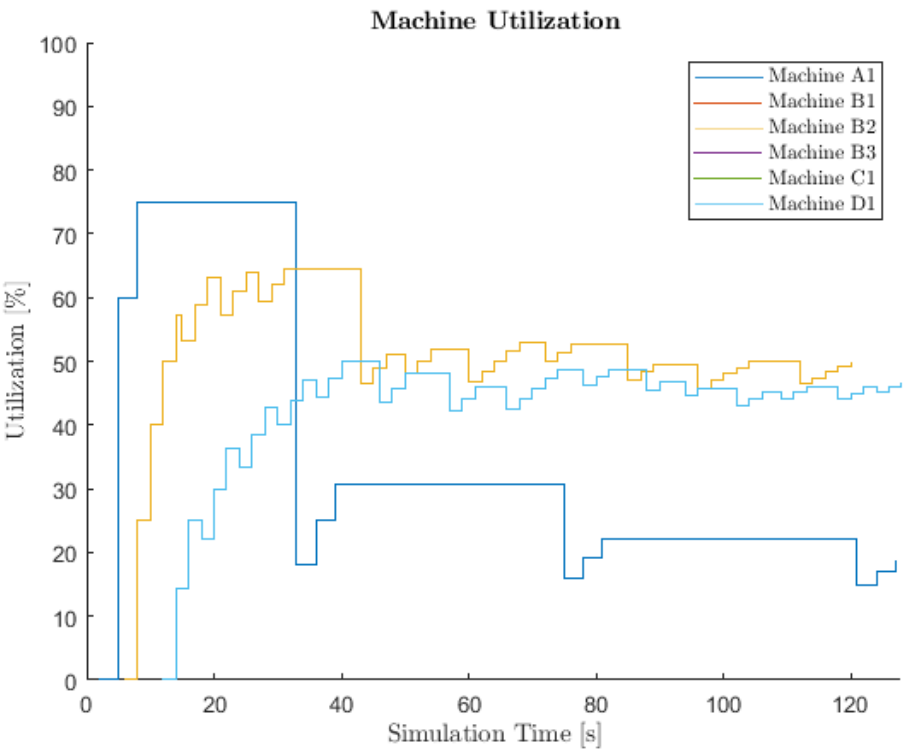

Diagram 40: Machine utilization for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

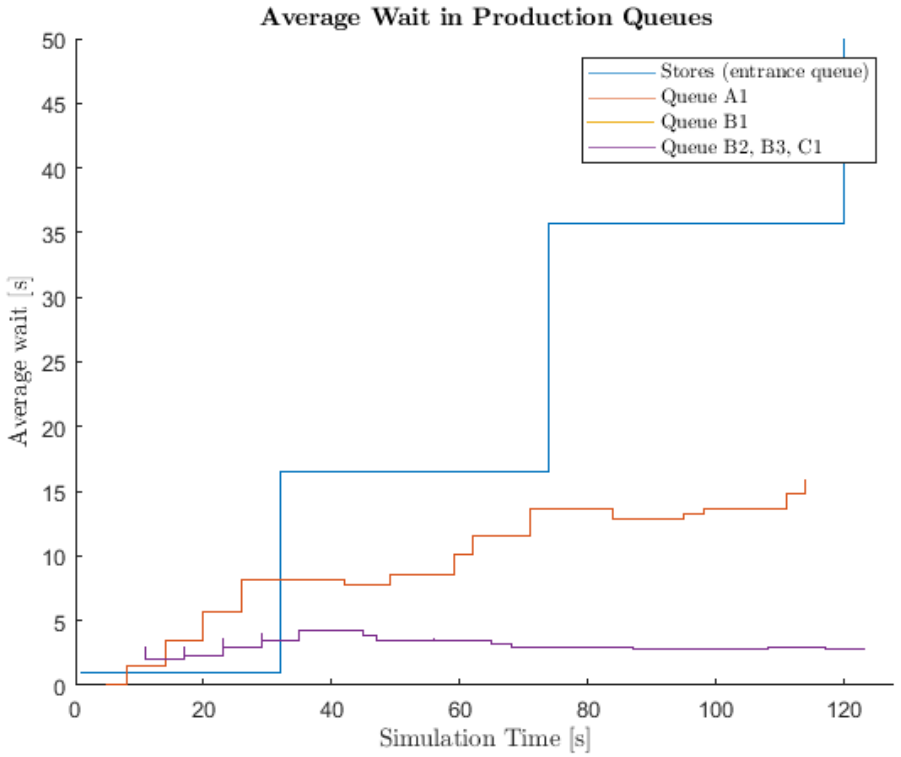

Diagram 41: Queue average waiting time for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

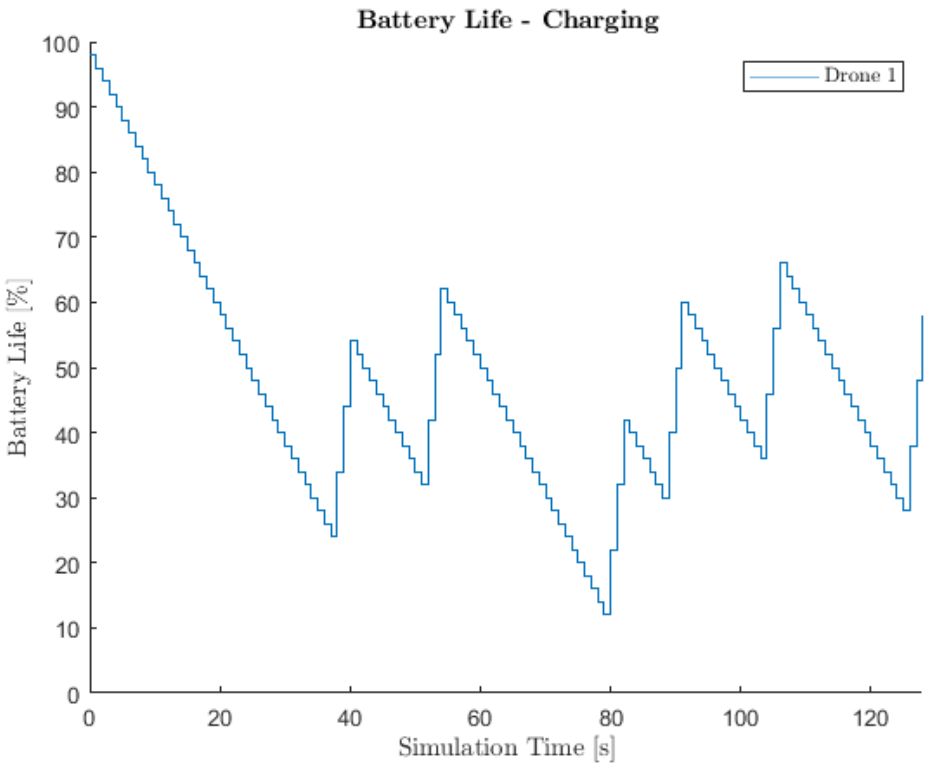

Diagram 42: Drone battery charging progress for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 1)

(b) 'Illegal' actions

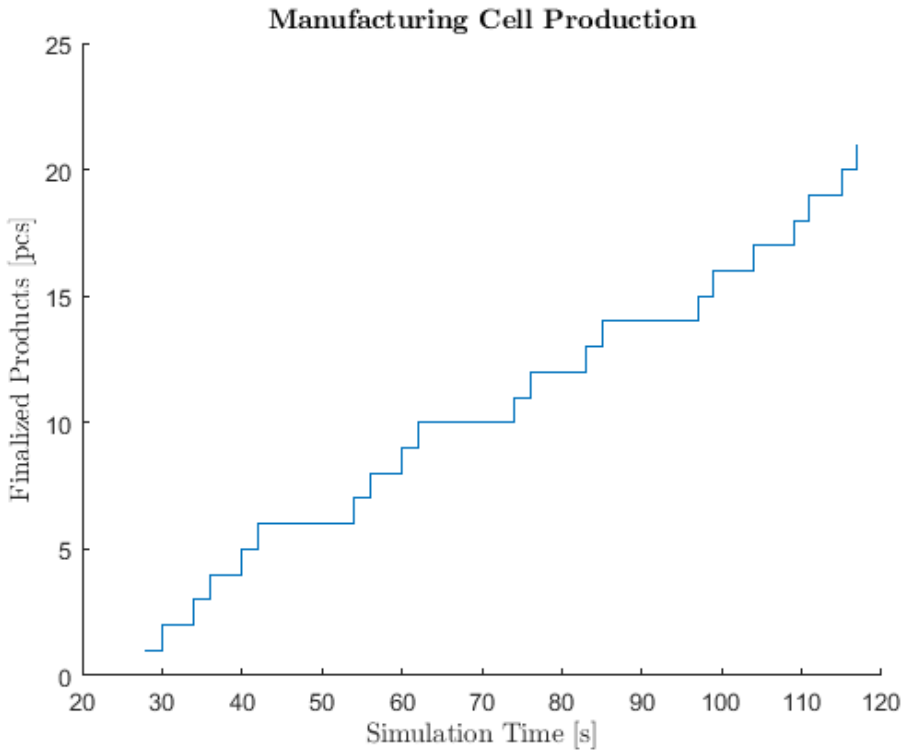

Diagram 43: Progress of manufacturing cell production for RL model (with charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

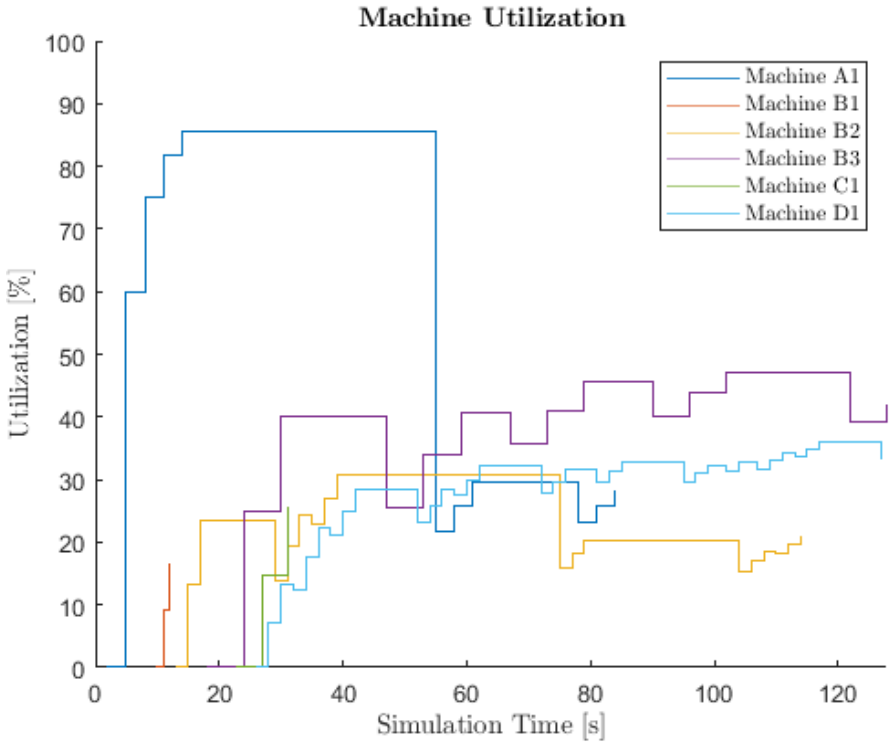

Diagram 44: Machine utilization for RL model (with charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

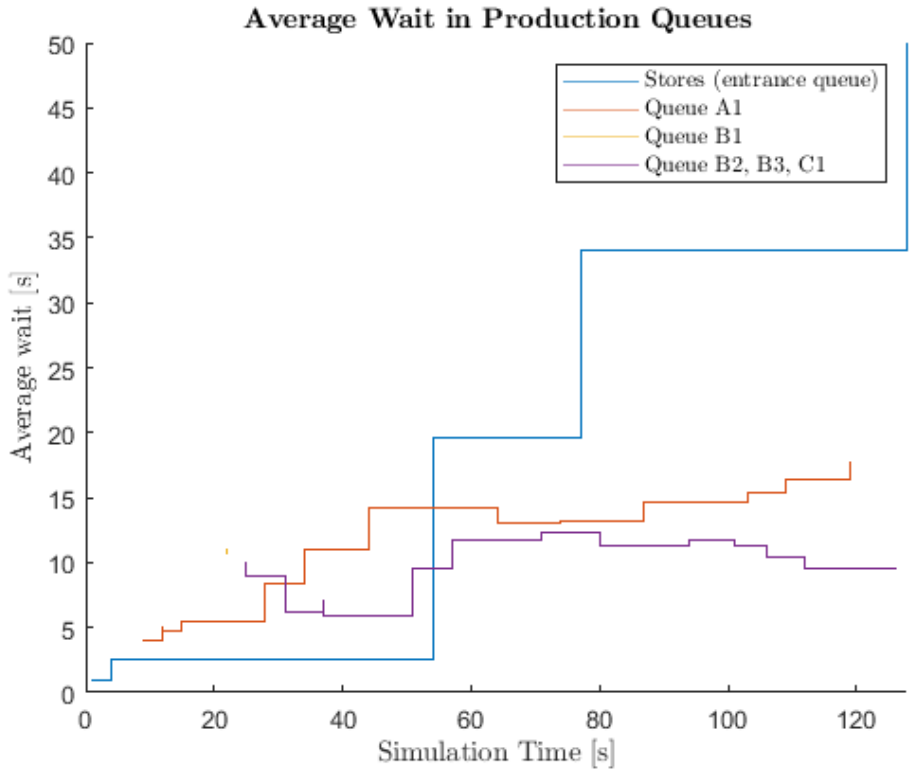

Diagram 45: Queue average waiting time for RL model (with charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

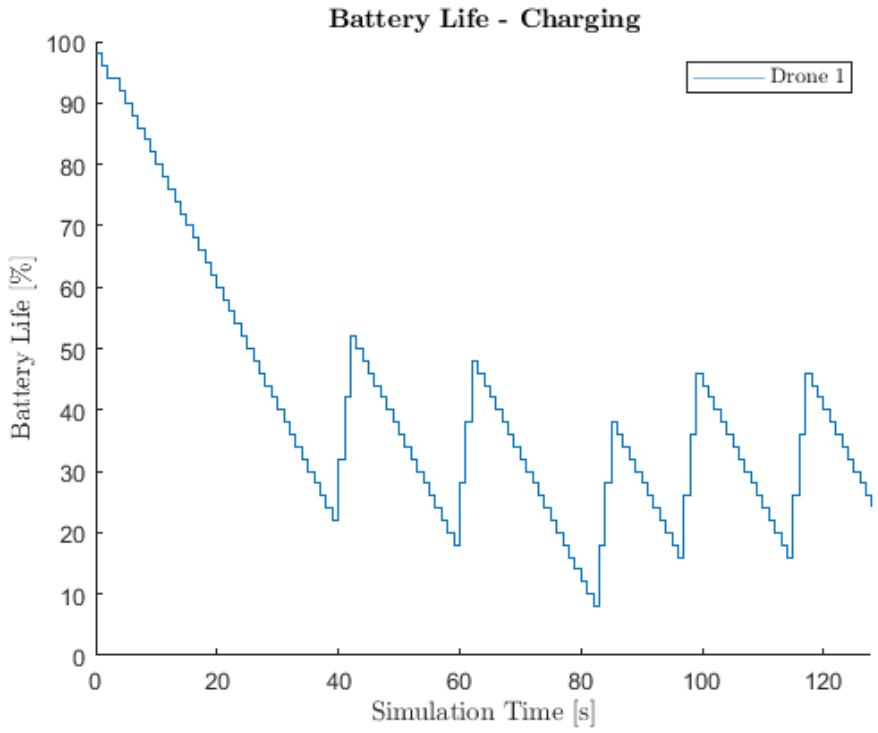

Diagram 46: Drone battery charging progress for RL model (with charging) for same actor-critic learning rate and 'illegal' case (Scenario 1)

ii. Different learning rate between actor-critic ($\varepsilon_{actor} = 0.008$, $\varepsilon_{critic} = 0.005$)

(a) 'Unconstrained' *actions*

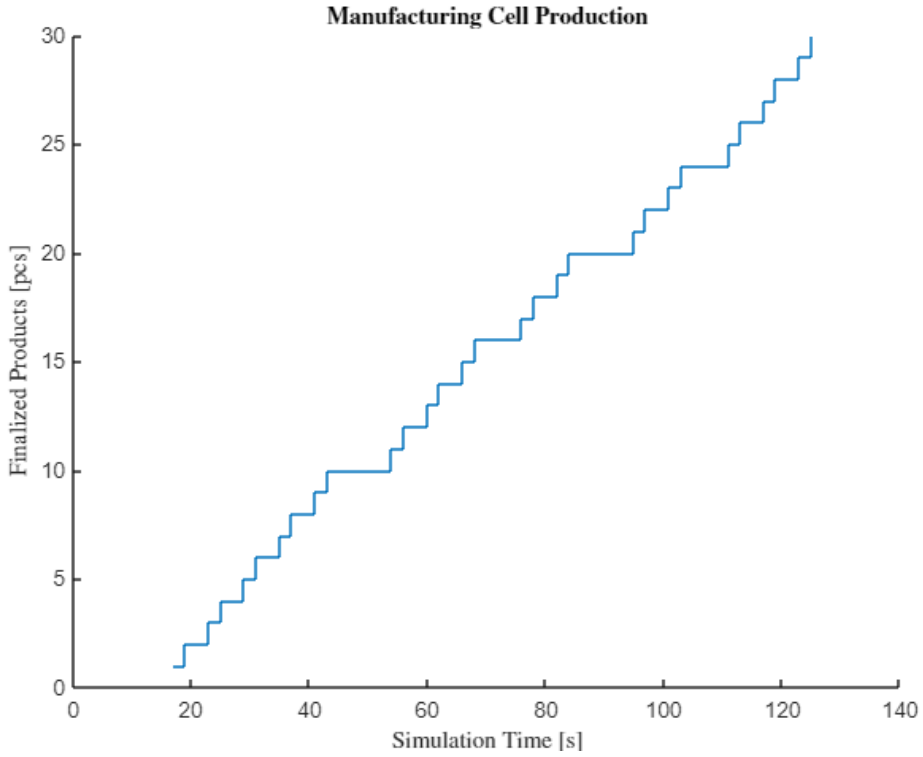

Diagram 47: Progress of manufacturing cell production for RL model (with charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

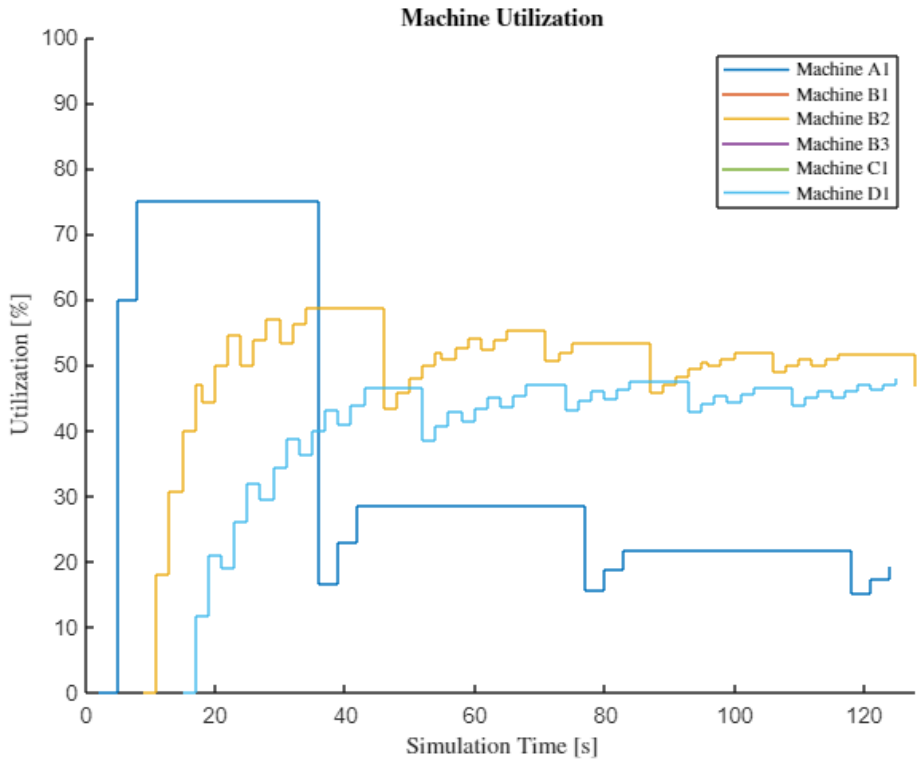

Diagram 48: Machine utilization for RL model (with charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

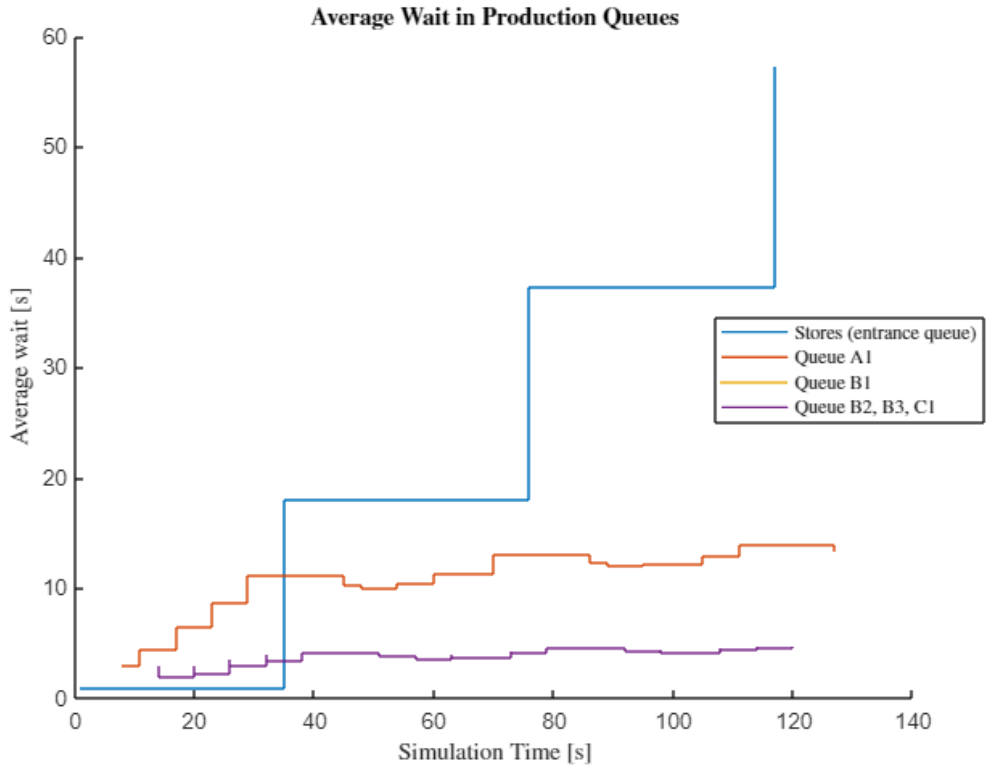

Diagram 49: Queue average waiting time for RL model (with charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

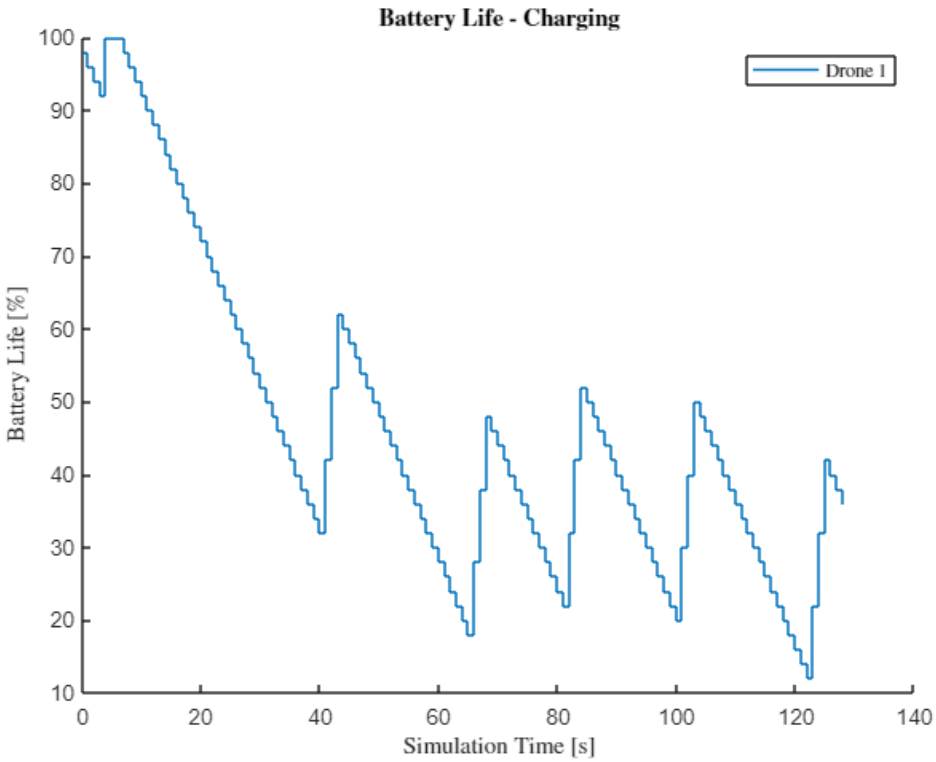

Diagram 50: Drone battery charging progress for RL model (with charging) for different actor-critic learning rate and 'unconstrained' case (Scenario 1)

(b) 'Illegal' actions

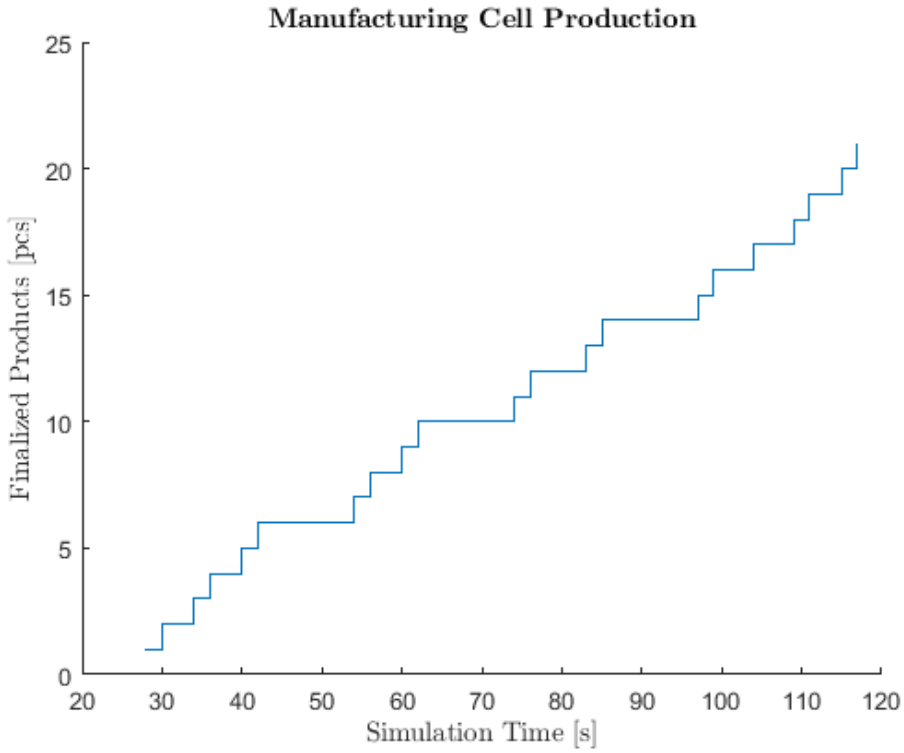

Diagram 51: Progress of manufacturing cell production for RL model (with charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

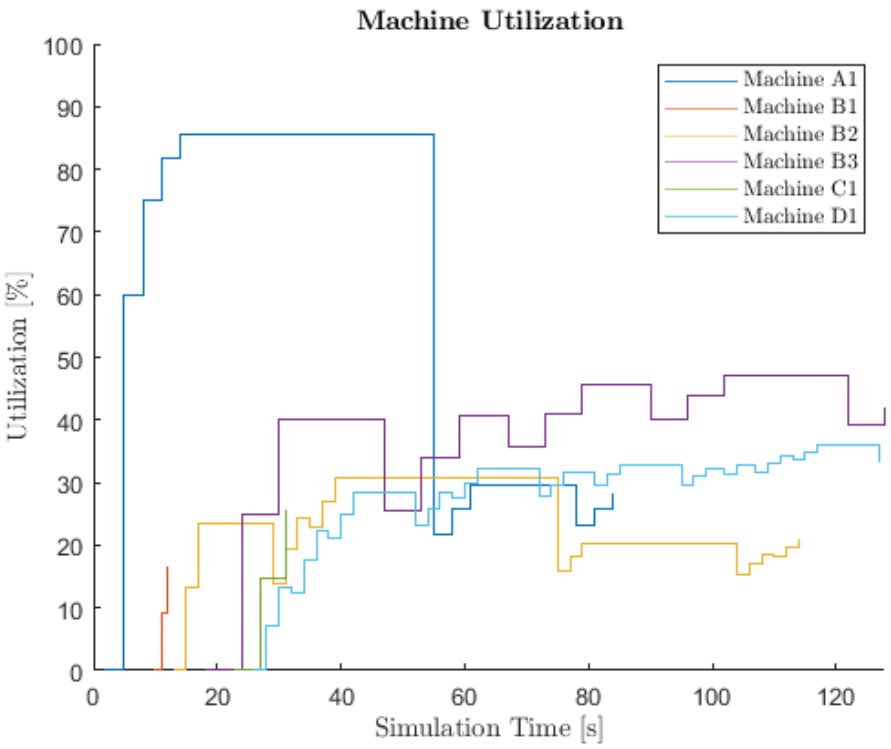

Diagram 52: Machine utilization for RL model (with charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

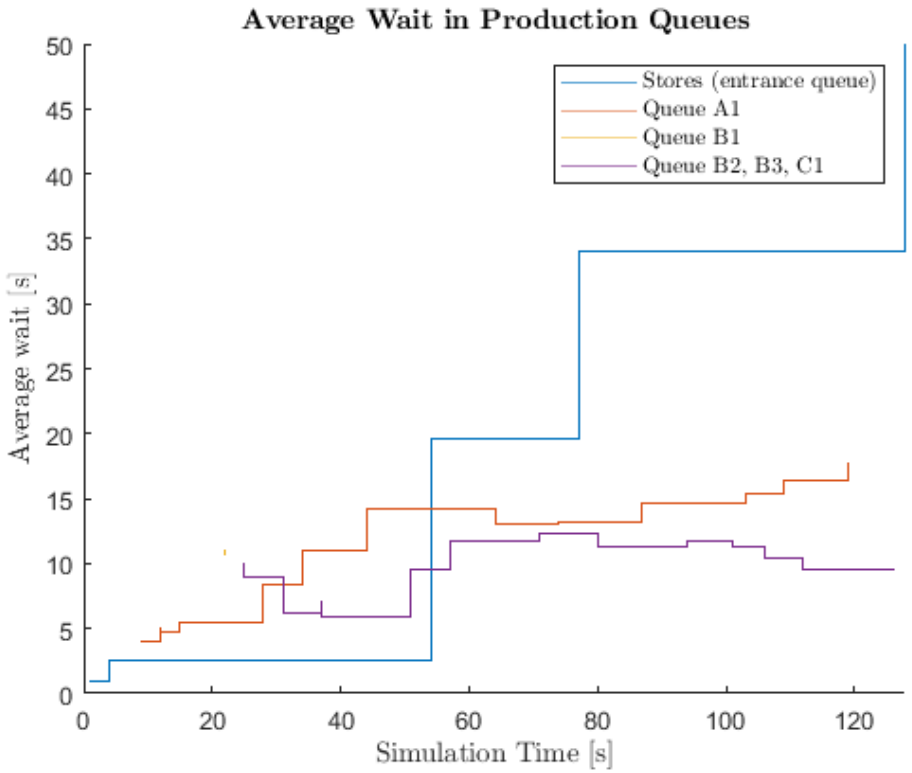

Diagram 53: Queue average waiting time for RL model (with charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

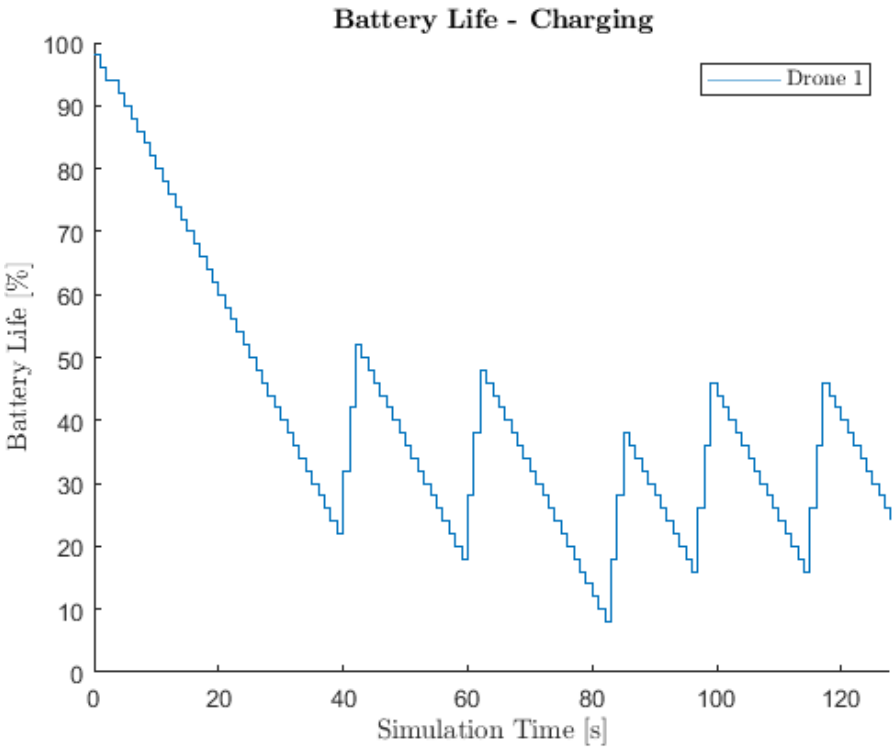

Diagram 54: Drone battery charging progress for RL model (with charging) for different actor-critic learning rate and 'illegal' case (Scenario 1)

Scenario 2

A. FCFS model with 1 drone – With charging

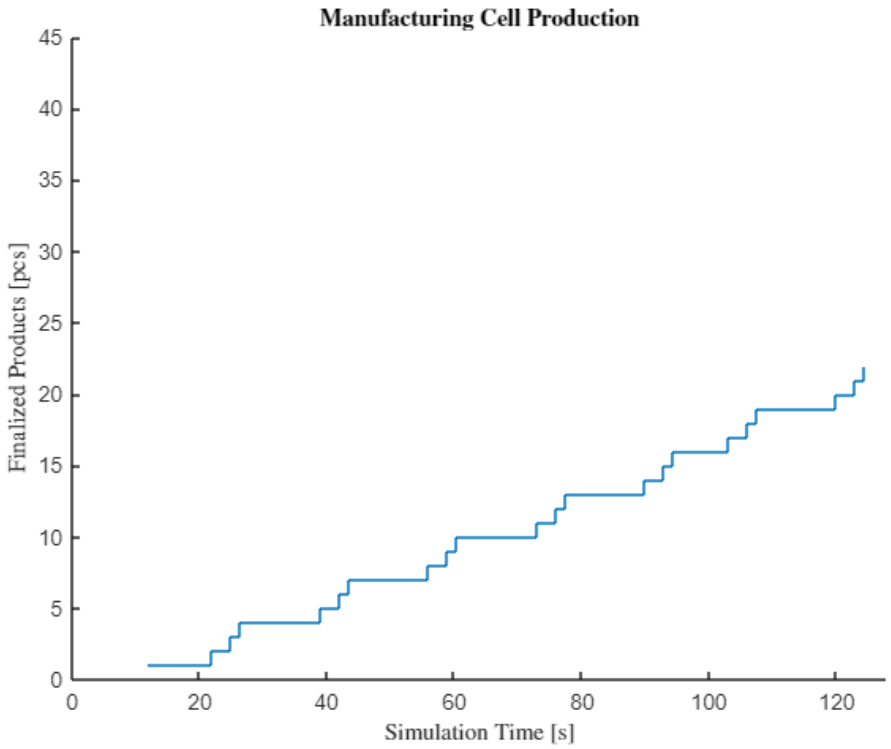

Diagram 55: Progress of manufacturing cell production for FCFS model (with charging) (Scenario 2)

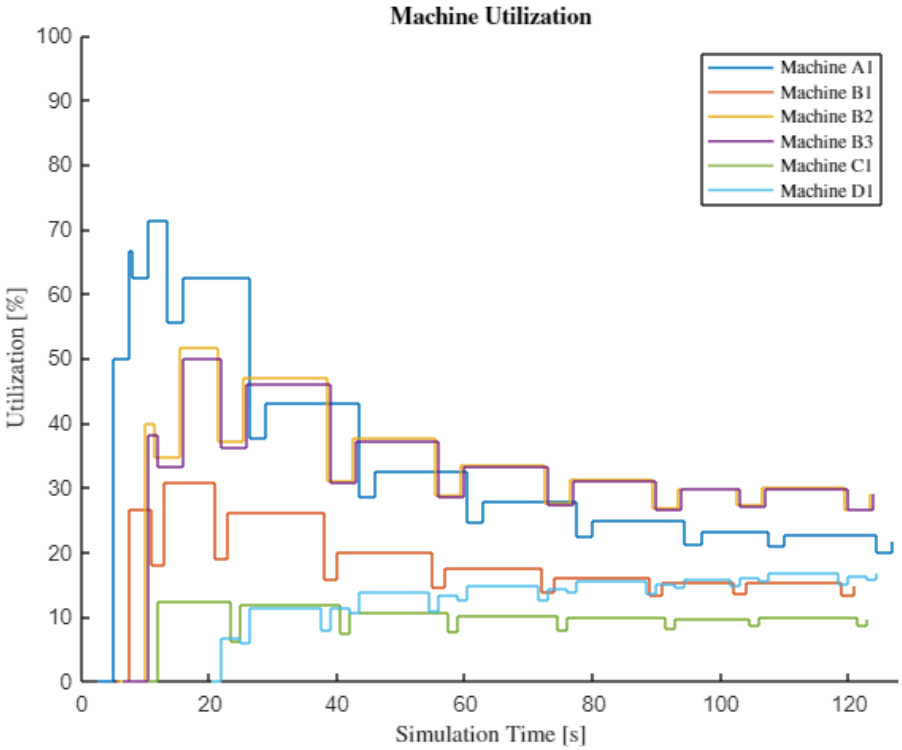

Diagram 56: Machine utilization for FCFS model (with charging) (Scenario 2)

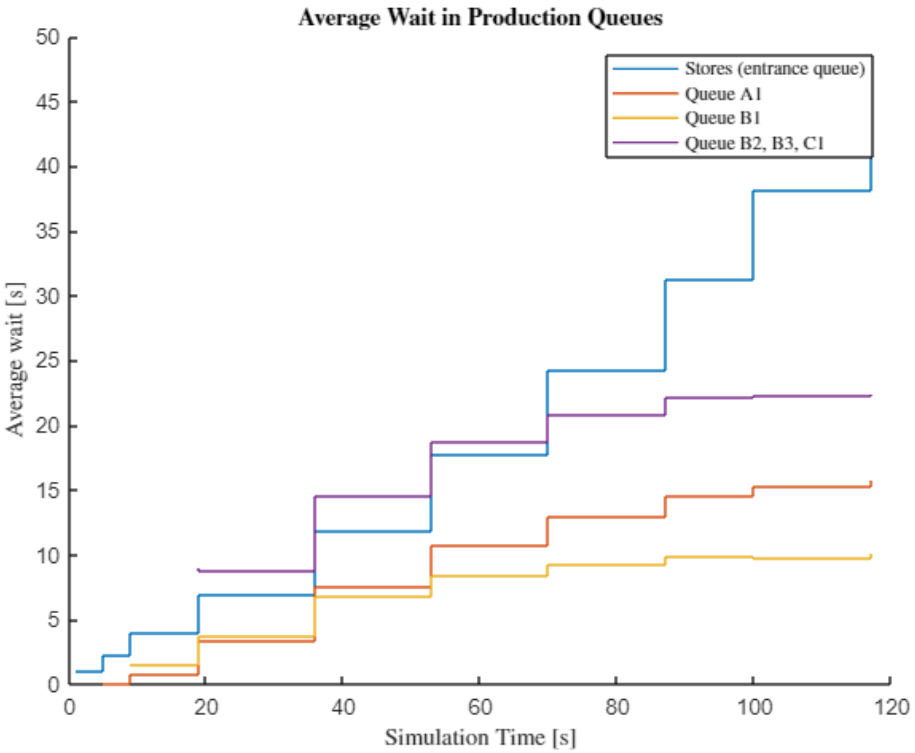

Diagram 57: Queue average waiting time for FCFS model (with charging) (Scenario 2)

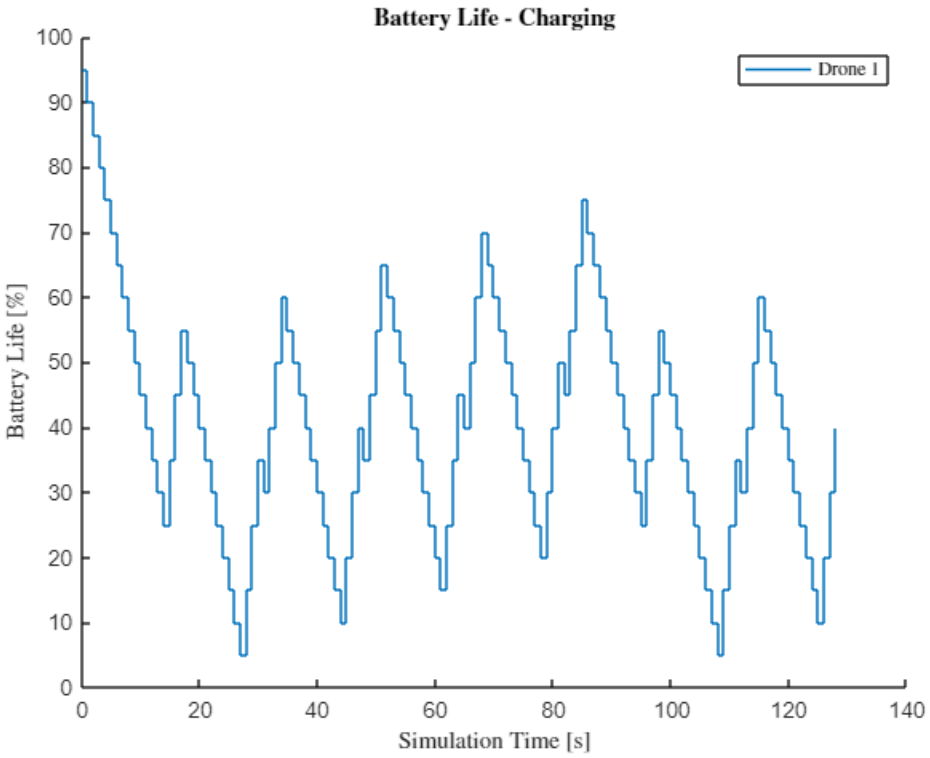

Diagram 58: Drone battery charging progress for FCFS model (with charging) for (Scenario 2)

B. RL model with 1 drone – With charging

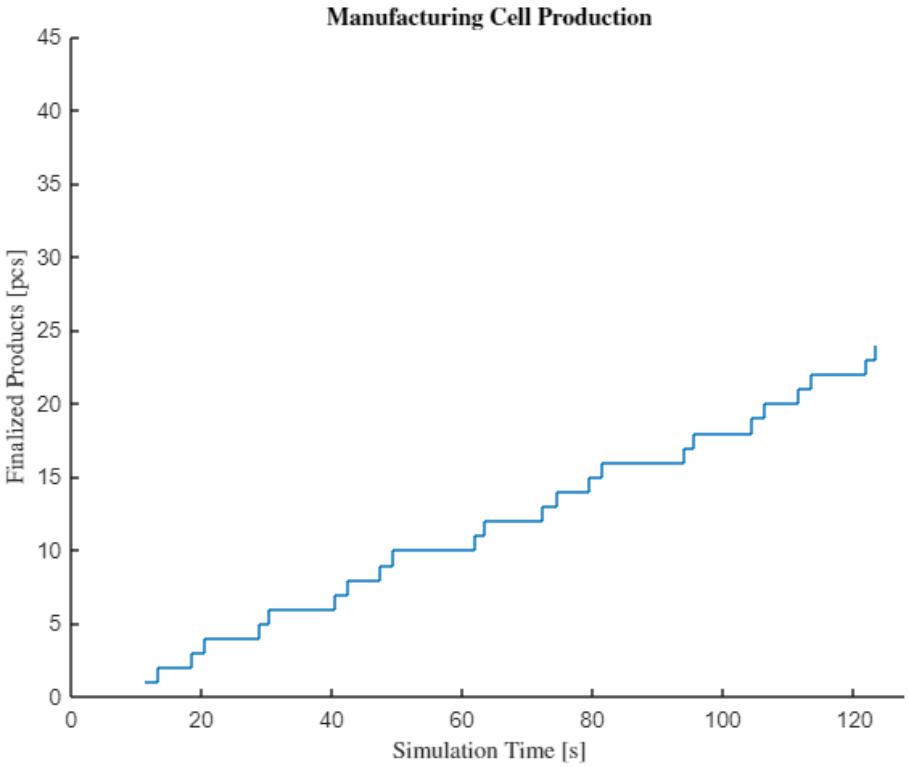

Diagram 59: Progress of manufacturing cell production for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 2)

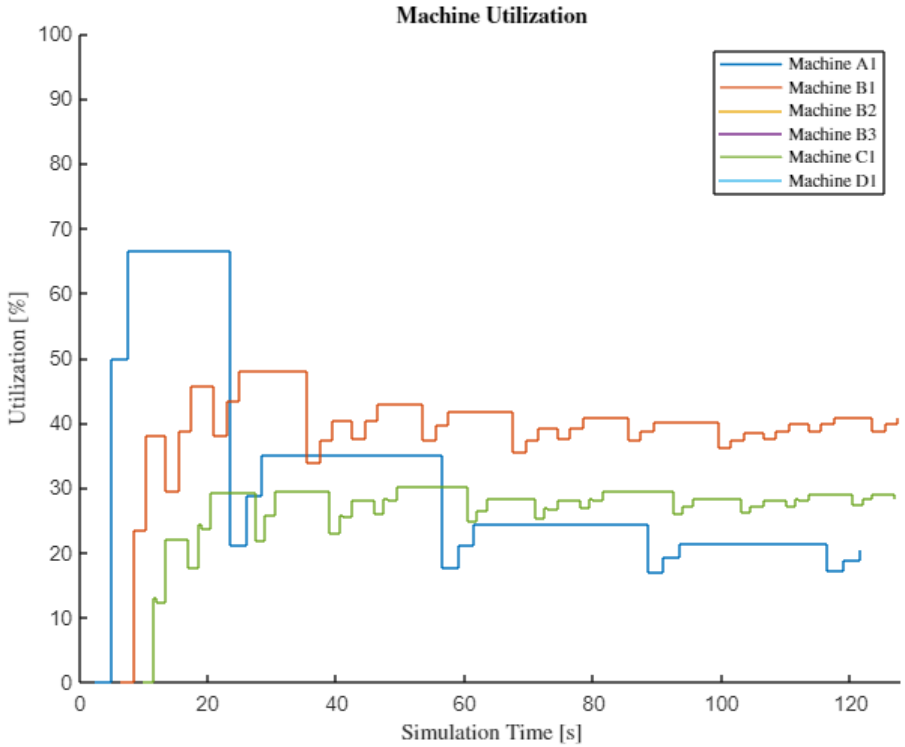

Diagram 60: Machine utilization for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 2)

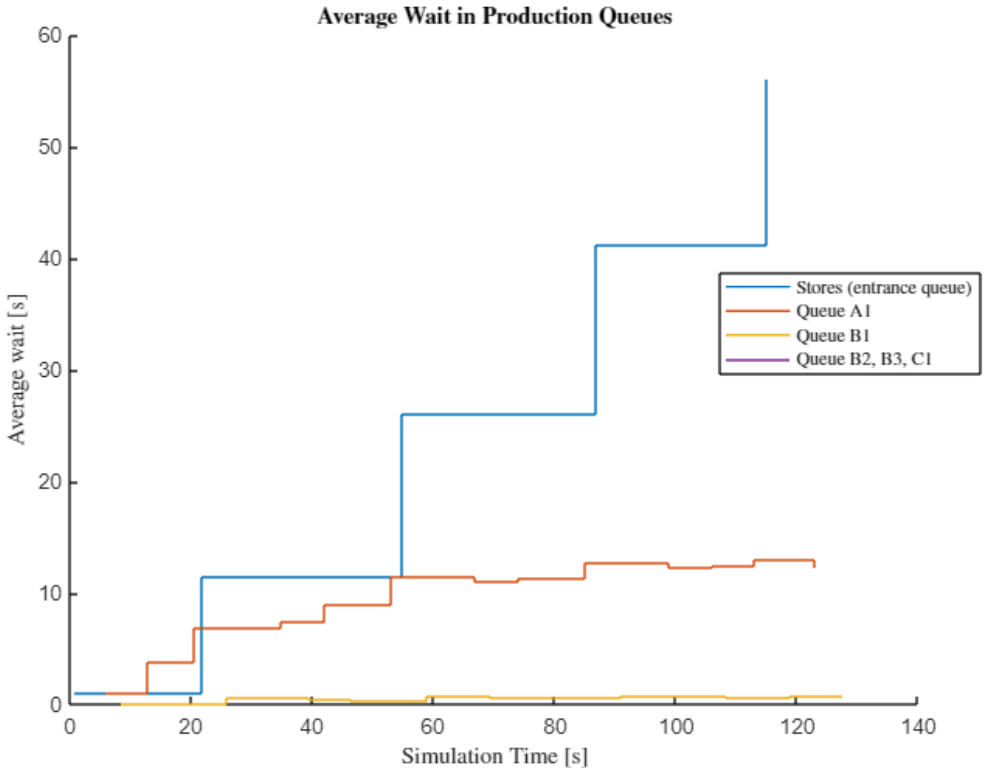

Diagram 61: Queue average waiting time for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 2)

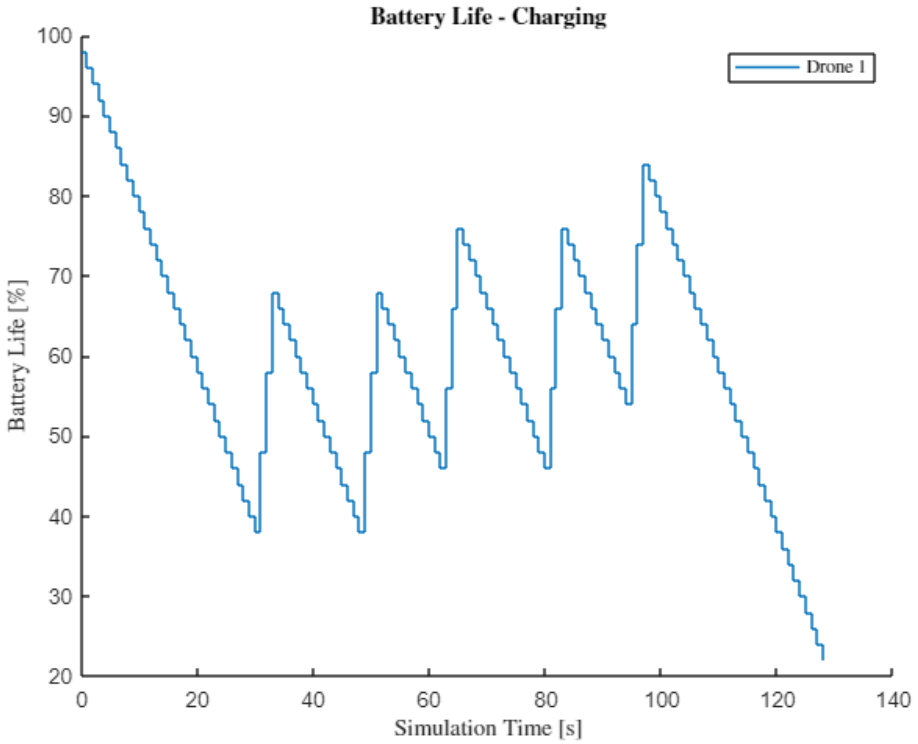

Diagram 62: Drone battery charging progress for RL model (with charging) for same actor-critic learning rate and 'unconstrained' case (Scenario 2)