**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**KALLIL MIGUEL CAPARROZ ZIELINSKI**

# CONTROLE FLEXÍVEL DE SISTEMAS A EVENTOS DISCRETOS UTILIZANDO SIMULAÇÃO DE AMBIENTE E APRENDIZADO POR REFORÇO

**DISSERTAÇÃO**

**PATO BRANCO**
**2021**

**KALLIL MIGUEL CAPARROZ ZIELINSKI**

# CONTROLE FLEXÍVEL DE SISTEMAS A EVENTOS DISCRETOS UTILIZANDO SIMULAÇÃO DE AMBIENTE E APRENDIZADO POR REFORÇO

## Flexible Control of Discrete Event Systems using Environment Simulation and Reinforcement Learning

Dissertação apresentada como requisito para obtenção do título de Mestre em Engenharia Elétrica, do Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Dalcimar Casanova
Coorientador: Prof. Dr. Marcelo Teixeira

**PATO BRANCO**

**2021**

**Ministério da Educação**
**Universidade Tecnológica Federal do Paraná**
**Campus Pato Branco**

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

KALLIL MIGUEL CAPARROZ ZIELINSKI

# CONTROLE FLEXÍVEL DE SISTEMAS A EVENTOS DISCRETOS UTILIZANDO SIMULAÇÃO DE AMBIENTE E APRENDIZADO POR REFORÇO

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Sistemas E Processamento De Energia.

Data de aprovação: 10 de Junho de 2021

Prof Dalcimar Casanova, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Joao Batista Florindo, Doutorado - Universidade Estadual de Campinas (Unicamp)

Prof Yuri Kaszubowski Lopes, Doutorado - Fundação Universidade do Estado de Santa Catarina (Udesc)

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 11/06/2021.

## ACKNOWLEDGEMENTS

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas e/ou instituições às quais presto minha homenagem. Certamente esses parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre estas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

A minha família, pelo carinho, incentivo e total apoio em todos os momentos da minha vida.

À minha namorada Marina, sempre do meu lado me incentivando a fazer o que gosto e fazendo de tudo para que isso aconteça.

Aos meus orientadores Dalcimar Casanova e Marcelo Teixeira, pela paciência e conhecimentos transmitidos ao longo desta jornada.

Enfim, a todos os que de alguma forma contribuíram para a realização deste trabalho.

# RESUMO

ZIELINSKI, Kallil M. C.. **Controle Flexível de Sistemas a Eventos Discretos Utilizando Simulação de Ambiente e Aprendizado por Reforço**. 2021. 65 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Tecnológica Federal do Paraná. Pato Branco, 2021.

*Sistemas a Eventos Discretos* (SEDs) são modelados classicamente com *Máquinas de Estados Finitos* (MEFs), e possuem máxima permissividade, controlabilidade e não bloqueabilidade utilizando a *Teoria de Controle Supervisório* (TCS). Enquanto a TCS é poderosa para lidar com eventos de um SED, ela falha ao processar eventos em que o controle é baseado em premissas probabilísticas. Neste documento, mostramos que alguns eventos podem ser tratados comumente na TCS, enquanto outros podem ser processados utilizando Inteligência Artificial. Primeiro apresentamos uma ferramenta para converter controladores da TCS em simulações de ambientes de Aprendizado por Reforço (AR), em que eles se tornam suscetíveis a processamento inteligente. Em sequência, propomos uma abordagem baseada em AR que reconhece o contexto em que um conjunto de eventos estocásticos ocorre e os trata de acordo, buscando uma tomada de decisões como complemento dos caminhos determinísticos da TCS. O resultado é uma eficiente combinação de um controle flexível e seguro, que tende a maximizar o desempenho de um SED que evolui de maneira probabilística. Dois algoritmos de AR são testados: SARSA e N-STEP SARSA, sobre uma planta automotiva controlada flexível. Os resultados sugerem um aumento de 9 vezes no desempenho utilizando a combinação proposta em comparação com decisões não inteligentes.

**Palavras-chave:** Sistemas a Eventos Discretos. Sistemas de Manufatura. Suporte a Decisões. Controle de Processos. Aprendizado por Reforço.

## ABSTRACT

ZIELINSKI, Kallil M. C.. **Flexible Control of Discrete Event Systems using Environment Simulation and Reinforcement Learning**. 2021. 65 p. Dissertation (MSc in Electrical Engineering) – Universidade Tecnológica Federal do Paraná. Pato Branco, 2021.

*Discrete Event Systems* (DESs) are classically modeled as *Finite State Machines* (FSMs), and controlled in a maximally permissive, controllable, and nonblocking way using *Supervisory Control Theory* (SCT). While SCT is powerful to orchestrate events of DESs, it fail to process events whose control is based on probabilistic assumptions. In this research, we show that some events can be approached as usual in SCT, while others can be processed apart using Artificial Intelligence. We first present a tool to convert SCT controllers into Reinforcement Learning (RL) simulation environments, from where they become suitable for intelligent processing. Then, we propose a RL-based approach that recognizes the context under which a selected set of stochastic events occur, and treats them accordingly, aiming to find suitable decision making as complement to deterministic outcomes of the SCT. The result is an efficient combination of safe and flexible control, which tends to maximize performance for a class of DES that evolves probabilistically. Two RL algorithms are tested, State-Action-Reward-State-Action (SARSA) and N-step SARSA, over a flexible automotive plant control. Results suggest a performance improvement 9 times higher when using the proposed combination in comparison with non-intelligent decisions.



**Keywords:** Discrete Event Systems. Manufacturing Systems. Decision Support. Process Control. Reinforcement Learning.

## LIST OF FIGURES

## LIST OF TABLES

**CONTENTS**

# 1 INTRODUCTION

Industrial systems have a high demand for flexibility, whether to minimize cost, maximize production, or synchronize actuators to finish or speed up manufacturing. The idea is to transform raw material into consumable products by integrating people, equipment, and technology (Groover 2011). For that, the engineering relies on models, methods, and tools that together integrate physical and virtual worlds through logical control structures that respond to the environment automatically, in time, and with the expected precision.

From the factory automation perspective, systems can be described by the way they evolve over time. In general, factory devices are event-driven, i.e., they are guided by the occurrence of asynchronous, time-independent, *events* that map physical signals. Systems that share these features are called *Discrete Event Systems* (DESs) (Cassandras and Lafortune 2009) and they can be modeled mathematically as *Finite State Machines* (FSMs). For the purpose of control, FSMs can be further processed by *Supervisory Control Theory* (SCT) (Ramadge and Wonham 1989) algorithms that coordinate the whole factory in a controllable, maximally permissive, modular, and nonblocking way.

While the SCT is recognized as an efficient method to synthesize industrial controllers, its application over emerging industrial scenarios is limited. Modern factories have moved from static to customizable control architectures that detect and react at real-time to changes in the physical layer, which also operates under multiple modes (Park and Febriani 2019). *Flexible Manufacturing Systems* (Manu *et al.* 2018), for example, are expected to produce simultaneously multiple types of products, using the same plant and distinct operations. When combined with modern computation technology, such as Big Data, Artificial Intelligence, and Internet of Things, flexible manufacturing aligns with *Industry 4.0* policies (Harrison *et al.* 2016). The price to be paid is a substantial increasing on the complexity to program context-sensitive controllers subject to reconfiguration. In this case, neither usual software development paradigms nor formal methods (like SCT) are fully appropriate.

The literature has tried to align SCT capabilities and emerging flexible manufacturing, converging efforts towards the industry of the future. Event distinctions (Cury *et al.* 2015), variables (Teixeira *et al.* 2015), and context recognition (Silva *et al.* 2017), for example, add to FSMs capabilities to handle multi-contexts over single same communication channels; access and identity preserving against intruders are addressed in

(Mohajerani and Lafortune 2020); large-scale and context-aware SCT problems are approached modularly in (Teixeira *et al.* 2018), or incrementaly in (Mohajerani *et al.* 2017), helping to reduce computational complexity either to synthesize or implement controllers.

Despite extensions and improvements, the SCT essence is still not aligned with optimization, flexibility, and self-adaptation properties, in addition to safety. The main difficulty is the integration of solid, deterministic, event decision under which a DES is exposed and treated (including modeling, synthesis, and implementation steps), with intelligent algorithms capable of discovering variable meanings for events whose control depends on stochastic decisions. Without this combination, the controller has to be replaced after each context switching of a flexible system, which can be quite often.

In this work, (static) safety properties are still ensured deterministically, by event disabling assumptions as usual in SCT, while (flexible) customization properties are calculated by processing artificial intelligence. It is shown that some events of a DES can be chosen and kept apart from the SCT synthesis, to be processed using *Reinforcement Learning* (RL) techniques with the purpose of deciding which and when they are more appropriate to be allowed under control, in conjunction with the SCT decisions. The intelligent decisions are derived by an operating RL-based agent that observes the system under control and gathers to it experiences about its interaction through the environment. This agent acts like a *recommender* for events that are eligible to occur under the SCT control, but whose order or feasibility are unknown. Based on the experiences accumulated by the agent, it is capable of adapting its behavior, letting components to be controlled safely and with some level of unplanned optimization.

Our contributions can be summarized as follows: (i) a formal method that integrates SCT and RL techniques with a step guide to apply it; (ii) a tool that converts DES controllers into *Markov Decision Process* (MDP) that accepts RL processing with reward markings; (iii) a case study involving a flexible automotive plant.

Structurally, Section 1.1 presents some related works, and where our approach fits in relation to it; Section 2 introduces and exemplifies DES and SCT; Section 3 presents the background on RL; Section 4 presents the results of the method while Section 5 shows a real automotive example and finally perspectives and conclusions are presented in Sections 6 and 7.

## 1.1 RELATED LITERATURE

This section summarizes the state-of-art related with this work, which is organized as in Fig. 1. The dashed arrow identifies a research domain possible, but not taken in this research. The gray block identifies our main contributions and their internal relationship, which is explained as follows.

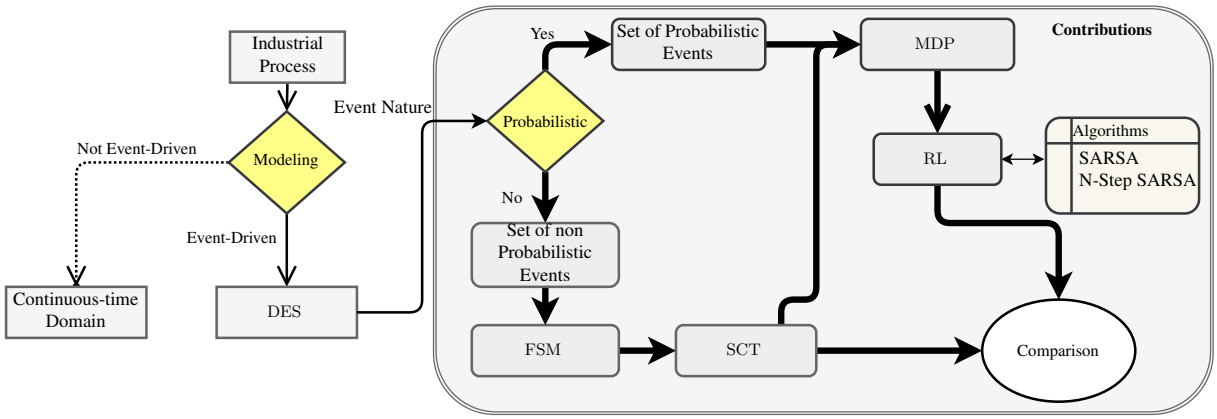



**Figure 1 – Classes of processes, state-of-the-art, and their relation with this work.**

We initially consider to split industrial processes into two main groups: the ones that are event-driven (DESs) (Cassandras and Lafortune 2009), and the ones that are not. Since manufacturing systems are classically seen as DESs, we focus this discussion on the first group, although sometimes it may be convenient to handle them in the continuous time domain (Roman *et al.* 2019; Turnip and Panggabean 2020; Precup *et al.* 2020).

When a system is seen as DES, it can be modeled by formal methods such as Petri Nets (Murata 1989), queue theory (Tiwari S. 2021), or FSMs (Ben-Ari and Mondada). The choice is not consensual in the literature, but a good measure is to observe the nature of the occurrence of events in the system, which can be probabilistic, or not. In general, probabilistic events are handled by timed versions of *Petri Nets* or *Automata*, or exposed as *Markov Decision Processes* (MDPs) (Puterman 1994) to be processed further using Fuzzy Logic (Gomes *et al.* 2021), *Reinforcement Learning* (RL) (Sutton and Barto 2018), *Deep Learning* (Mnih *et al.* 2015), or similar. Differently, non-probabilistic events are preferably modeled as FSMs to facilitate the specification (Mohajerani *et al.* 2021), synthesis (Ramadge and Wonham 1989), and verification (Malik and Ware 2020) processing.

In our approach, non-probabilistic events are handled by using FSMs, and specifications are enforced by processing SCT synthesis, as usual. The resulting controller, and the set of probabilistic events, are then joined and converted into MDP representations for further RL

processing.

It is possible to apply other machine learning algorithms than RL (Ahmed *et al.* 2019), but this implies that a predefined dataset has to be available, which removes part of the flexibility possibly learned by simulating an agent through the environment. It is also possible to handle SCT and RL separately. For example, SCT can be used for throughput optimization (Putten *et al.* 2020), context recognition (Silva *et al.* 2017), order preserving (Nooruldeen and Schmidt 2020) and synthesize successful hacker attacks (Matsui and Lafortune 2021), while RL studies focus on production scheduling (Waschneck *et al.* 2018), order dispatching (Stricker *et al.* 2018), and pricing optimization (Krasheninnikova *et al.* 2019).

However, this separation implies that each study returns different benefits for the automatic control practices, and leads to variable performance indexes. For example, SCT is well known by its efficiency for safety, but not for flexibility, while RL is appropriate to handle dynamic environments, but not for safety guarantees. As each approach adds specific advantages to the process control and is structured over a distinct framework, their combination is not straightforward. Therefore, this document is dedicated to combine them into a single framework that gathers their best features and supports in parallel the safe, flexible, and expert control of industrial process.

## 2 OVERVIEW ON EVENT-DRIVEN SYSTEMS

While part of the behavior of an industrial systems can be continuous in time, some events (e.g., sensing, actuation, faults) have a discrete nature, and variations (e.g., customer, production, cost profiles) are essentially stochastic. Systems that evolve according to the asynchronous occurrences of events are called *Discrete Event System* (DESs) (Cassandras and Lafortune 2009), and they can be modeled by *Finite State Machines* (FSMs).

Formally, an FSM is a tuple $G = \langle \Sigma, Q, q^\circ, Q^\omega, \rightarrow \rangle$ where: $\Sigma$ is finite set of *events*; $Q$ is a finite set of *states*; $Q^\omega \subseteq Q$ is a subset of marked states (in general associated with the idea of completing tasks); $q^\circ \in Q$ is the *initial state*; and $\rightarrow \subseteq Q \times \Sigma \times Q$ is the *transition relation*.

Events are taken from $\Sigma$ and used to compose *strings*. The set $\Sigma^*$ includes all finite strings, including the *empty string* $\varepsilon$. The transition relation between any two states $q, q' \in Q$ with the event $\sigma \in \Sigma$ is exposed as $q \xrightarrow{\sigma} q'$, and extended to strings in the standard way, i.e., $q \xrightarrow{s} q'$, for $s \in \Sigma^*$. Then, notation $q \xrightarrow{\sigma}$ means $q \xrightarrow{\sigma} q'$ for some state $q' \in Q$; $G \xrightarrow{\sigma}$ means $q^\circ \xrightarrow{\sigma} q$ for some state $q \in Q$; and $q \not\xrightarrow{\sigma}$ and $G \not\xrightarrow{\sigma}$ mean the respective event disabling.

As an industrial DES is usually formed by a set $J = \{1, \cdots, m\}$ of components, it is convenient to design each component as an FSM $G_j$, $j \in J$, and combine them by *synchronous composition* (denoted $\|$). Given two FSMs, $G_1 = \langle \Sigma_1, Q_1, q_1^\circ, Q_1^\omega, \rightarrow_1 \rangle$ and $G_2 = \langle \Sigma_2, Q_2, q_2^\circ, Q_2^\omega, \rightarrow_2 \rangle$, define $G_1 \| G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, (q_1^\circ, q_2^\circ), Q_1^\omega \times Q_2^\omega, \rightarrow \rangle$, where $\rightarrow$ is constructed as follows:

**(i)** $(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2')$ if $\sigma \in \Sigma_1 \cap \Sigma_2, q_1 \xrightarrow{\sigma} q_1', q_2 \xrightarrow{\sigma} q_2'$ ;

**(ii)** $(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2)$ if $\sigma \in \Sigma_1 \setminus \Sigma_2, q_1 \xrightarrow{\sigma} q_1'$ ;

**(iii)** $(q_1, q_2) \xrightarrow{\sigma} (q_1, q_2')$ if $\sigma \in \Sigma_2 \setminus \Sigma_1, q_2 \xrightarrow{\sigma} q_2'$ ;

**(iv)** undefined.

That is, events enabled by both $G_1$ and $G_2$ (case (i)) are merged into a single transition; events enabled by only one FSM and unknown by the other (cases (ii) and (iii)) are interleaved in any order; and all other cases are undefined (case (iv)), e.g., an event may be enabled by one FSM, and known but not enabled by the other. In this case, it is considered undefined and disabled by composition. Other examples are presented as supplementary material (Zielinski *et al.* 2021).

## 2.1 MODELING

When using FSMs to design a set of DESs components, the system *plant* (Cassandras and Lafortune 2009; Ramadge and Wonham 1989) (also known as *open-loop plant*) can be exposed by the composition

$$G = \|_{j \in J} G_j.$$

This model maps how the system evolves in parallel, without restrictions. In practice, however, the plant is expected to be coordinated, so that components can interact with each other as intended. For this purpose, an additional structure called *specification* can be composed with the plant. A specification can be seen as a prohibitive action that restricts the system behavior, adjusting it to cope with some expected standard.

Similarly to the plant, the specification $E$, for

$$E = \|_{i \in I} E_i \,,$$

with $i \in I = \{1, \cdots, n\}$, can also be expressed by a set of FSMs and composed automatically to $G$. The result is a composition

$$K = G \| E$$

that expresses the behavior of the system under control exactly as projected by the engineer, which may require further software verification. Another option is to process *supervisor synthesis* (Ramadge and Wonham 1989) to extract from $K$ its largest sub-model that includes quality guarantees by construction, without the need for further checking, which is discussed as follows.

## 2.2 CONTROL SYNTHESIS

In general, a controller for a DES is expected to have quality and performance properties before it is implemented. “Performance", here, refers to the ability for the controller to allow the entire system behavior, except prohibitive actions. And by “quality" we mean mainly that a controller should recognize the nature of events enablement in the system, which may be uncontrollable, and does not ever lead the system under control to block.

By processing *supervisor synthesis* (Ramadge and Wonham 1989), those three properties are ensured by construction, without further checking, and they can be defined as follows.

**Definition 1.** *An* FSM $G = \langle \Sigma_G, Q_G, q_G^\circ, Q_G^\omega, \rightarrow_G \rangle$ *is nonblocking if, for all* $s \in [G]$, $G \xrightarrow{s} q$ *implies that* $\exists\, t \in [G]$ *such that* $q \xrightarrow{t} q'$, *for some* $q' \in Q_G^\omega$.

Differently, controllability deals with the impossibility of disabling some events in the system. For a DES $G$, let $\Sigma_G$ be partitioned into $\Sigma_G = \Sigma_{\mathrm{c}}[G] \cup \Sigma_{\mathrm{u}}[G]$, where $\Sigma_{\mathrm{c}}$ [$G$ ] is the set of *controllable* and $\Sigma_{\mathrm{u}}$ [$G$ ] is the set of *uncontrollable* events in $G$. Then, *controllability* can be defined as follows.

**Definition 2.** *Let* $G = \langle \Sigma_G, Q_G, q_G^\circ, Q_G^\omega, \rightarrow_G \rangle$ *and* $K = \langle \Sigma_K, Q_K, q_K^\circ, Q_K^\omega, \rightarrow_K \rangle$ *be two* FSMs. $K$ *is said to be controllable with respect to* $G$ *if, for all* $s \in [G]$ *and all* $\mu \in \Sigma_{\mathrm{u}}[G]$, *if*

$$K \xrightarrow{s} q_K \qquad \textit{and} \qquad G \xrightarrow{s} q_G \xrightarrow{\mu} q'_G$$

*then there exists* $q'_K \in Q_K$, *such that*

$$K \xrightarrow{s} q_K \xrightarrow{\mu} q'_K.$$

That is, if after any string $s$, an uncontrollable event $\mu$ is possible in the plant $G$, then it is also possible in $K$.

The operation that computes from $K$ its largest sub-model that respects the impossibility of disabling events in $\Sigma_{\mathrm{u}}[G]$ is called *control synthesis*, and it is the kernel of the *Supervisory Control Theory* (SCT) (Ramadge and Wonham 1989) and its several extensions. The synthesis result is also called *supremal* and denoted $\sup\mathcal{C}(K,G)$. In practice, $\sup\mathcal{C}(K,G)$ models the most permissive behavior possible to be imposed to $G$, while complying with controllability and satisfying the specification $K$. If $\sup\mathcal{C}(K,G)$ is also nonblocking, it is called an *optimal* controller, and will play an essential role in the results to be derived later in Section 5, involving RL.

## 2.3 COMPLEXITY ANALYSIS

In SCT, complexity issues emerge at three distinct steps: modeling, synthesis, and implementation. During the modeling phase, complexity is difficult to be measured, as it is associated with unsystematic engineering tasks. So, in the following, we assume it to be well defined from the engineering point of view.

After modeling, complexity can be well measured in terms of computational cost to process plant and specification models, given respectively as FSMs

$$G = \|_{j=1}^{m} G_j \qquad \text{and} \qquad E = \|_{i=1}^{n} E_i.$$

If $G_j$ has in the worst case $p$ states, then the composed plant $G$ has $p^m$ states. Similarly, if $E_i$ has $q$ states, then the specification $E$ has $q^n$ states. Therefore, the computation of the supremal sub-model $\mathrm{sup}\mathcal{C}(K,G)$ grows on the order of $O[(p^m.q^n)^2]$ in time.

In summary, any computation over $G$ and $E$ can be processed polynomially in their number of states. However, $G$ and $E$ may grow exponentially in the number of models that they compose. While in practice this should limit the $\mathrm{SCT}$ to be applied over large $\mathrm{DESs}$, the computational cost can be mitigated by using modular frameworks (Teixeira *et al.* 2018) in the literature.

As for the implementation phase, complexity can be assessed by the number of states (indicating memory usage) and transitions (indicating communication channels) of the synthesized controller. In this work, we do not go through the implementation step, but its analysis is important as the $\mathrm{RL}$ training is processed over the state-space of the $\mathrm{SCT}$ controller. By definition, $\mathrm{sup}\mathcal{C}(K,G)$, input to the $\mathrm{RL}$, is always a sub-model of $K = G \parallel E$, so that the state-space used for $\mathrm{RL}$ training is always expected to be $\leq$ than the state-space of $K$.

# 3 OVERVIEW ON REINFORCEMENT LEARNING

*Reinforcement Learning* (RL) is a computational paradigm in which an agent aims to increase its performance based on rewards received upon interacting through an environment, learning a policy or a sequence of actions (Kaelbling *et al.* 1996; Sutton and Barto 2018). At each time step $t$, this agent observes its current state $s_t \in S$ and chooses an action $a_t \in A$ to be executed following a policy $\pi$, which will take it to state $s_{t+1}$. Each state-action pair ($s_t$,$a_t$) returns a reinforcement signal, $R(s_t,a_t) \rightarrow \mathbb{R}$, given by the environment to the agent in return for its actions.

Formally, the RL method can be exposed as a *Markov Decision Process* (MDP) (Puterman 1994), defined as a quadruple $M = \langle S,A,T,R,\gamma \rangle$, where:

- $S = \{s_1, \cdots ,s_n\}$ is the finite set of states of the environment;
- $A = \{a_1, \cdots ,a_n\}$ is the finite set of actions that the agent can perform;
- $T : S \times A \rightarrow \Pi(S)$ is a state transition function where: $\Pi(S)$ is a probability distribution over the set of states $S$; and $T(s_{t+1}, s_t|a_t)$ is the probability of a transition from $s_t$ to $s_{t+1}$ occur with an action $a_t$;
- $R : S \times A \rightarrow \mathbb{R}$ is a function that defines the reward received by the agent for selecting an action $a_t$ in a state $s_t$ at instant $t$;
- $\gamma \in [0,1]$ is a discount factor that determines how far in the future an RL agent looks for rewards in comparison with its immediate future.

A single episode $e_{MDP}$ can be described as a sequence of states, actions, and rewards (i.e. the experience) acquired by an agent from the initial state of the environment to a terminal state, and can be defined as:

$$e_{MDP} = s_0,a_0,r_0,s_1,a_1,r_1, \cdots ,s_{n-1},a_{n-1},r_{n-1},sn\,, \tag{1}$$

where $s_i$ represent the $i^{th}$ state, $a_i$ the $i^{th}$ action, and $r_i$ the $i^{th}$ reward. Then, the amount of future reward, at any time point $t$, is given by:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n\,. \tag{2}$$

Note that the discount factor $\gamma \in [0,1]$ expresses how seriously the agent takes future rewards into account. Values close to $0$ represent a short-sighted strategy, as higher-order terms for rewards in the distant future become negligible. If the environment is deterministic, $\gamma$ can be set to $1$ as the same actions always result in the same reward (Lapan 2018).

An $\mathrm{RL}$ agent implements, at each time step, a function that maps an action for every state in the environment. This function is called the agent's policy and is denoted by $\pi$. Although there are a large number of possible policies to be followed by the agent, we can say that the optimal policy $\pi^*$ represents the actions to perform in each state which aims for the best possible reward that can be acquired by the agent and is better or equal to all other policies $\pi$ (Sutton and Barto 2018).

In the literature, there is a wide variety of $\mathrm{RL}$ algorithms that aim on learning the optimal policy $\pi*$, such as Q-Learning (Watkins and Dayan 1992), H-Learning (Tadepalli and Ok 1994), Dyna (Sutton 1991), SARSA (Sutton and Barto 2018), and Deep Q (Mnih *et al.* 2015). In this work, without loosing generality, we test our approach using SARSA and n-step SARSA, assessing the practical evolution of n-step SARSA compared with its ordinary version. Details are provided in Section 5. Also, it is important to note that the conversion of $\mathrm{FSMs}$ to $\mathrm{MDPs}$ results in a Gym Wrapper (Brockman *et al.* 2016), over which it is possible to apply any $\mathrm{RL}$ algorithm for optimization. This is discussed in details in (Zielinski *et al.* 2020).

## 3.1 SARSA

The algorithm *State–Action–Reward–State–Action* (SARSA) (Sutton and Barto 2018) has attracted attention for its simplicity and effectiveness. The basic idea of SARSA is to estimate a Q function $Q_{\pi(s_t,a_t)}$ for the current behavior policy $\pi$ and for all states $s_t$ and actions $a_t$. This function is implemented in the form of a map $Q$ that assigns to $S \times A$ a value $V$ that represents the return for performing an action $a_t$ in a state $s_t$. Since the agent's state and action space do not omit relevant information, once the optimal function is learned, the agent knows which action results in a better future reward, which is true for all states.

As SARSA uses a on-policy method to update the action values, that is, it chooses the actions at each time step upon looking at $\pi$, the Q-function update can be formulated as:

$$Q^{\pi}_{t+1}(s_t,a_t) \doteq Q^{\pi}_{t}(s_t,a_t) + \alpha\Big(r + \gamma Q^{\pi}_{t}(s_{t+1},a_{t+1}) - Q^{\pi}_{t}(s_t,a_t)\Big), \tag{3}$$

where $Q_t^{\pi}(s_t,a_t)$ is the execution of an action $a_t$ in a state $s_t$, both taken at a time step $t$ and following a policy $\pi$; $\alpha$ is the learning rate of the training; and $Q_t^{\pi}(s_{t+1},a_{t+1})$ is the quality value of executing action $a_{t+1}$ in state $s_{t+1}$ following a policy $\pi$. Equation 3 aims to find a local maximum value, since it is a gradient ascent based approach. The Q function represents the discounted expected reward for taking an action $a$ when visiting a state $s$, and following an optimal policy since then. Algorithm 1 shows the procedural form of SARSA.

**Algorithm 1 – SARSA procedure**

**Ensure:** $S$, $A$, $Q$ as input
1: For each $s_t$,$a_t$, set $Q(s_t,a_t) = 0$
2: Select action $a_t$ under policy $\pi$
3: **while** Stop Condition = False **do**
4: Executes $a_t$
5: Receive immediate reward $r(s_t,a_t)$
6: Observe new state $s_{t+1}$
7: Select action $a_{t+1}$ under policy $\pi$
8: Update $Q(s_t,a_t)$ according to Equation 3
9: $a \leftarrow a_{t+1}$
10: $s \leftarrow s_{t+1}$
11: **end while**
12: Return $Q$

The stop condition at line 3 can be defined, for example, by executing a specific number $n$ of steps in a single episode, reaching a terminal state. Once all state-action pairs are visited a finite number of times, termination is guaranteed and the method generates $Q$ and converges to a value $Q^*$ in a finite amount of time.

However, selecting an action under a policy in many cases means choosing the action with the highest Q-value for a given state $s_i$. Suppose, for example, that the agent is in the initial state ($s_0$) of the $\mathrm{MDP}$ and takes an action $a_1$, which gives a good reward. Thus, upon updating the Q-table, there is a value higher than 0 in $Q(s_0,a_1)$. The agent is unaware that the action $a_0$ in $s_0$ returns a better reward than $a_1$. Yet, it always chooses $a_1$ because it is the maximum value in the Q-table. Improvements can be obtained by using *epsilon-greedy* as a policy $\pi$ of line 7 in Algorithm 1, forcing the agent to explore all state-action pairs in table Q. The policy epsilon-greedy is detailed in Algorithm 2.

**Algorithm 2 – Epsilon greedy policy**

**Ensure:** $s_t$, $Q$,$\epsilon$ as input
1: Generate random number $x$ between 0 and 1
2: **if** $x > \epsilon$ **then**
3: Choose action with highest Q-value in $Q(s_t)$
4: **else**
5: Choose random action
6: **end if**

The objective here is to switch between exploration (choose random actions) and exploitation (choose the best action) according to a value $\epsilon$ between 0 and 1, which defines the probability of choosing a random action to explore the environment. Upon receiving as input the current state $s_t$, the Q-table $Q$, and $\epsilon$, the algorithm generates a random value between 0 and 1. If it is higher than $\epsilon$, then the action with the highest Q-value is chosen in state $s_t$ according to $Q$. Otherwise ($\leq\ \epsilon$) a random action is taken.

Despite useful, the effectiveness of SARSA is limited to simple environments with a reduced number of states and actions. This is because Equation 3 processes the action value $Q(s_t,a_t)$ by looking only one step forward in the environment, i.e., $Q(s_{t+1}.a_{t+1})$. Although this allows knowing the Q value for the next step, many steps ahead change with the learning rate $\alpha$, and may vanish over time. For example, let A and B be two paths in an environment, such that A returns reward 10 after 4 steps, while B returns an immediate reward of 1. In SARSA, as the profit received after 4 steps decreases with time, it chooses path B. Improvements can be obtained from a variant of SARSA, called *n-step SARSA* (Sutton and Barto 2018), which is the base for the results to be presented in this document.

## 3.2 N-STEP SARSA

Multi-step (or n-step) tabular methods can be seen as the result of processing many step of different lengths (Asis *et al.* 2017). While the SARSA algorithm presented in the previous section considers 1-step further, its n-step version considers n steps further, and their comparison is better shown in Fig. 2.

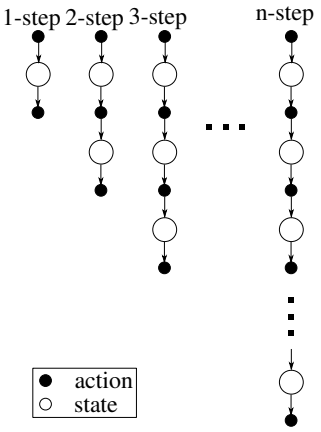



**Figure 2 – Backup diagrams for n-step SARSA algorithms**

The n-step returns can be defined in terms of estimated action values:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(s_{t+n}, a_{t+n}) \\ n \geq 1, 0 \leq t < T - n \cdot \tag{4}$$

Then, the update function in Equation 3 can be replaced to:

$$Q^{\pi}_{t+n}(s_t, a_t) \doteq Q^{\pi}_{t+n-1}(s_t, a_t) + \alpha \Big[ G_{t:t+n} - Q^{\pi}_{t+n-1}(s_t, a_t) \Big] \\ 0 \leq t < T \cdot \tag{5}$$

Algorithm 3 implements the n-step method. The SARSA variables are initialized on lines 1, 2, and 3. The difference this time is that an integer $n$ identifies how many steps further are to be considered. Also, three lists are defined to store states, actions, and rewards of the path to be processed. The lowercase $t$ (line 10) is the current timestamp of the agent, starting from 0 and reaching the end of the episode; the uppercase $T$ (line 9) is the number of steps that the current episode has, and its value is assigned on line 16; $\tau$ is the timestamp of the $Q$ value that is being updated.

When entering the loop (line 11), an action $A_t$ is taken following the policy $\pi$. The tuple ($A_t$, $R_{t+1}$, $S_{t+1}$) is stored until the next state is not terminal. When it is terminal, the algorithm stops. On line 23, $\tau$ is used to sum the discounted reward $G$ for $n$ steps forward. If the $n^{th}$ step is the last timestamp of the episode (line 24), then it picks up the value $Q(S_{\tau+n}, A\tau + n)$ as the last term to the sum. Finally, line 27 updates the Q-value using the reward $G$.

In summary, the n-step SARSA brings the advantage of looking multiple steps further. Its disadvantage is an increasing in the computational cost and memory to record information about previous timestamps. Yet, this seems to be a reasonable price to be paid for escaping from limited single time-step methods (Sutton and Barto 2018). Examples using both 1 and n step SARSA are presented in (Zielinski *et al.* 2021).

## 3.3 COMPLEXITY ANALYSIS

The $\mathrm{RL}$ input is an $\mathrm{FSM}$ that models a $\mathrm{SCT}$ controller computed as in section 2. As it includes only the part of the system behavior that survives under control, its state-space is, in general, much smaller in comparison with the entire system, which favors the $\mathrm{RL}$ processing.

In case SARSA is processed, its stop condition is given by the number of episodes $E$, i.e., by the loop content on line 4 running $E$ times. Notice that, on line 8, the action $a'$ is selected

**Algorithm 3 – N-step SARSA**

1: Initialize $Q(s,a)$
2: Initialize policy $\pi$ with respect to $Q$ (normally $\epsilon$-greedy)
3: Parameters: step size $\alpha$, $\gamma$ and $\epsilon$ between 0 and 1
4: A positive integer n
5: Initialize list of States, Actions and Rewards
6: **while** Stop Condition == False **do**
7: Initialize and store $S_0 \neq Terminal$
8: Select and store an Action $A_0$ following $\pi$
9: $T \leftarrow \infty$
10: $t \leftarrow 0$
11: **while** $\tau \neq T - 1$ **do**
12: **if** $t < T$ **then**
13: Take action $A_t$
14: Observe and store $R_{t+1}$ and $S_{t+1}$
15: **if** $S_{t+1}$ is terminal **then**
16: $T \leftarrow t + 1$
17: **else**
18: Select and store $A_{t+1}$ following $\pi$
19: **end if**
20: **end if**
21: $\tau \leftarrow t - n + 1$
22: **if** $\tau \geq 0$ **then**
23: $G \leftarrow \Sigma_{i=\tau+1}^{min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
24: **if** $\tau + n < T$ **then**
25: $G \leftarrow G + \gamma^n Q(S_{\tau+n},A_{\tau+n})$
26: **end if**
27: $Q(S_\tau,A_\tau) \leftarrow Q(S_\tau,A_\tau) + \alpha\Big[G - Q(S_\tau,A_\tau)\Big]$
28: **end if**
29: **end while**
30: **end while**

under the policy $\pi$, which is the $\epsilon$-greedy policy. This means that the $\epsilon$-greedy policy itself is executed $E$ times, and its code is therefore also used to determinate SARSA complexity.

Notice (line 4) that the $\epsilon$-greedy chooses the action with the highest Q-value from the table, which means that it has to iterate through the entire table to find the highest value from all possible actions in that state. Hence, the number of iterations of the $\epsilon$-greedy policy is limited by the number of actions of the $\mathrm{MDP}$ ($A$). Therefore, the complexity of the SARSA procedure should be $O(A \cdot E)$, but it is clear that $E$ is far bigger than $A$, so that the asymptotic complexity of the SARSA procedure can be summarized as $O(E)$.

Differently, the n-step SARSA is more complex than its 1-step version. The loop on line 11 is executed $T + n$ times: $T$ times due to the condition on line 12, that is executed until the value of $t$ reaches $T$; and $n$ times because of the condition on line 23, where $n$ is the number of steps, i.e., the hyperparameter for the n-step. Asymptotically, the execution of the $\epsilon$-greedy policy into this loop on lines 13 and 19 can be ignored. As the stop criteria is also the number of episodes ($E$), this should multiply the argument $T + n$, i.e., leading to $O(E \cdot (T + n))$.

# 4 METHODOLOGICAL PROCEDURES

This section shows how a SCT controller can be translated to MDP (Section 4.1); how event controllability can be extended to RL treatment (Section 4.2); and how the entire approach can be applied in practice.

## 4.1 RELATIONSHIP BETWEEN AN FSM AND AN MDP

In order to train a DES with RL, the structure of an MDP has to be combined with FSMs, as they are slightly different. To fairly compare them, let us start with the simplest Markov family representation: the *Markov Chain* (MC). An MC is a tuple $(Q, T)$, where $Q$ is the set of states and $T$ is the transition matrix, which informs the probabilities for moving from a state to another (Ashraf 2018).

This evidences the main difference between FSMs and MCs: FSMs are fully deterministic, i.e., if the current state and the event to be triggered are known, the next state can be anticipated. Differently, MCs are stochastic, i.e., there are randomness about the outcome from a given state, provoked by the transition probability matrix.

Whenever convenient, MCs can be extended with a set of actions, so that the agent no longer observes passively to states and transitions, but also chooses actions to be taken at every time-step (Lapan 2018). This alters probabilities of target states, which is a useful ability. By enriching an MC with this feature, it becomes partially deterministic, i.e., the outcome from a given state is no longer fully random. A particular case appears when, for all states of an MC, there is 100% probability that an action leads to a next state. This defines the *Deterministic Markov Chains* (DMCs), which are finally quite similar to FSMs structures and represent the base for our conversion method that follows.

In order to convert DMCs back into MDPs, one has to add a value to the set of transitions, to represent the reward matrix, and a discount factor, i.e., a number between 0 and 1 (Lapan 2018) (see definition in Section 3. As we also want to allow some stochasticity in the MDP behavior, we consider complementing the matrix $T$ with transition probabilities, transforming the DMC into a stochastic MDP. A methodology that summarizes how FSMs are translated into MDPs is presented as follows:

(i) FSM *to* DMC: translate the FSM $\langle \Sigma, Q, q^{\circ}, Q^{\omega}, \rightarrow \rangle$ into the DMC $(S, A, T)$, where $S$ is

the state set, $A$ is the action set and $T$ is the deterministic transitions of the $\mathrm{DMC}$. Here, we assume that $S \equiv Q$, $A \equiv \Sigma$ and $T \equiv \rightarrow$.

(ii) $\mathrm{DMC}$ *to deterministic* $\mathrm{MDP}$: transform the $\mathrm{DMC}$ into an $\mathrm{MDP}$ by adding a reward function $R$. This is done by allocating rewards for every $a \in A$. Also, it is necessary to add a discount factor $\gamma$ to the $\mathrm{DMC}$. As a result, the tuple $(S,A,T)$, of the previous step, is transformed into the $\mathrm{MDP}$ tuple $\langle S,A,T,R,\gamma \rangle$.

(iii) *Deterministic* $\mathrm{MDP}$ *to stochastic* $\mathrm{MDP}$: add transition probabilities to the transition matrix $T$.

The result of the translation process is an $\mathrm{MDP}$ environment over which it is possible to apply $\mathrm{RL}$ algorithms in order to add flexibility to a $\mathrm{DES}$. Yet, some controllability issues remain to be discussed in order to semantically harmonize the integration of $\mathrm{MDP}$ and the controller resulting from the $\mathrm{SCT}$ step.

## 4.2 CONTROLLABILITY CONSIDERATIONS

From the partitioned set of events $\Sigma = \Sigma_c \cup \Sigma_u$, the $\mathrm{SCT}$ approach in Section 2.2 calculates controllers that respect the impossibility of disabling events in $\Sigma_u$. In this section, we enrich a $\mathrm{RL}$ agent with the same notion of controllability, which allows us to preserve the nature of event occurrences, an important feature for the consistency of the flexible control system to be derived.

Following the translation step-guide in the previous section, the $\mathrm{FSM}$ set of events becomes the $\mathrm{MDP}$ set of actions, which is equivalently split into $A = A_c \cup A_u$, also meaning controllable and uncontrollable actions, respectively. Their type is now considered by the $\mathrm{RL}$ agent when measuring the quality value $Q$. As the $\mathrm{RL}$ agent cannot decide to choose among uncontrollable actions (as they are free to occur), it only chooses actions: (i) eligible by the $\mathrm{SCT}$ controller; and (ii) belonging to the set of controllable actions.

This reflects on the $\epsilon$-greedy policy used by both algorithms 1 and 3. The following remarks help to understand the controllability issues that need to be considered by the $\mathrm{RL}$ action policy.

**Remark 1.** *In case of choosing exploitation, pick up the action with the highest Q-value in that state, as long as the action is controllable.*

**Remark 2.** *Triggering or not uncontrollable actions, based on probabilities or exploration, is a matter only during the training phase. Upon training, the agent chooses only controllable actions with the highest Q-value.*

**Remark 3.** *Controllable actions are not triggered by probabilities, as the control agent decides whether or not to take them. Thus, the transition probabilities are only valid for uncontrollable actions. In this way, the* RL *agent learns an optimal policy on training, and chooses the action to take in a given state based on the best possible reward, as long as the selected action is controllable.*

Now, we are in position to complement the $\epsilon$-greedy policy in Algorithm 2 with controllability, as shows the Algorithm 4. This approach is called *Controllable $\epsilon$-greedy policy*, and it aims to harmonize controllability issues when integrating SCT and MDP, allowing for an agent to search only through the set of possible actions $A$, in a given state $i$ ($A^i$).

**Algorithm 4 – Controllable Epsilon greedy policy**

1: Receives $s_i$, $Q$, $\epsilon$, $A^i$, $P$, $A_u$, $A_c$ as input
2: $A_u^i = A^i \cap A_u$
3: $A_c^i = A^i \cap A_c$
4: Create empty list $L$
5: **if** $A_u^i \neq \emptyset$ **then**
6: Create empty list $\Delta$
7: **while** Iterate over $A_u^i$ such that $A_u^i[j] = a_{uj}^i$ **do**
8: **if** $P(a_{uj}^i) > 0$ **then**
9: $\Delta \leftarrow a_{uj}^i$
10: Remove $a_{uj}^i$ from $A^i$
11: **else**
12: Stores $a^i uj$ in $L$
13: **end if**
14: **end while**
15: **end if**
16: **if** $\Delta \neq \emptyset$ **then**
17: Select action $a_p$ from $\Delta$ based on probabilities of $P$
18: Stores $a_p$ in $L$
19: **end if**
20: **if** $A_c^i \neq \emptyset$ **then**
21: Generate random number $x$ between 0 and 1
22: **if** $x > \epsilon$ **then**
23: Stores action of $A_c^i$ with highest Q-value $Q(s_i)$ in $L$
24: **else**
25: Stores random action of $A_c^i$ in $L$
26: **end if**
27: **end if**
28: Select random action in $L$

On line 2, the algorithm stores all possible uncontrollable actions of $A^i$ in a list of uncontrollable transitions called $A_u^i$, i.e., $A_u^i = A^i \cap A_u$. Similar idea applies for controllable

transitions on line 3, i.e., $A_c^i = A^i \cap A_c$. If $A_u^i \neq \emptyset$, then an empty list $L$ of liable actions to be taken is created on line 4. On line 6, another empty list $\Delta$ is created to store all probabilistic actions. On line 7, the algorithm iterates through $A_u^i$, in which $A_u^i[j] = a_{uj}^i$, and verifies if $P[a_{uj}^i] > 0$. If so (line 9), the probabilistic action $a_{uj}^i$ is stored in $\Delta$. If the action is not probabilistic, the uncontrollable action is stored in $L$.

On line 16, if the list $\Delta$ is not empty, then (line 17) one action is selected from the list, based on the triggering probabilities of each action, and stored in $L$. If $A_c^i$ is not empty (line 20), a random number between 0 and 1 is generated. If this number is less than $\epsilon$, the agent chooses the action with the highest Q-value in $A_c^i$ to append in $L$. Otherwise it chooses a random action. Finally, a random action in $L$ is chosen on line 28. At this point, the list has one controllable action that was chosen by the epsilon-greedy method, one uncontrollable action with triggering probability, and all the uncontrollable actions without probabilities.

The diagram in Fig. 3 summarizes a step-guide for using RL in DESs.

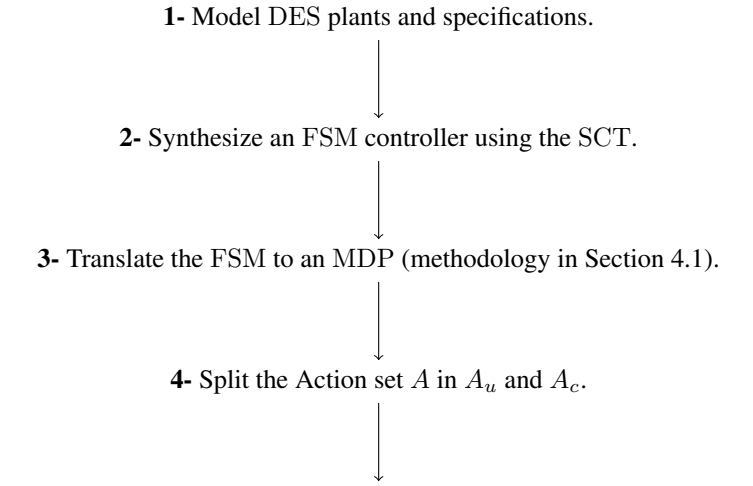



**Figure 3 – General methodology to apply RL on DESs.**

# 5 CASE STUDY ON AN INDUSTRY APPLICATION

Consider the partial version of a flexible automotive manufacturing system shown in Fig. 4, where multiple robots perform distinct operations, over distinct cars, that pass through the line. In automotive manufacturing, different car models are expected to be assembled using the same plant, but each model has its own set of components. Therefore, cars are assembled differently, may require different actuation setups, and may follow particular workflows. For these reasons, controlling systems like these is in general associated with complex engineering tasks and sensitive decision making.

A more viable alternative is using modular FSMs to approach engineering tasks, over which control synthesis can be processed automatically. In this way, programming is never actually faced by the engineer, while the control logic still coordinates the manufacturing tasks with the required precision. We further show how RL can optimize decision making and complement the logic control with artificial perception.

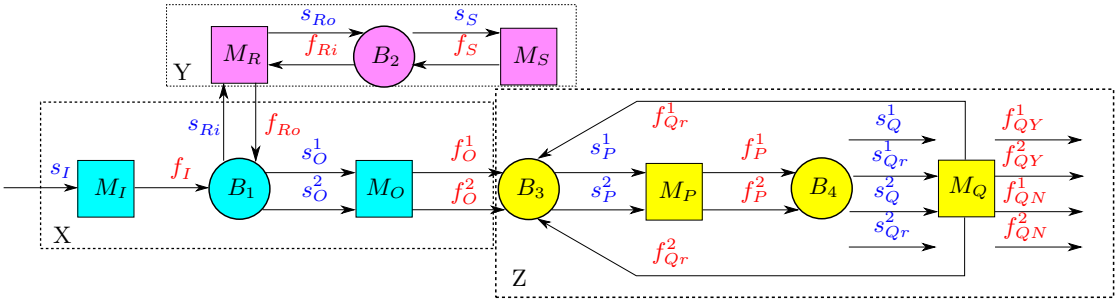



**Figure 4 – Partial version of a flexible automotive manufacturing system. Squares represent machines; circles model buffers; arrows represent flow. Blue labels mean controllable, while red labels mean uncontrollable events. Block X is responsible for receive and redirect cars to other blocks, block Y adds an optional to the car and block Z sends a car to rework or out of the production line.**

The example is split into 3 blocks, X, Y, and Z. Each block is responsible for a specific task to complement the production. It is assumed that:

(i) Cars may enter or not block Y, which leads to a different manufacturing;

(ii) An unlimited number rework cycles is allowed for each car. The decision on whether leaving or not the line is not in charge of the controller, but of an intelligent agent that calculates whether or not a car is worth to be reworked, based on costs and profit rates;

Cars enter the system through the block X. In $B_1$, a decision is required: a car may be selected for customization (e.g., painting, air conditioning, or safety devices) in block Y (event

$s_{Ri}$) before moving towards block Z; or it may follow directly to block Z without customization. Two events of machine $M_O$, $s_O^1$ and $s_O^2$, detect the type (1 or 2) of car, respectively with or without customization. This leads us to the first question we try to answer.

**Question 1.** *Entering or not the block $Y$ is a decision that does not depend on the process control, but on what is better for the company production and expected profit, so that the controller has only to allow each path. Thus, how this choice could be provided to the automatic controller?*

The two types of cars then enter block Z, where the remaining manufacturing is conducted. Upon reaching the station $B_4$, two outcomes are possible in the test machine $M_Q$: (i) the car leaves the system, independently on whether or not it is approved by the quality test (event labels $f_{QY}^i$ (YES) and $f_{QN}^i$ (NO), for types $i = 1,2$); or (ii) the car is reworked in Z in case of a rejection in the quality test (event labels $f_{Qr}^i$, for the types $i = 1,2$). This leads us to the second question that we aim to answer.

**Question 2.** *When a car fails the quality test (machine $M_Q$), the control system has to opt by: (i) a rework, trying to complement its customization and hopefully increase its chances to be approved in the next test; or (ii) removing it from the manufacturing line. This is also a decision that does not depend on the process control, but on the company's profit and sales expectations. The controller is only expected to enable both possibilities for an external agent to choose. How this decision could be provided to the automatic controller?*

Questions 1 and 2 are aligned with the intelligent solution proposed in this study to be integrated with the logic solution resulting from the SCT, and their application and integration are detailed better in the following.

## 5.1 PROBABILISTIC ASSUMPTIONS

It is reasonable to assume that the test unit (machine $M_Q$) has a probability associated with how likely each type of car is to pass the test. Cars type 2 are more likely to be approved, as they include less equipment to be tested in comparison with cars type 1. The inclusion of more devices in cars type 1 is, therefore, a decision that must balance the value added to the car, in return for more chances for it to fail the quality test. Our approach captures these decisions using 2 parameters: (i) the probability for approval after the quality test, which is associated with the

transition matrix $T$; and (ii) the reward structure $R$ that reflects the profit upon manufacturing a car.

In the next sections, we use the Fig. 3 to present the modeling, control, and optimization steps for the example in Fig. 4. Our solution is expected to maximize profit, reduce costs and waste (time and row material), and provide environment simulation to anticipate manufacturing scenarios.

## 5.2 MODELING AND SYNTHESIS

This section addresses the steps 1 and 2 of the guide in Fig 3. Each component of the process in Fig 4 is modeled as an FSM. The event meaning is as in Table 1, and the result of the plant modeling step is shown in Fig. 5.

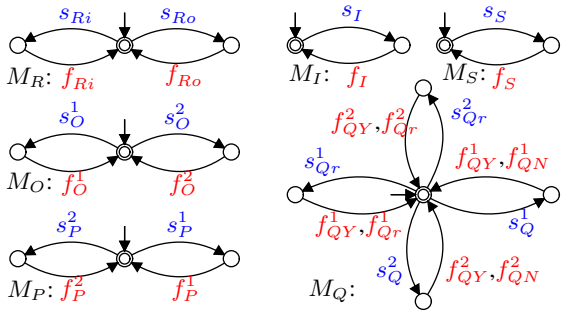



**Figure 5 – Plant of the automotive manufacturing system.**

| Controllable Events | | Uncontrollable Events | |
|---|---|---|---|
| Event | Description | Event | Description |
| $s_I$ | Machine MI starts working with a car | $f_I$ | Machine MI inserts a car in buffer B1 |
| $s_O^i$, $i = 1, 2$ | Machine MO starts working with a car of type $i$ | $f_O^i$, $i = 1, 2$ | Machine MO inserts car of type $i$ in buffer B3 |
| $s_P^i$, $i = 1, 2$ | Machine MP start working with a car of type $i$ | $f_P^i$, $i = 1, 2$ | Machine MP inserts car of type $i$ in buffer B4 |
| $s_Q^i$, $i = 1, 2$ | Test unit MQ picks car of type $i$, from buffer B4 to acceptance or rejection | $f_{QN}^i$, $i = 1, 2$ | Test unit MQ rejects car of type $i$, respectively |
| $s_{Qr}^i$, $i = 1, 2$ | Test unit MQ picks car of type $i$ from buffer B4 to acceptance or rework | $f_{Qr}^i$, $i = 1, 2$ | Test unit MQ sends car of type $i$ to rework |
| $s_{Ri}$ | Machine MR picks car from buffer B1 to insert some optionals in it | $f_{QY}^i$, $i = 1, 2$ | Test unit MQ approves car of type $i$ |
| $s_{RO}$ | Machine MR picks car from buffer B2 to reinsert the car in the main line | $f_{Ri}$ | Machine MR inserts car in buffer B2 |
| $s_S$ | Machine MS picks car from buffer B2 to insert some optionals in it | $f_{RO}$ | Machine MR inserts car in buffer B1 |
| | | $f_S$ | Machine MS inserts car in buffer B2 |

**Table 1 – Event description table**

Models $M_I$ and $M_S$ represent the homonym machine with 2 states that evolve with start and finish transitions. Models $M_O$, $M_P$, and $M_R$ design the machines that work in two modes, one for each type of car (1 and 2). Finally, FSM $M_Q$ expresses the two ways cars type 1 and 2 can either leave the system or go back to recycle. The composition $G = M_I \parallel M_O \parallel M_S \parallel$

$M_P \parallel M_Q$, with 540 states, corresponds to the unrestricted behavior of the plant. It is restricted by the set of specifications detailed in Table 2.

| Specification | Description |
|---|---|
| $B_i, i = 1, \cdots, 4$ | Avoid overflow and underflow in the buffer $B_i$ |
| $C_j, j = O,P,Q$ | Control the types of car handled by machine $M_j$ |
| $O_{Rewk}$ | Limit the number of rework cycles to be allowed for each car |
| $O_{Cust}$ | Decide whether or not adding customization for each car |

**Table 2 – Specifications to be considered in the example.**

Notice that designing specifications $B_i$ and $C_j$ is straightforward. In fact, they involves no memory tracing, only the disabling of certain events in order to impose the correct flow of cars through the production line. However, it has been shown in the literature (Malik and Teixeira 2021) that designing $O_{Rewk}$ and $O_{Cust}$ as FSMs can be quite tricky and lead to state space explosion, due to the following reasons:

(i) They both have a dynamic cost factor associated, which requires a more flexible approach to be expressed than FSMs.

(ii) Designing $O_{Rewk}$ as FSMs fixes a deterministic limit of cycles to be allowed. Whatever that limit is, it implies designing by hands the entire memory that precedes the control action, which increases exponentially with the number of cycles and is not practical for real cases (Teixeira *et al.* 2015; Cury *et al.* 2015; Teixeira *et al.* 2018; Malik and Teixeira 2021).

(iii) FSMs would impose to $O_{Rewk}$ and $O_{Cust}$ a static definition, which is not compatible with the purposes in this work.

In the following, the possibility of entering block Y, and recycling in block Z (specifications $O_{Rewk}$ and $O_{Cust}$), are kept free from the supervisory control synthesis. These decisions will emerge from the proposed RL software layer and provided automatically to the controller. The FSMs modeling $B_i$ and $C_j$ are represented in Fig. 6.

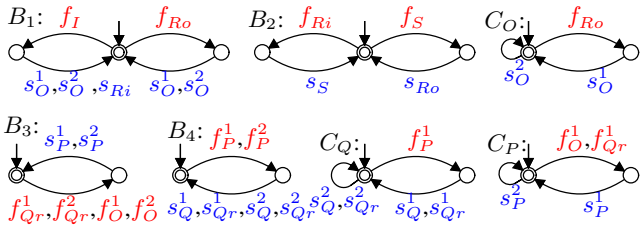



**Figure 6 – Restriction of overflow and underflow.**

In $B_i$, overflow is controlled by disabling events of arrival of cars in the buffer when it is full; underflow is controlled otherwise by disabling the start events of machines in the initial state, while the buffer is still empty. In addition to buffering control, it is also necessary to keep memory of the type of car to be processed, so that the actuation can be compatible. This is enforced by specifications $C_j$. For example, $C_O$ disables the operation of machine $M_O$ for cars type 1 (event $s_O^1$) in the initial state, and only enables it if the path though block Y (which culminates in the event $f_{Ro}$) is chosen. $C_O$ performs similarly for cars type 2. Restrictions $C_P$ and $C_Q$ work likewise and the result is the complete tracking of each type of car throughout the system.

The composition $E = B_1 \parallel B_2 \parallel B_3 \parallel B_4 \parallel C_O \parallel C_P \parallel C_Q$ has 81 states. When joined to the plant, it forms the expected behavior for the manufacturing system under control, modeled by $K = G \parallel E$ with 18630 states. Synthesis processing results a supervisor $\mathrm{sup}\mathcal{C}(K,G)$ with 1309 states and 4878 transitions.

We are now in position to translate the FSM that models $\mathrm{sup}\mathcal{C}(K,G)$ into an MDP. This will later serve as the environment to process the RL algorithm proposed to enforce specifications $O_{Rewk}$ and $O_{Cust}$, hopefully providing reasonable answers to questions 1 and 2. This corresponds to phases 3 and 4 of the step-guide in 3.

## 5.3 TRANSLATING FSMs INTO MDPs

From the methodology in Section 4.1, the first conversion step is to transform the FSM $\mathrm{sup}\mathcal{C}(K,G)$ into a DMC. Remark that, for an FSM $\langle \Sigma, Q, q^\circ, Q^\omega, \rightarrow \rangle$, and a DMC tuple $(S,A,T)$, equivalences follow for the set of states $S \equiv Q$, set of events and actions $A \equiv \Sigma$, and deterministic transition relations $T \equiv \rightarrow$. It is further necessary to add a reward function $R$ and a discount factor $\gamma$ in order to transform the DMC tuple $(S,A,T)$ into an MDP tuple $\langle S,A,T,R,\gamma \rangle$.

Function $R$ is created by allocating rewards for every $a \in A$. That is, for each event in the DES presented in Figure 4 (which coincides with the MDP actions after the FSM translation procedure), we allocate a reward to map its profit, in case it occurs in the system. For example, event $f_{QY}^1$ returns a high positive profit, while event $f_{QN}^1$ returns a high negative profit. It is also considered that machine actions have low negative profit (related with power consumption, wear, maintenance, etc.) and customized cars are more profitable than otherwise.

The final step towards deriving a RL environment is to add transition probabilities to

the deterministic MDP obtained so far. For the automotive manufacturing example, we consider only probabilities of approving a car after a quality test, and the other transitions are deterministic. We also assume that cars type 2 (not customized) have more or equal chances of approval than type 1, since more equipment implies more chances to violate the quality test.

For practical purposes, we tested the RL efficiency through the environment for 9 cases, each one varying the probability of approving cars type 1. Probability was started in 90% and reduced from 10% to 10% in the subsequent cases until reaching 10% in case 9. For the above setup, the rewards and probabilities adopted for every action are shown in Table 3. It was assumed a discount factor $\gamma$ of 0.9, as we pursue a low discount on long term rewards. Those parameters are only illustrative and they can be replaced without loosing generality, as shown in (Zielinski *et al.* 2021).

| **Controllable Actions** | | | **Uncontrollable Actions** | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -30000 | 10 - 90 |
| $s^1_O$ | -10 | Not considered | $f^2_{QN}$ | -20000 | 10 |
| $s^2_O$ | -10 | Not considered | $f_I$ | -10 | Not considered |
| $s^1_P$ | -10 | Not considered | $f^1_O$ | -10 | Not considered |
| $s^2_P$ | -10 | Not considered | $f^2_O$ | -10 | Not considered |
| $s^1_Q$ | -10 | Not considered | $f^1_P$ | -10 | Not considered |
| $s^1_{Qr}$ | -10 | Not considered | $f^2_P$ | -10 | Not considered |
| $s^2_Q$ | -10 | Not considered | $f_{Ri}$ | -10 | Not considered |
| $s^2_{Qr}$ | -10 | Not considered | $f_{Ro}$ | -10 | Not considered |
| $s_{Ri}$ | -10 | Not considered | $f_S$ | -10 | Not considered |
| $s_{Ro}$ | -10 | Not considered | $f^1_{QY}$ | 30000 | 90 - 10 |
| $s_S$ | -10 | Not considered | $f^2_{QY}$ | 20000 | 90 |
| | | | $f^1_{Qr}$ | -1000 | 10 - 90 |
| | | | $f^2_{Qr}$ | -1000 | 10 |

**Table 3 – Rewards and probabilities assumed for the example.**

It is assumed that actions leading machines to start working cause a minor -10 penalty, while producing an entire car returns a profit of 30000 for type 1 and 20000 for type 2. The same setup applies negatively, in case the car is rejected by the quality test. Also, starting a rework cycle implies a penalty of -1000. It was considered further that not customized cars (type 2) have constant probability of 90% to be approved, while customized cars (type 1) have variable probability from 90% to 10% through the 9 test cases.

We are now in position to assess how worth it is to customize a car, considering the combined impact of multiple variables and probabilities over the profit collected from the manufacturing line.

## 5.4 APPLYING $\mathrm{RL}$ OVER THE EXAMPLE

The algorithms SARSA and N-step SARSA (sections 3.1 and 3.2) are now applied over the example, considering the 9 test cases defined previously. This corresponds to the fifth and final phase of the step-guide in Fig. 3. We first check the performance of the algorithm SARSA, which is simple, but looks only one step forward in the environment; then we present improved results using the N-step SARSA method; and, finally, we generalize the results for larger test cases in order to show the effectiveness of the presented method.

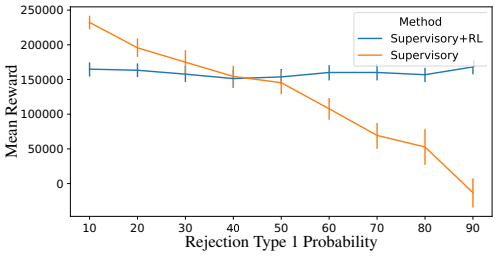
For training, the environment was tested over 10000 episodes, considering that each episode ends after producing 10 cars. It was also used the *controllable epsilon greedy* policy detailed in Algorithm 4, with $\epsilon$ value of 0.5. Then, each algorithm was tested over 20 episodes, and mean and standard deviation of rewards obtained with and without an $\mathrm{RL}$ agent helping the $\mathrm{SCT}$ controller, were measured. For each case, we collect and compare how many cars, of each type, was produced or discarded.

### 5.4.1 SARSA

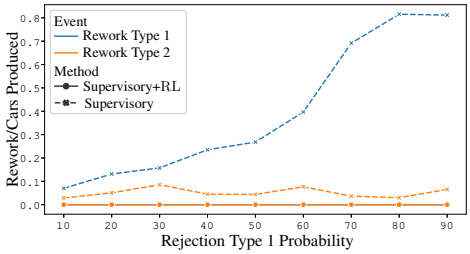
The performance (mean reward) resulting from SARSA over the manufacturing system example, in comparison with letting production to be controlled purely by the $\mathrm{SCT}$ method, is presented in Fig. 7(a). As the $\mathrm{SCT}$ does not choose whether or not to customize or rework, we set this possibilities in 50%. Figures 7(c) and 7(d) also show the number of occurrences of each event that leads a car to leave the production (either approved or not), both under the influence of $\mathrm{RL}$ and without it, respectively, while varying the fail probability.

Remark in Fig. 7(a) that the pure $\mathrm{SCT}$ obtains better results when running under low fail probabilities. This happens because the $\mathrm{SCT}$ controller chooses randomly whether to customize or not, while the $\mathrm{RL}$ method chooses mostly not to customize (see figures 7(c) and 7(d)).

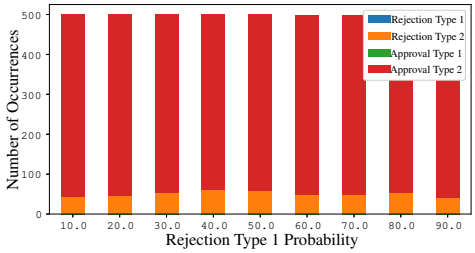
As the SARSA method updates the action values of the Q-table by looking only one step forward, it prioritizes short-term rewards. As the production of cars type 2 is shorter, it is



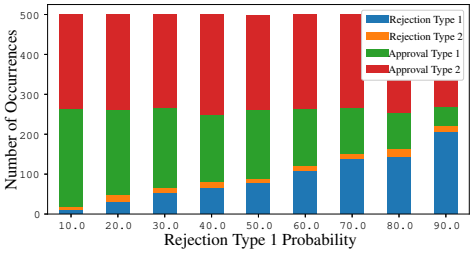

(a) Mean reward obtained.




(b) Number of rework cycles per car, for each event and method.




(c) Frequency of event occurrence with SARSA method.




(d) Frequency of event occurrence without the SARSA method.

**Figure 7 – Results of applying SARSA while varying the probability of rejecting cars type 1.**

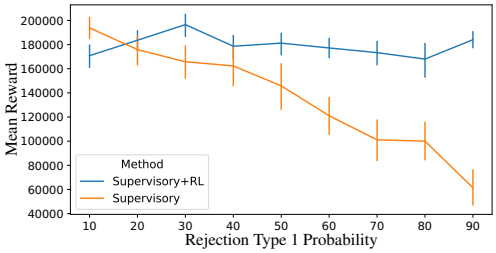
chosen mostly by the RL. This suggests that the 1-step SARSA is not an appropriate method to process RL over the example, even with mean reward slightly better than the SCT.

We can also notice in Fig. 7(b) that only SCT decisions allowed reworks of cars type 1. The RL decided not to produce this type, which may also not be an attractive decision. Generally, the RL method returned a reward of 160725, against 120892 from the SCT, meaning a performance improvement of approximately 32.9%.

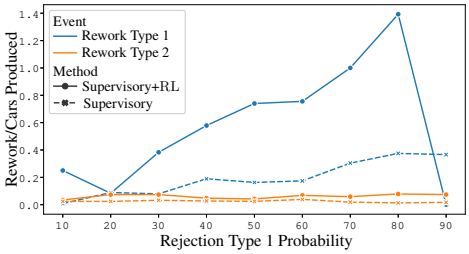
### 5.4.2 N-step SARSA

The second experiment exposes the performance (mean reward) resulting from the application of the N-step SARSA algorithm over the manufacturing system example. Its performance is again compared with the ordinary SCT.

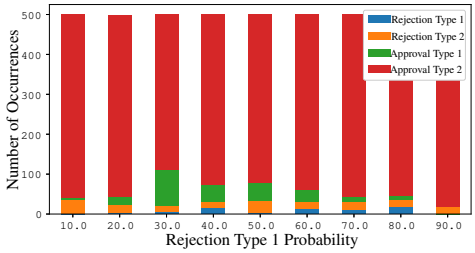
Differently from SARSA, the N-step SARSA performs better along all fail probabilities variations, as shown in Fig. 8(a). This happens because, while SARSA mostly chooses to produce cars without customization, and SCT chooses randomly, the N-step SARSA prioritizes customization when fail probability is low, as it feels that it may return higher profit. Over high fail probabilities, however, it prefers mostly not to risk producing cars type 1. In average, these intelligent decisions lead the algorithm to accumulate better rewards. The N-step SARSA resulted a mean reward of 179230, in comparison with 122483 from the SCT, representing an



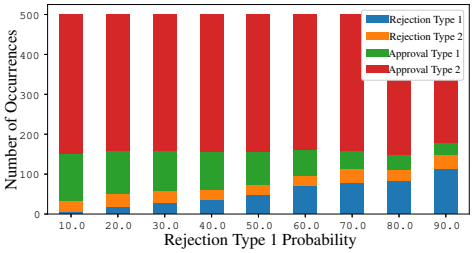

(a) Mean reward obtained.




(b) Number of rework cycles per car, for each event and method.




(c) Frequency of event occurrence with the N-step SARSA method.




(d) Frequency of event occurrence without the N-step SARSA method.

**Figure 8 – Results of applying the N-step SARSA while varying the probability of rejection of cars type 1.**

improvement of 46.3%.

Figure 8(b) shows the number of rework cycles applied per car. Under high fail probabilities, the SCT causes more rework cycles, since it reworks whenever the car is not approved, as expected. As for the intelligent method, the number of reworks also keep increasing with the fail probability.

Also see in figures 7(c), 8(d), 8(c), and 8(d) that there is a dominant event $f^2_{QY}$ (Approval type 2). This is because the environment supports parallel production, that is, it is possible to produce more than one car in parallel. When a car enters the block Y, its production takes more time than cars that do not enter that block. Suppose, for example, that a car reaches the station $B_2$, and another car reaches $B_1$. The car in $B_2$ will be blocked to proceed until the car in $B_1$ is moved to machine $M_O$. When it does, another car may enter the system and reach $B_1$, again blocking the car in $B_2$, i.e., the system has no *justice* property. The result may lead more cars to be produced, but they are prevented from customization. Here, we leave justice to be assigned in future results.

So far, we have tested the efficiency of the N-step SARSA, in comparison with its 1-step version, over the proposed example. As this method is expected to be independent on reward or probability settings, one remains to be shown its generalization capabilities, which is discussed in (Zielinski *et al.* 2021) though additional examples of small, moderate, and large sizes.

This can be assessed more generally over randomly-defined scenarios, which is approached next.

### 5.4.3 N-step SARSA for random scenarios

In this section, we consider simulating 100 different scenarios, with random values of reward and probabilities. Table 4 shows the parameters which were randomly varied. The main objective of this experiment is to verify if the intelligent method is able to optimize the profit, regardless of the adopted configuration. This is an important feature of our proposal, as it is expected to optimize DESs with variable configuration policies and/or objectives.

| **Controllable Actions** | | | **Uncontrollable Actions** | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10/-800 | Not considered | $f^1_{QN}$ | -20000/-40000 | 10-90 |
| $s^1_O$ | -10/-800 | Not considered | $f^2_{QN}$ | -20000/-40000 | 10-90 |
| $s^2_O$ | -10/-800 | Not considered | $f_I$ | -10/-800 | Not considered |
| $s^1_P$ | -10/-800 | Not considered | $f^1_O$ | -10/-800 | Not considered |
| $s^2_P$ | -10/-800 | Not considered | $f^2_O$ | -10/-800 | Not considered |
| $s^1_Q$ | -10/-800 | Not considered | $f^1_P$ | -10/-800 | Not considered |
| $s^1_{Qr}$ | -10/-800 | Not considered | $f^2_P$ | -10/-800 | Not considered |
| $s^2_Q$ | -10/-800 | Not considered | $f_{Ri}$ | -10/-800 | Not considered |
| $s^2_{Qr}$ | -10/-800 | Not considered | $f_{Ro}$ | -10/-800 | Not considered |
| $s_{Ri}$ | -10/-800 | Not considered | $f_S$ | -10/-800 | Not considered |
| $s_{Ro}$ | -10/-800 | Not considered | $f^1_{QY}$ | 20000/40000 | 10-90 |
| $s_S$ | -10/-800 | Not considered | $f^2_{QY}$ | 20000/40000 | 10-90 |
| | | | $f^1_{Qr}$ | -1000/-9000 | 10-90 |
| | | | $f^2_{Qr}$ | -1000/-9000 | 10-90 |

**Table 4 – Rewards and probabilities adopted for the test case. The parameters were varied randomly in order to simulate the algorithm through 100 different scenarios.**

Figure 9 shows that the intelligent method achieved better results in 99 of the 100 cases, in comparison with the SCT. In case 40, however, the SCT returned better profit. One hypothesis is that the created environment (with randomly generated rewards and probabilities) was harder for the RL agent to learn than the other scenarios, and it would require more training to adapt.

The mean reward of the SCT was of 24022, while the combined method returned 217736. This means approximately 9 times more profit, which reinforces the role played by the combination of RL and SCT, independently on probabilities setup, cost of production, vol-

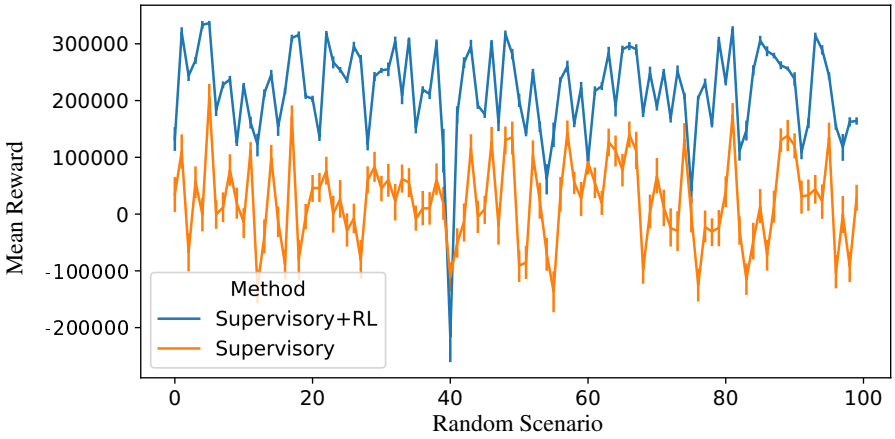



**Figure 9 – Results of the N-step SARSA while varying the environment parameters randomly.**

ume of manufactured products, etc. At the same time, SCT preserves safety properties that are essential along manufacturing.

# 6 FUTURE OPTIMIZATIONS

In this study, we consider that the probability distribution is uniform for all events. In future attempts, we aim to adapt these probabilities to other distributions, such as the exponential. For example, a machine may have 5% chances to crash for some episodes, while this may increase to 10% for others. This could be approached differently by modeling in future researches.

Another possible variation is to allow rework cycles to reduce the probability of failure, i.e., a car that reworks once reduces its fail probability by 10%, while twice reduces 20%, and so on. It is also possible to test more complex algorithms for the intelligent part, such as SARSA-$\lambda$ or $Q(\sigma,\lambda)$ (Sutton and Barto 2018). Other properties are also expected to be exploited, such as justice, in order to avoid that cars block when entering the customization zone. This may assign flexibility and efficiency for the $\mathrm{RL}$ agent to choose between sending cars to customization or not.

# 7 FINAL CONSIDERATIONS

This research is dedicated to match flexibility and safety in the control of DES. This is difficult to be ensured by hands, as engineers are in general limited to handle a few states and transitions, while real-scale systems are far more complex. Therefore, our contributions enrich the automatic control of DESs with simulation and intelligent capabilities that improve profit and are more tuned with the reality of modern flexible industries, while safety properties are still ensured. Besides presenting a method to handle the control of DESs intelligently, our results also point out differences between RL and DESs, and how they can be integrated by modeling. This creates an alternative to simulate, optimize, and control DES that are trained via RL.

The use of RL in DESs evidences practical appeal, considering that the concept of reward, used in RL algorithms, can reflect many practical tasks of an industrial systems, such as those related with cost, profit, and performance aspects in general. The RL agent can then learn how to perform those tasks and return valuable decision making skills, in order to aid the control engineering. In the presented example, the contribution brought by the association of RL and SCT allowed to obtain a mean reward (profit, in this case) approximately 9 times higher in comparison with the SCT acting alone to control the system. This leads us to believe that our approach has a wide and promising applicability in factory floors, under the price of a minor engineering effort: providing the input FSMs, a reward list for each triggered event in the system, and the probabilities for uncontrollable events.

By generalizing well for any system that can be modeled as FSM, our method supports any RL algorithm on the training step, without concerning about compromising the system functioning under control, as it explores an already safe, nonblocking, and controllable state space. As the approach is released as an open source tool, we believe that it can further support other studies involving RL and DESs, specially in industry 4.0-aware scenarios.

# REFERENCES


AHMED, Mobyen Uddin; BRICKMAN, Staffan; DENGG, Alexander; FASTH, Niklas; MIHAJLOVIĆ, Marko; NORMAN, Jacob. A machine learning approach to classify pedestrians' event based on imu and gps. **International Journal of Artificial Intelligence**, v. 17, p. 164–167, 2019.

ASHRAF, M. **Reinforcement Learning Demystified: Markov Decision Processes (Part 1)**. Towards Data Science, 2018. Available at: shorturl.at/tzJMS.

ASIS, K. de; HERNANDEZ-GARCIA, J. F.; HOLLAND, G. Z.; SUTTON, R. S. Multi-step reinforcement learning: A unifying algorithm. **CoRR**, abs/1703.01327, 2017. Available at: http://arxiv.org/abs/1703.01327.

BEN-ARI, Mordechai; MONDADA, Francesco. **Elements of Robotics**. *[S.l.: s.n.]*.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. **OpenAI Gym**. 2016.

CASSANDRAS, C. G; LAFORTUNE, S. **Introduction to discrete event systems**. *[S.l.]*: Springer Science & Business Media, 2009.

CURY, J. E. R.; QUEIROZ, M. H. de; BOUZON, G.; TEIXEIRA, M. Supervisory control of discrete event systems with distinguishers. **Automatica**, v. 56, p. 93 – 104, 2015.

GOMES, Leandro; MADEIRA, Alexandre; BARBOSA, Luís Soares. A semantics and a logic for fuzzy arden syntax. **Soft Computing**, v. 25, p. 6789–6805, 05 2021.

GROOVER, M.P. **Introduction to Manufacturing Processes**. *[S.l.]*: Wiley, 2011. ISBN 9781118213629.

HARRISON, R.; VERA, D.; AHMAD, B. Engineering methods and tools for cyber-physical automation systems. **Proceedings of the IEEE**, v. 104, n. 5, p. 973–985, May 2016.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of Artificial Intelligence Research**, v. 4, p. 237–285, 1996.

KRASHENINNIKOVA, E.; GARCíA, J.; MAESTRE, R.; FERNáNDEZ, F. Reinforcement learning for pricing strategy optimization in the insurance industry. **Engineering Applications of Artificial Intelligence**, v. 80, p. 8–19, 04 2019.

LAPAN, M. **Deep reinforcement learning hands-on : apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more**. Birmingham, UK: Packt Publishing, 2018. ISBN 9781788834247.

MALIK, Robi; TEIXEIRA, Marcelo. **Optimal Modular Control of Discrete Event Systems with Distinguishers and Approximations**. 2021. Accepted for publication at the Discrete Event Dynamic Systems Journal.

MALIK, Robi; WARE, Simon. On the computation of counterexamples in compositional nonblocking verification. **Discrete Event Dynamic Systems**, v. 30, p. 301–334, 2020.

MANU, G.; M., V. Kumar; NAGESH, H.; JAGADEESH, D.; GOWTHAM, M.B. Flexible manufacturing systems (fms), a review. **International Journal of Mechanical and Production Engineering Research and Development**, v. 8, p. 323–336, 04 2018.

MATSUI, Shoma; LAFORTUNE, Stéphane. **Synthesis of Winning Attacks on Communication Protocols using Supervisory Control Theory**. 2021.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning. **Nature**, Springer Science and Business Media LLC, v. 518, n. 7540, p. 529–533, Feb. 2015.

Mohajerani, S.; Lafortune, S. Transforming opacity verification to nonblocking verification in modular systems. **IEEE Transactions on Automatic Control**, v. 65, n. 4, p. 1739–1746, 2020.

MOHAJERANI, S.; MALIK, R.; FABIAN, M. Compositional synthesis of supervisors in the form of state machines and state maps. **Automatica**, v. 76, p. 277 – 281, 2017.

MOHAJERANI, Sahar; MALIK, Robi; WINTENBERG, Andrew; LAFORTUNE, Stephane; OZAY, Necmiye. Divergent stutter bisimulation abstraction for controller synthesis with linear temporal logic specifications. **Automatica**, v. 130, p. 109723, 08 2021.

MURATA, Tadao. Petri nets: Properties, analysis and applications. **Procedings of the IEEE**, v. 77, p. 541–580, 1989.

Nooruldeen, A.; Schmidt, K. W. Order-preserving languages for the supervisory control of automated manufacturing systems. **IEEE Access**, v. 8, p. 131901–131919, 2020.

PARK, H. S.; FEBRIANI, R. A. Modelling a platform for smart manufacturing system. **Procedia Manufacturing**, v. 38, p. 1660 – 1667, 2019. ISSN 2351-9789.

PRECUP, Radu-Emil; ROMAN, Raul-Cristian; TEBAN, Teodor-Adrian; ALBU, Adriana; PETRIU, Emil; POZNA, Claudiu. Model-free control of finger dynamics in prosthetic hand myoelectric-based control systems. **Studies in Informatics and Control**, v. 29, p. 399–410, 12 2020.

PUTERMAN, M. L. **Markov Decision Processes: Discrete Stochastic Dynamic Programming**. 1st. ed. USA: John Wiley  Sons, Inc., 1994. ISBN 0471619779.

PUTTEN, B.; SANDEN, B. V. D.; RENIERS, M.; VOETEN, J.; SCHIFFELERS, R. Supervisor synthesis and throughput optimization of partially-controllable manufacturing systems. **Discrete Event Dynamic Systems**, p. 1–33, 11 2020.

QAMSANE, Y.; HAMLAOUI, M. El; ABDELOUAHED, T.; PHILIPPOT, A. A model-based transformation method to design plc-based control of discrete automated manufacturing systems. *In*: **International Conference on Automation, Control, Engineering and Computer Science**. Sousse, Tunisia: *[s.n.]*, 2018. p. 4–11.

RAMADGE, P. J. G.; WONHAM, W. M. The control of discrete event systems. **Proceedings of the IEEE**, v. 77, n. 1, p. 81–98, 1989. ISSN 00189219.

ROMAN, Raul-Cristian; PRECUP, Radu-Emil; BOJAN-DRAGOS, Claudia-Adina; SZEDLAK-STINEAN, Alexandra-Iulia. Combined model-free adaptive control with fuzzy component by virtual reference feedback tuning for tower crane systems. **Procedia Computer Science**, v. 162, p. 267–274, 2019. ISSN 1877-0509.

SILVA, A. L.; RIBEIRO, R.; TEIXEIRA, M. Modeling and control of flexible context-dependent manufacturing systems. **Information Sciences**, v. 421, p. 1 – 14, 2017.

STRICKER, N.; KUHNLE, A.; STURM, R.; FRIESS, S. Reinforcement learning for adaptive order dispatching in the semiconductor industry. **CIRP Annals**, v. 67, n. 1, p. 511 – 514, 2018. ISSN 0007-8506.

SUTTON, R. S. Dyna, an integrated architecture for learning, planning, and reacting. Association for Computing Machinery, New York, NY, USA, v. 2, n. 4, p. 160–163, jul 1991.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. Second. *[S.l.]*: The MIT Press, 2018.

TADEPALLI, Prasad; OK, DoKyeong. **H-Learning: A Reinforcement Learning Method for Optimizing Undiscounted Average Reward**. USA, 1994.

TEIXEIRA, M.; CURY, J. E. R.; QUEIROZ, M. H. de. Exploiting distinguishers in local modular control of discrete-event systems. **IEEE Trans. on Automation Science and Engineering**, v. 15, n. 3, p. 1431–1437, July 2018.

TEIXEIRA, M.; MALIK, R.; CURY, J. E. R.; QUEIROZ, M. H. de. Supervisory control of des with extended finite-state machines and variable abstraction. **IEEE Transactions on Automatic Control**, v. 60, n. 1, p. 118–129, 2015.

TIWARI S., Al-Aswadi F.N. Gaurav D. Recent trends in knowledge graphs: theory and practice. **Soft Comput**, 2021.

TURNIP, A.; PANGGABEAN, Jonny H. Hybrid controller design based magneto-rheological damper lookup table for quarter car suspension. **International journal of artificial intelligence**, v. 18, p. 193–206, 2020.

WASCHNECK, B.; REICHSTALLER, A.; BELZNER, L.; ALTENMüLLER, T.; BAUERNHANSL, T.; KNAPP, A.; KYEK, A. Optimization of global production scheduling with deep reinforcement learning. **Procedia CIRP**, v. 72, p. 1264 – 1269, 2018. ISSN 2212-8271. 51st CIRP Conf. on Manufacturing Systems.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, Springer Science and Business Media LLC, v. 8, n. 3-4, p. 279–292, May 1992.

ZIELINSKI, Kallil M. C.; HENDGES, Lucas V.; TEIXEIRA, Marcelo; CASANOVA, Dalcimar. **Suplementary Material**. 2021. Available at: https://bit.ly/3h0KL9m.

ZIELINSKI, K. M. C.; TEIXEIRA, M.; RIBEIRO, R.; CASANOVA, D. **Concept and the implementation of a tool to convert industry 4.0 environments modeled as FSM to an OpenAI Gym wrapper**. 2020.

# APPENDIX

# APPENDIX A – COMPLEMENTARY EXAMPLES

In this section, we present some examples to facilitate the understanding of the methodologies presented in this document. Our goal here is to bring the reviewer closer to the research methodology.

**Example 1** (FSM compositions)**.** Consider the example in Fig 10.

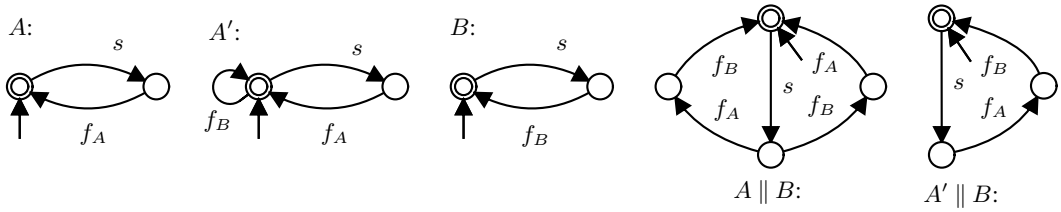
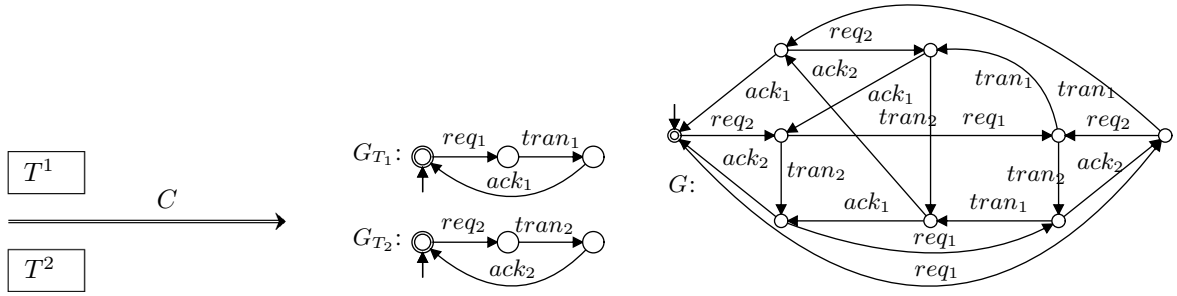



**Figure 10 – Graphical view of** FSMs **and compositions.**

The three machines, $A$, $A'$, and $B$, start with the same event $s$ ($s \in \Sigma_A \cap \Sigma_B \cap \Sigma_{A'}$), which is therefore merged (case (i) of composition definition) in both $A \parallel B$ and $A' \parallel B$. In $A \parallel B$, machines $A$ and $B$ can finish in any order after $s$, as $f_A \in \Sigma_A \setminus \Sigma_B$ and $f_B \in \Sigma_B \setminus \Sigma_A$ (cases (ii) and (iii)). However, in $A' \parallel B$, machine $A$ is forced to finish before $B$ can finish (event $f_B$ is prohibited after $s$ in $A'$ until $f_A$ occurs). In this case, $f_B \in \Sigma_{A'} \cap \Sigma_B$ (suggesting case (i)), but $A' \xrightarrow{s} q \not\xrightarrow{f_B}$, which falls in case (iv) and causes the event to be definitively disabled by composition, i.e., $A' \parallel B \xrightarrow{s} q \not\xrightarrow{f_B}$.

**Example 2** (Control synthesis)**.** Consider an example of two transmitters, $T_1$ and $T_2$, sharing a communication channel $C$, as in Fig. 11(a). $T_1$ and $T_2$ can be respectively modeled as FSMs $G_{T_1}$ and $G_{T_2}$ in Fig. 11(b).




(a) Layout of the transmission process.

(b) Plant model.

(c) Composed plant model $G = G_{T_1} \| G_{T_2}$.

**Figure 11 – Example of a concurrent transmission system.**

The following events are observable:

- $req_1$ and $req_2$: request messages arriving for transmission in $T^1$ and $T^2$, respectively. These events are controllable, since the requisition is manually triggered by the system;
- $tran_1$ and $tran_2$: model the start of transmission in $T^1$ and $T^2$, respectively. They are uncontrollable, since the control agent can not disable; and
- $ack_1$ and $ack_2$: reset the channel $C$ for new transmissions. Also uncontrollable events, because the acknowledge signal is not triggered by the control agent.

It is assumed that $\Sigma_{\mathrm{c}} = \{tran_i\}$, while $\Sigma_{\mathrm{u}} = \{req_i, ack_i\}$, for $i = 1{,}2$. The plant $G = G_{T_1} \| G_{T_2}$ has 9 states and 18 transitions, and it is displayed in Fig 11(c). The behavior of each transmitter (and of $G$) is unrestricted, i.e., it does not consider channel limitations neither the sharing of $C$ with other transmitters. One possible specification is to mutually exclude transmissions in $C$, which is modeled by $E$ in Fig. 12.

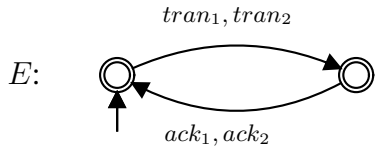



**Figure 12 – Mutual exclusion specification for the channel $C$.**

$E$ allows both transmitters to start transmission (events $tran_1$ and $tran_2$) in the initial state but, as soon as one of them occupies the channel, the other is prohibited to transmit until an acknowledge ($ack_1$ or $ack_2$) is received (both $tran_1$ and $tran_2$ are disabled in the non-initial state).

From the composition $K = G \| E$, the optimal controller $\sup\mathcal{C}(K,G)$ can be obtained. In this example it has 8 states and 14 transitions, and the model can be used to generate hardware-compatible code (Qamsane *et al.* 2018).

**Example 3** (SARSA vs N-step SARSA )**.** Consider an $\mathrm{MDP}$ with 6 states and 2 actions as shown in the example in Fig 13. A robot moves through six rooms with the objective of collecting raw material. Consider that room number 5, which is correlated to state 5 in our $\mathrm{MDP}$, has a considerable mount of raw material, so that the robot receives a reward of 10 after it collects it all. Differently, room number 1 has also raw material available to collect, but its total reward is only 2.

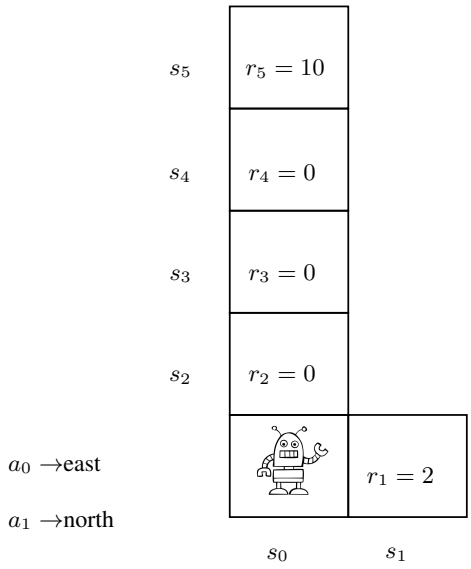



**Figure 13 – Example of a robot walking in a room**

So now we show the outcome of the environment for both algorithms. First of all, we calculate the Q-values for SARSA, since it is less complex than its N-step version. After initializing all Q-values to 0, we apply Equation 5 to both outcomes: going east, and going north. We also allocate some values to our training parameters. In this example, we consider a learning rate of $\alpha = 0.1$, and a discount factor $\gamma = 0.9$. Since there are only two paths that the robot can move, we follow the only possible policy for each path in the environment.

Next, we show the Q-values for all the robot outcomes after one updating step on all states. We can note that the Q-value of making an action $a_1$ (north) in the initial state $s_0$ is 0.000729, while the value of making an action $a_0$ in the same state is 0.2. See that the reward on going 4 steps north is higher than the reward of going just one step east. This is because SARSA loses track of future rewards over time, and to overcome that, it needs to keep memory storage in order to keep track of the path that is taken.

$$
\begin{aligned}
Q(s_4,a_1) &= Q(s_4,a_1) + \alpha * (r_5 + \gamma * Q(s_5,a_1)) \\
&= 0 + 0.1 * (10 + 0.9 * 0) = 1
\end{aligned}
$$

$$
\begin{aligned}
Q(s_3,a_1) &= Q(s_3,a_1) + \alpha * (r_4 + \gamma * Q(s_4,a_1)) \\
&= 0 + 0.1 * (0 + 0.9 * 1) = 0.09
\end{aligned}
$$

$$
\begin{aligned}
Q(s_2,a_1) &= Q(s_2,a_1) + \alpha * (r_3 + \gamma * Q(s_3,a_1)) \\
&= 0 + 0.1 * (0 + 0.9 * 0.09) = 0.0081
\end{aligned}
$$

$$
\begin{aligned}
Q(s_0,a_0) &= Q(s_0,a_0) + \alpha * (r_1 + \gamma * Q(s_1,a_0)) \\
&= 0 + 0.1 * (2 + 0.9 * 0) = 0.2
\end{aligned}
$$

$$
\begin{aligned}
Q(s_0,a_1) &= Q(s_0,a_1) + \alpha * (r_2 + \gamma * Q(s_2,a_1)) \\
&= 0 + 0.1 * (0 + 0.9 * 0.0081) = 0.000729
\end{aligned}
$$

Now, we use N-step SARSA to look at the environment's outcome and see the behavior of the $\mathrm{RL}$ agent looking at its Q-values. The parameters used for $\alpha$ and $\gamma$ are the same. In addition, we use $n = 5$ to keep track of 5 steps further. After one single update in each step of the environment, the Q-values are the following.

$$
\begin{aligned}
Q(s_4,a_1) &= Q(s_4,a_1) + \alpha * (r_5 + \gamma * Q(s_5,a_1)) \\
&= 0 + 0.1 * (10 + 0.9 * 0) = 1
\end{aligned}
$$

$$
\begin{aligned}
Q(s_3,a_1) &= Q(s_3,a_1) + \alpha * \\
&\quad (r_4 + \gamma * r_5 + \gamma^2 * Q(s_4,a_1)) \\
&= 0 + 0.1 * (0 + 0.9 * 10 + 0.81 * 1) \\
&= 0.981
\end{aligned}
$$

$$
\begin{aligned}
Q(s_2,a_1) &= Q(s_2,a_1) + \alpha * \\
&\quad (r_3 + \gamma * r_4 + \gamma^2 * r_5 + \gamma^3 * Q(s_3,a_1)) \\
&= 0 + 0.1 * (0 + 0.9 * 0 + 0.81 * 10 + 0.729 * 0.981) \\
&= 0.866
\end{aligned}
$$

$$
\begin{aligned}
Q(s_0,a_0) &= Q(s_0,a_0) + \alpha * \\
&\quad (r_1 + \gamma * Q(s_1,a_0)) \\
&= 0 + 0.1 * (2 + 0.9 * 0) \\
&= 0.2
\end{aligned}
$$

$$
\begin{aligned}
Q(s_0,a_1) &= Q(s_0,a_1) + \alpha * \\
&\quad (r_2 + \gamma * r_3 + \gamma^2 * r_4 + \\
&\quad \gamma^3 * r_5 + \gamma^4 * Q(s_2,a_1)) \\
&= 0 + 0.1 * (0 + 0.9 * 0 + 0.81 * 0 + \\
&\quad 0.729 * 10 + 0.656 * 0{,}866) \\
&= 0{,}785
\end{aligned}
$$

Note that, in this case, $Q(s_0,a_0) = 0.2$, the same as in SARSA, but now $Q(s_0,a_1) =$

$0.785$, which is higher than the previous $Q(s_0,a_0)$. So, if the agent decides to follow a greedy policy in the initial state, it always chooses to pick up the action $a_1$ in the initial state, because it has a higher Q-value and it produces a better profit by the end of the iteration.

**Example 4** (Assembler System for Computer Manufacturing)**.** *Figure 14 exposes a partial version of system that assembles personal computers using 2 production lines. A computer enters the system through an event $a_1$ and leaves the system through the events $out_1$ and $out_2$. Rectangular blocks represent machines, while circular blocks represent temporary storage buffers. In this example, a computer may either pass directly through machine $M_1$ to $M_3$, or manufacture further (and optionally) in machine $M_2$. For example, let us assume that machine $M_2$ adds an optional GPU attribute to be installed in the computer.*

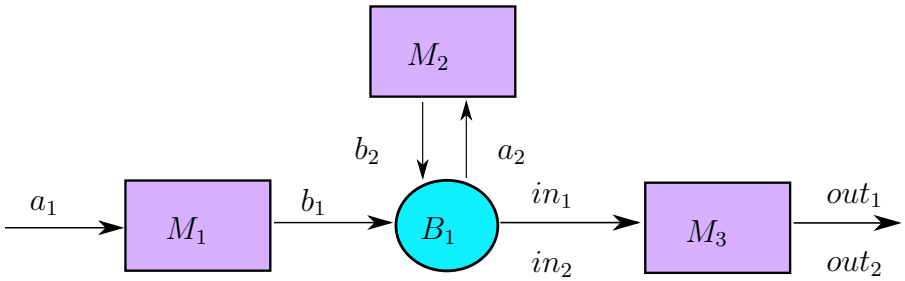



**Figure 14 – Partial layout of an assembler system for personal computer manufacturing.**

The first step to apply our method, is to treat the system as a DES, modeling the FSMs that represents the plant, and express the specifications for the system to follow. Table 5 shows the description and controllability of the events to be observed. Figure 15 shows the models for each machine in the system, whose composition leads to the global unrestricted $G$. In order to add restrictions of overflow and underflow in the buffer $B_1$, to prevent computers to be handled inappropriately, we design the FSM $E$ that controls the events $in_1$ and $in_2$ as the computer enters or not the optional line, as shown in Figure 16. The composition $K = G \| E$ represents the desired behavior for the system under control.

In sequence, the optimal supervisor of the system $\sup\mathcal{C}(K,G)$ was processed by the SCT and it resulted in an FSM with 15 states and 24 transitions. Then, we translate this FSM into an MDP, converting the FSM states, events, and transition function into the corresponding MDP states, actions, and transition function, respectively (see more details in Section **??**). For this example, we do not assume transition probabilities in the model, in order to simplify the problem in question. The reward values for each action in the environment are shown in Table 6.

**Table 5 – Description and controllability of the events used in the computer assembler system.**

| Event | Controllability | Description |
|---|---|---|
| $a_1$ | Controllable | Inserts a computer in machine $M_1$. |
| $a_2$ | Controllable | Inserts a computer in machine $M_2$. |
| $in_1$ | Controllable | Sends a computer to $M_3$ line without optional items |
| $in_2$ | Controllable | Sends a computer to $M_3$ line with optional items |
| $b_1$ | Uncontrollable | Inserts a computer in buffer $B_1$ through machine $M_1$. |
| $b_2$ | Uncontrollable | Inserts a computer in buffer $B_1$ through machine $M_2$. |
| $out_1$ | Uncontrollable | Dispatches a computer off the production line without optional items |
| $out_2$ | Uncontrollable | Dispatches a computer off the production line with optional items |

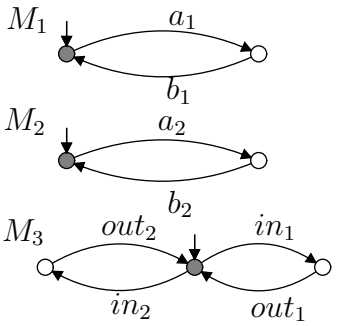



**Figure 15 – Design of the free behavior of each machine of the system.**

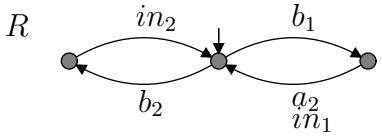



**Figure 16 – Restriction of underflow and overflow in buffer $B_1$ and also the control of events $in_1$ and $in_2$.**

Finally, after splitting the action set into controllable and uncontrollable actions, we simulate the environment for 3000 episodes, considering that an episode ends after 100 steps of the agent in the environment, and also considering the following values for the parameters: $\alpha = 0.1$, $\gamma = 0.9$, $\epsilon = 0.1$, and $n = 10$. To validate the method, we simulated each agent (with and without $\mathrm{RL}$) for 50 episodes. The mean reward obtained for each one is shown in Figure 17.

| Action set A | Reward set R |
|---|---|
| $a_1$ | -0.1 |
| $a_2$ | -0.1 |
| $in_1$ | -0.1 |
| $in_2$ | -0.1 |
| $b_1$ | -0.1 |
| $b_2$ | -0.1 |
| $out_1$ | 0.7 |
| $out_2$ | 0.9 |

**Table 6 – Rewards adopted for the computer assembler example.**

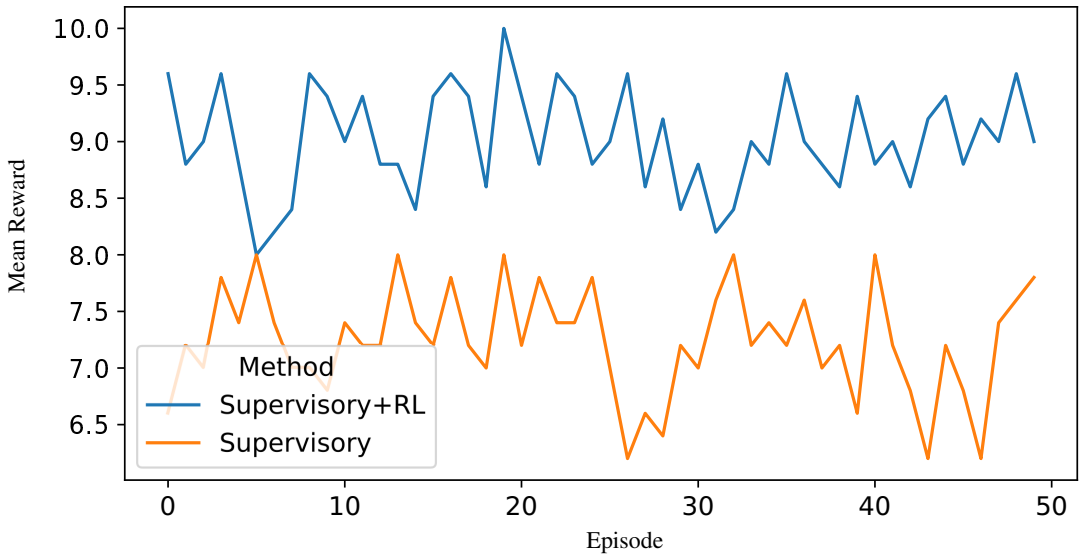



**Figure 17 – Mean rewards obtained for the case study of a system that assembles computer parts.**

Remark that, as expected, the intelligent approach returned a better profit in comparison with the non-intelligent method. The rewards considered for each action presented in Table 6 shows that computers with manufacture of optional devices return a better reward (0.9) than a computer manufactured through the otherwise flow (0.7). However, the process of assembling an optional device on the computer has a cost, and the RL agent can balance it in order to maximize the final reward to be obtained. Since the approach without RL keeps taking random action decisions, it comparably receives a lower profit.

**Example 5** (Automotive industry with a rotating table)**.** *This case study exposes a particular case in which the automotive industry, presented as the main example of our work, is complemented with a rotating table that separates both approved and not approved cars. Figure 18 shows the components of the system, note that the table has 4 places in it. Cars enter the table in position* $P_1$ *and travel all the way through position* $P_4$ *(if they are approved by* $M_Q$*), where a robotic arm* $M_A$ *picks the car and transfers it out of the production line. Non approved cars are discarded by machine* $M_D$ *in position* $P_2$ *while approved cars pass by* $P_2$ *without being discarded and go to position* $P_3$ *to be tested by machine* $M_T$*. Finally, the approved cars go to* $P_4$ *in order to be dispatched of the production line.*

The modeled plants, representing the unrestricted behavior of the components, are presented in Fig. 20. Note that there are 4 models, one for each of the three new machines of the system and also a model for the rotating table. Event $s_{Tu}$ from $Table$ model indicates that the

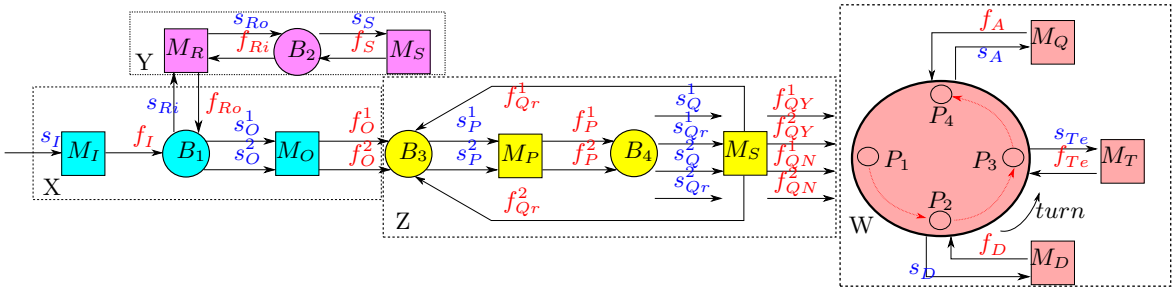



**Figure 18 – Automotive system complemented with a rotating table at the end. Machine $M_D$ discards cars that are not approved by $M_Q$ and send approved cars out of the production line by a robotic arm represented by machine $M_A$**

.

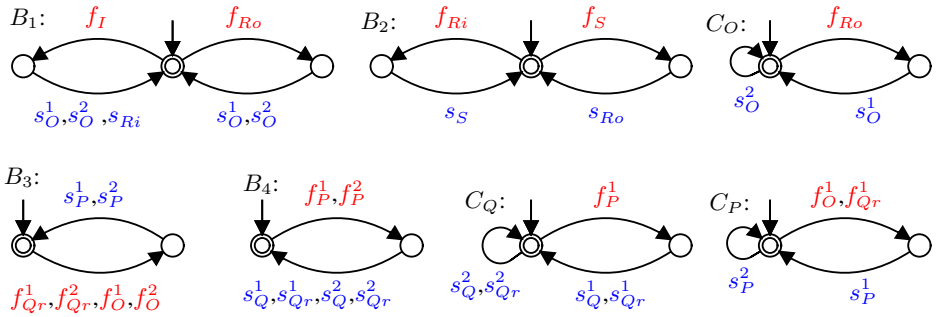


**Figure 19 – Restriction of overflow and underflow of buffers in blocks X,Y and Z**

table is moving, while event $f_{Tu}$ is a signal that informs that it has stopped moving. From model $M_D$, note that there is a loop at the initial state, with event $s_{Nd}$ indicating that the car that entered position $P_2$ can not be discarded, so it has to go directly through position $P_3$.

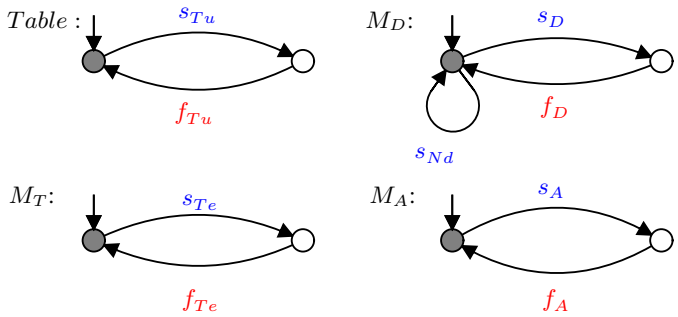


**Figure 20 – Design of the unrestricted behavior of the components representing block W on the Automotive system.**

The restrictions adopted for the components in block W are represented in Fig. 21, while the description of each one of the models is detailed by Table 7.

By synthesizing the models in order to get the optimal supervisor of the system, we applied the SCT and the result was a state machine with 153289 states and 741736 transitions, that is, a state space of around 117 times higher than the system without the rotating table.

In order for this model to be apt to learn by RL, we adopted the rewards and probabil-

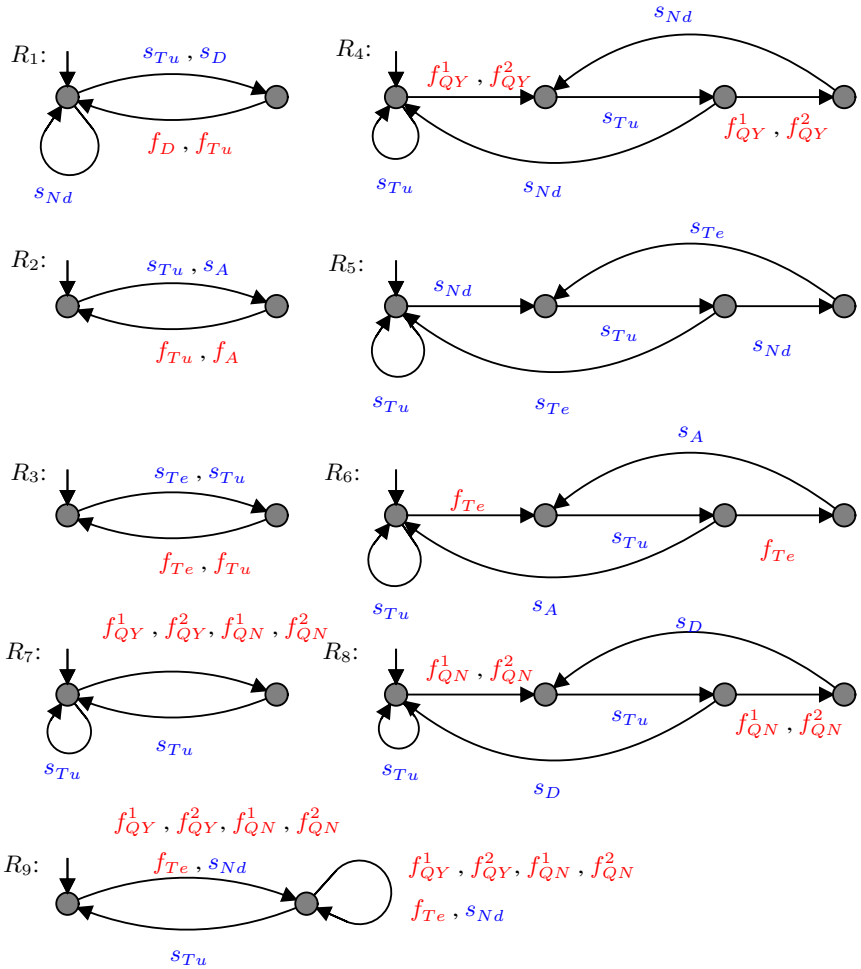



**Figure 21 – Design of the restrictions implemented in block** W**.**

ities represented in Table 8. The model was trained for 10 scenarios varying the probability of rejection of a car type 2. Each scenario was trained for 10000 episodes and the mean reward obtained is represented by Fig. 22.

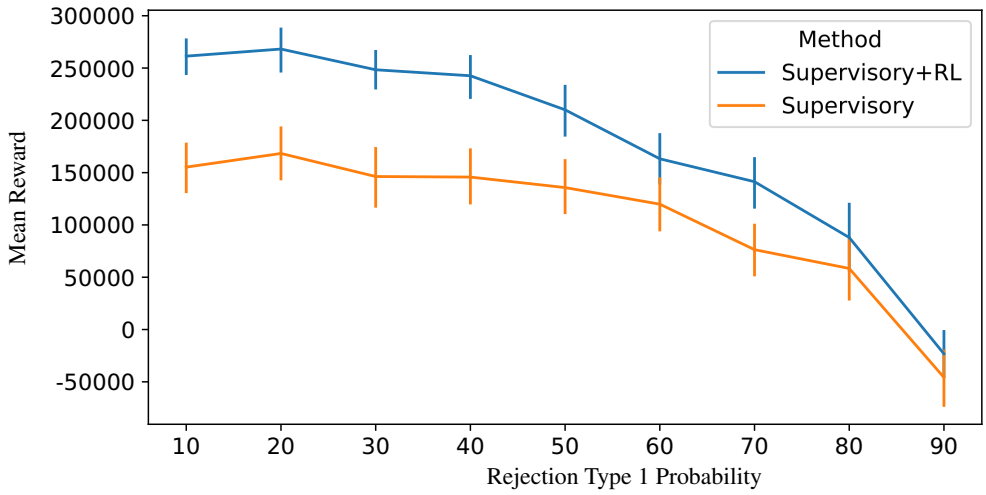



**Figure 22 – Mean reward obtained for the system with a complementary rotating table example.**

| Specification | Description |
|---|---|
| R1 | Avoid to turn the table while discarding a car |
| R2 | Avoid to turn the table while the robotic arm is transferring a car |
| R3 | Avoid to turn the table while testing a car |
| R4 | Controls the flow of approved cars between positions P1 and P2 |
| R5 | Controls the flow of approved cars between positions P2 and P3 |
| R6 | Controls the flow of approved cars between positions P3 and P4 |
| R7 | Controls the overflow of pieces in P1 |
| R8 | Controls the flow of non approved cars between P1 and P2 |
| R9 | Avoid the table from moving when there are no cars |

**Table 7 – Description of each specification model presented in Fig 21.**

| **Controllable Actions** | | | **Uncontrollable Actions** | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -30000 | 10 - 90 |
| $s^1_O$ | -10 | Not considered | $f^2_{QN}$ | -20000 | 90 |
| $s^2_O$ | -10 | Not considered | $f_I$ | -10 | Not considered |
| $s^1_P$ | -10 | Not considered | $f^1_O$ | -10 | Not considered |
| $s^2_P$ | -10 | Not considered | $f^2_O$ | -10 | Not considered |
| $s^1_Q$ | -10 | Not considered | $f^1_P$ | -10 | Not considered |
| $s^1_{Qr}$ | -10 | Not considered | $f^2_P$ | -10 | Not considered |
| $s^2_Q$ | -10 | Not considered | $f_{Ri}$ | -10 | Not considered |
| $s^2_{Qr}$ | -10 | Not considered | $f_{Ro}$ | -10 | Not considered |
| $s_{Ri}$ | -10 | Not considered | $f_S$ | -10 | Not considered |
| $s_{Ro}$ | -10 | Not considered | $f^1_{QY}$ | 30000 | 90 - 10 |
| $s_S$ | -10 | Not considered | $f^2_{QY}$ | 20000 | 90 |
| $s_D$ | -10 | Not considered | $f^1_{Qr}$ | -1000 | 10 - 90 |
| $s_{Nd}$ | -10 | Not considered | $f^2_{Qr}$ | -1000 | 90 |
| $s_{Tu}$ | -10 | Not considered | $f_D$ | -10 | Not considered |
| $s_{Te}$ | -10 | Not considered | $f_{Tu}$ | -10 | Not considered |
| $s_A$ | -10 | Not considered | $f_{Te}$ | -10 | Not considered |
| | | | $f_A$ | -10 | Not considered |

**Table 8 – Rewards and probabilities adopted for the example. The probability of rejection of a car of type 2 was fixed at 50% while varying the probability of a type 1.**

We can note that, although the model has a significantly higher state space for the RL agent to explore, it could still learn well and obtain a higher profit that the non-intelligent approach, showing that regardless of how big the state space of the environment is, our method will still be able to get a better reward than the method without intelligence.

# APPENDIX B – ADDITIONAL RESULTS

For all our case studies, we show the comparison of mean reward and event occurrences, for both the SCT method and the n-step sarsa RL method combined. In order to show that our method works for many configurations of a system, we picked the example from Figure 4. For all tests, 9 scenarios were considered, and each one was trained through 5000 episodes.

## B.1 CASE 1

In our first additional study case, we fixed the probability of a car of type 2 to be approved at 50% while varying the approval of type from 10% to 90% . Table 9 shows the parameters used for this scenario.

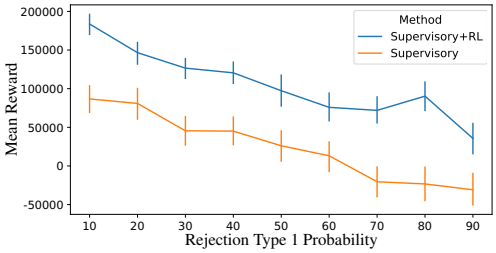
| **Controllable Actions** | | | **Uncontrollable Actions** | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -30000 | 10 - 90 |
| $s^1_O$ | -10 | Not considered | $f^2_{QN}$ | -20000 | 50 |
| $s^2_O$ | -10 | Not considered | $f_I$ | -10 | Not considered |
| $s^1_P$ | -10 | Not considered | $f^1_O$ | -10 | Not considered |
| $s^2_P$ | -10 | Not considered | $f^2_O$ | -10 | Not considered |
| $s^1_Q$ | -10 | Not considered | $f^1_P$ | -10 | Not considered |
| $s^1_{Qr}$ | -10 | Not considered | $f^2_P$ | -10 | Not considered |
| $s^2_Q$ | -10 | Not considered | $f_{Ri}$ | -10 | Not considered |
| $s^2_{Qr}$ | -10 | Not considered | $f_{Ro}$ | -10 | Not considered |
| $s_{Ri}$ | -10 | Not considered | $f_S$ | -10 | Not considered |
| $s_{Ro}$ | -10 | Not considered | $f^1_{QY}$ | 30000 | 90 - 10 |
| $s_S$ | -10 | Not considered | $f^2_{QY}$ | 20000 | 50 |
| | | | $f^1_{Qr}$ | -1000 | 10 - 90 |
| | | | $f^2_{Qr}$ | -1000 | 50 |

**Table 9 – Rewards and probabilities adopted for the example. The probability of rejection of a car of type 2 was fixed at 50% while varying the probability of a type 1.**

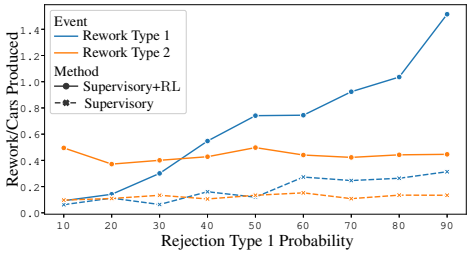
Figure 23(a) shows the mean reward obtained. Note that the intelligent method continues to give a higher profit than the non-intelligent approach over all values, but both methods decrease their reward as the rejection probability increases, which is quite expected.

Figure 23(c) shows the frequency of accepted and rejected cars for the n-step SARSA method while Figure 23(d) shows it for the supervisory method. Note that the intelligent method chooses to produce more cars of type 1 over low rejection probabilities, while the non-intelligent is random over its choices.

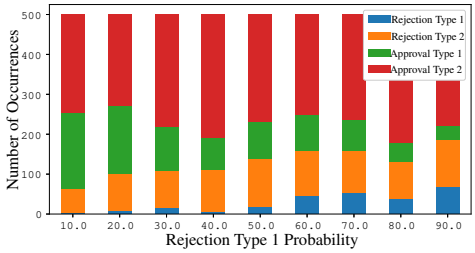
Also, as the rejection probability increases, the number of rework cycles per car produced also increases, which is shown in Fig. 23(b). Also from this figure we can note that the



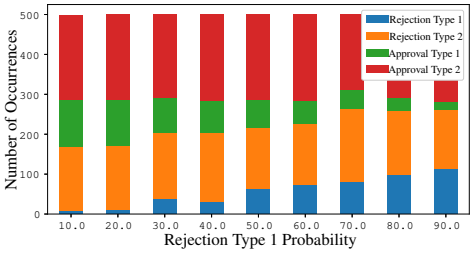

(a) Mean reward obtained.




(b) Number of rework cycles per car produced for each event and method.




(c) Frequency of occurrence of events with the N-step SARSA method.




(d) Frequency of occurrence of events without the N-step SARSA method.

**Figure 23 – Results of applying the N-step SARSA while fixing the probability of rejection of cars type 2, and varying the probability of rejection of cars type 1. See Table 9 for the adopted parameters.**

supervisory method also increases its rework cycles, and the cars type 2 are kept approximately constant. This is because its rejection probability was fixed.

## B.2 CASE 2

In this study case, we fixed the probability of a car of type 2 to be approved at 10%. The parameters adopted are represented in Table 10.

In this scenario, the mean reward obtained by the n-step SARSA method is also better than the supervisory method, as shown in Figure 24(a), but it decreases over high rejection probabilities.

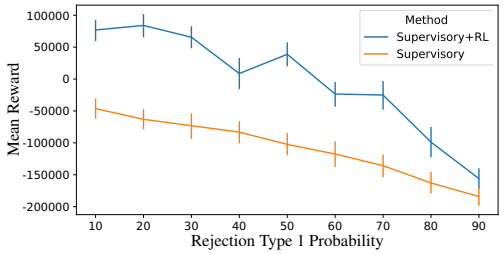
Also, note that even with high rejection probabilities for both types of car, the mean reward is still better under the intelligent method. The reason is because the intelligent agent decides in most cases to rework in order for cars to be approved, as it is shown in Figure 24(b).

## B.3 CASE 3

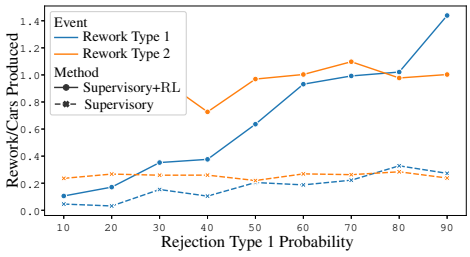
In this case study, we varied the rework cost while fixing the other values. The scenario was analyzed following the values on Table 11, which varied the rework cost from 1000 to 9000

| Controllable Actions | | | Uncontrollable Actions | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -30000 | 10 - 90 |
| $s^1_O$ | -10 | Not considered | $f^2_{QN}$ | -20000 | 90 |
| $s^2_O$ | -10 | Not considered | $f_I$ | -10 | Not considered |
| $s^1_P$ | -10 | Not considered | $f^1_O$ | -10 | Not considered |
| $s^2_P$ | -10 | Not considered | $f^2_O$ | -10 | Not considered |
| $s^1_Q$ | -10 | Not considered | $f^1_P$ | -10 | Not considered |
| $s^1_{Qr}$ | -10 | Not considered | $f^2_P$ | -10 | Not considered |
| $s^2_Q$ | -10 | Not considered | $f_{Ri}$ | -10 | Not considered |
| $s^2_{Qr}$ | -10 | Not considered | $f_{Ro}$ | -10 | Not considered |
| $s_{Ri}$ | -10 | Not considered | $f_S$ | -10 | Not considered |
| $s_{Ro}$ | -10 | Not considered | $f^1_{QY}$ | 30000 | 90 - 10 |
| $s_S$ | -10 | Not considered | $f^2_{QY}$ | 20000 | 10 |
| | | | $f^1_{Qr}$ | -1000 | 10 - 90 |
| | | | $f^2_{Qr}$ | -1000 | 90 |

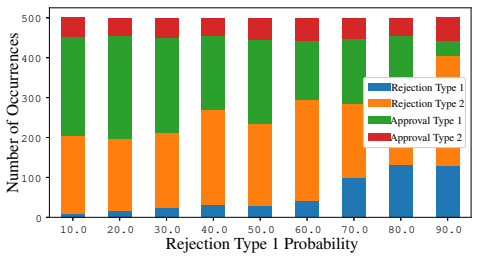
**Table 10 – Rewards and probabilities adopted for case 2. This time we fixed the probability of rejection of a type 2 car at 90%.**



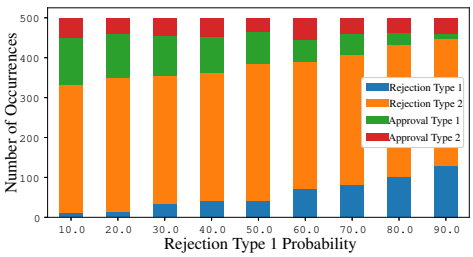

(a) Mean reward obtained.




(b) Number of rework cycles per car produced for each event and method.




(c) Frequency of occurrence of events with the N-step SARSA method.

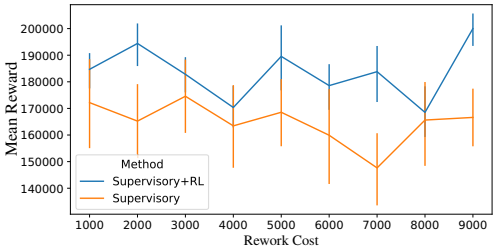



(d) Frequency of occurrence of events without the N-step SARSA method.

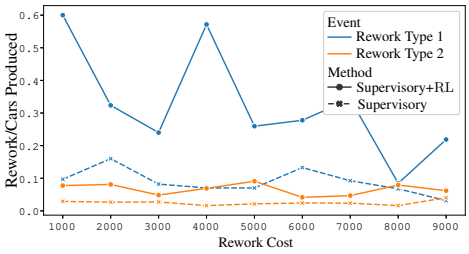
**Figure 24 – Results of N-step SARSA application fixing the probability of rejection of a type 2 car while variating the probability of rejection of a type 1 car. See Table 10 for the adopted parameters.**

into 9 different cases.

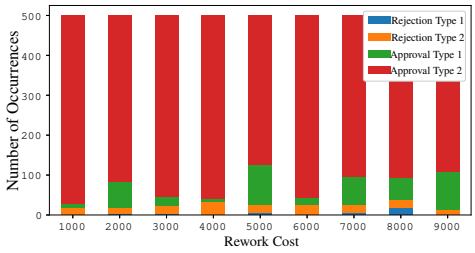
The mean reward is shown in Figure 25(a), which shows that even when the rework cost is high, the mean reward does not have a significant variation in its absolute value for both methods.



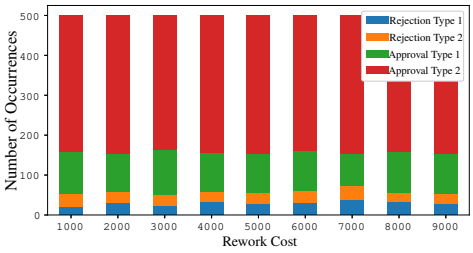

(a) Mean Reward Obtained.




(b) Number of rework cycles per car produced for each event and method.




(c) Frequency of occurrence of events with the N-step SARSA method.




(d) Frequency of occurrence of events without the N-step SARSA method.

**Figure 25 – Results of N-step SARSA application variating the probability of the rework cost. See Table 11 for the adopted parameters.**

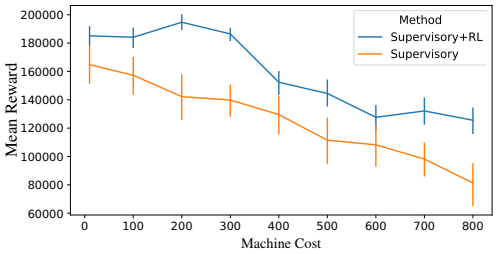
Figure 25(b) shows the number of rework occurrences per car produced in the test. It shows that the rework decisions tend to occur more frequently under the intelligent method, even under high costs, but in this case with less frequency. This is due to the even higher rejection costs, and to the fact that the rejection probability is low, so that it is worth for the agent to decide reworking the car.

The event frequency of occurrence for the intelligent method is shown on Figure 25(c), while for the non-intelligent approach is shown in Figure 25(d). Both results are expected, since the intelligent agent prefers to work safely, and it decides to work with cars of type 2 in most cases. This is due to the higher chances of approval, while the non-intelligent method chooses randomly and this is expected to reflect on performance.

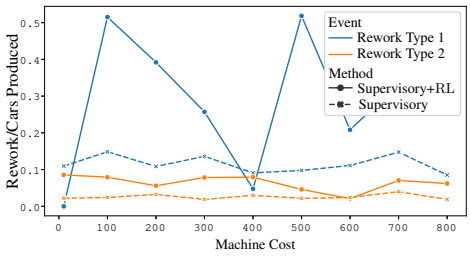
## B.4 CASE 4

For this scenario, we varied the cost of an event aside from rejection, approval and rework. Table 12 shows the values assumed for this case, varying the cost of a step from 10 through 800 in 9 different cases.

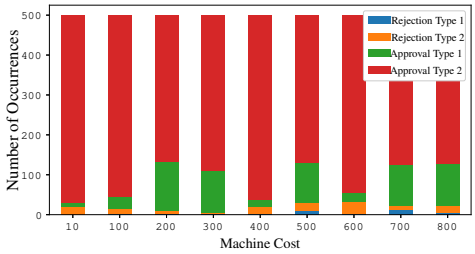
The mean reward for this case is shown in Figure 26(a). Remark that when the cost increases, the production profit decreases, as expected. Also, the RL method has shown better performance at taking decisions in comparison with the pure supervisory approach.



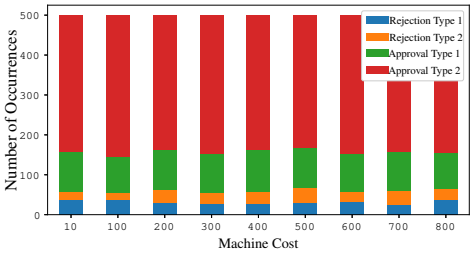

(a) Mean reward obtained.




(b) Number of rework cycles per car produced for each event and method.




(c) Frequency of occurrence of events with the N-step SARSA method.




(d) Frequency of occurrence of events without the N-step SARSA method.

**Figure 26 – Results of applying the N-step SARSA while varying the cost of a step in the system. See Table 12 for the adopted parameters.**

Fig. 26(b) shows the number of rework cycles per car produced for each method. We can see that without RL, the agent opts to make less rework cycles than the intelligent approach, which is expected due to the random manner it takes its actions. Also, cars type 1 tend to have more rework cycles, which is also expected, since their rejection probability is higher.

Fig. 26(c) shows the relation of cars approved and cars rejected in the simulation using the intelligent method, while Fig. 26(d) shows this relation using the pure supervisory method.

In the intelligent method we can see that the agent prefers to work with cars of type 2, since the probability of approval is higher. As for the supervisory method, there is an expected randomness in this decisions, since it balances between producing type 1 or 2.

| Controllable Actions | | | Uncontrollable Actions | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -30000 | 30 |
| $s^1_O$ | -10 | Not considered | $f^2_{QN}$ | -20000 | 10 |
| $s^2_O$ | -10 | Not considered | $f_I$ | -10 | Not considered |
| $s^1_P$ | -10 | Not considered | $f^1_O$ | -10 | Not considered |
| $s^2_P$ | -10 | Not considered | $f^2_O$ | -10 | Not considered |
| $s^1_Q$ | -10 | Not considered | $f^1_P$ | -10 | Not considered |
| $s^1_{Qr}$ | -10 | Not considered | $f^2_P$ | -10 | Not considered |
| $s^2_Q$ | -10 | Not considered | $f_{Ri}$ | -10 | Not considered |
| $s^2_{Qr}$ | -10 | Not considered | $f_{Ro}$ | -10 | Not considered |
| $s_{Ri}$ | -10 | Not considered | $f_S$ | -10 | Not considered |
| $s_{Ro}$ | -10 | Not considered | $f^1_{QY}$ | 30000 | 70 |
| $s_S$ | -10 | Not considered | $f^2_{QY}$ | 20000 | 90 |
| | | | $f^1_{Qr}$ | -1000/-9000 | 30 |
| | | | $f^2_{Qr}$ | -1000/-9000 | 10 |

**Table 11 – Rewards and probabilities adopted for case 3, varying the rework cost while fixing the other values.**

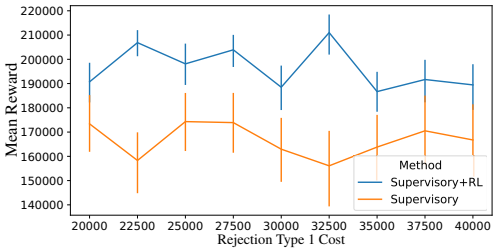
| Controllable Actions | | | Uncontrollable Actions | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -30000 | 30 |
| $s^1_O$ | -10/-800 | Not considered | $f^2_{QN}$ | -20000 | 10 |
| $s^2_O$ | -10/-800 | Not considered | $f_I$ | -10/-800 | Not considered |
| $s^1_P$ | -10/-800 | Not considered | $f^1_O$ | -10/-800 | Not considered |
| $s^2_P$ | -10/-800 | Not considered | $f^2_O$ | -10/-800 | Not considered |
| $s^1_Q$ | -10/-800 | Not considered | $f^1_P$ | -10/-800 | Not considered |
| $s^1_{Qr}$ | -10/-800 | Not considered | $f^2_P$ | -10/-800 | Not considered |
| $s^2_Q$ | -10/-800 | Not considered | $f_{Ri}$ | -10/-800 | Not considered |
| $s^2_{Qr}$ | -10/-800 | Not considered | $f_{Ro}$ | -10/-800 | Not considered |
| $s_{Ri}$ | -10/-800 | Not considered | $f_S$ | -10/-800 | Not considered |
| $s_{Ro}$ | -10/-800 | Not considered | $f^1_{QY}$ | 30000 | 70 |
| $s_S$ | -10/-800 | Not considered | $f^2_{QY}$ | 20000 | 90 |
| | | | $f^1_{Qr}$ | -1000 | 30 |
| | | | $f^2_{Qr}$ | -1000 | 10 |

**Table 12 – Rewards and probabilities adopted for the case 4, which varies the cost of a step that is not an approval, rejection, or rework.**

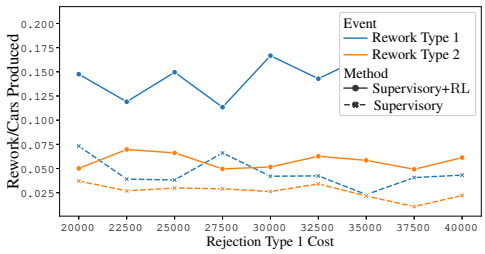
## B.5 CASE 5

We conducted yet another test fixing all values, except the cost of rejection of cars type 1. Table 13 shows the values adopted for this test case.

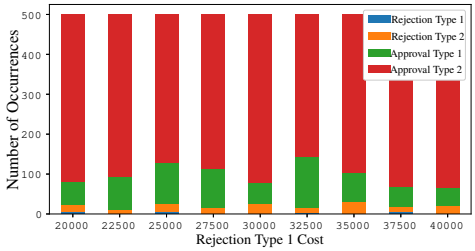
The graph of the mean reward presented in Fig. 27(a) shows that no matter the cost of rejection, the intelligent agent still makes decisions such that its mean reward is better than the supervisory approach.



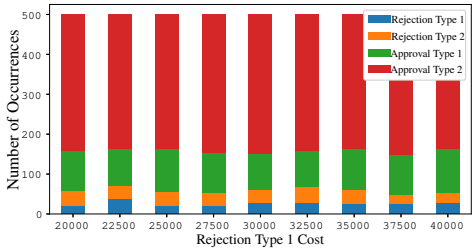

(a) Mean reward obtained.




(b) Number of Rework cycles per car produced for each event and method.




(c) Frequency of occurrence of events with N-step SARSA method.




(d) Frequency of occurrence of events without the N-step SARSA method.

**Figure 27 – Results for the system under variation of the cost to reject cars type 1 (Table 13).**

Fig. 27(b) shows the number of rework cycles per car produced in each method. As we can see, there is more rework cycles for cars type 1, since its probability of rejection is higher than otherwise, as expected. We can also note that the supervisory method does not decide to do as many rework cycles as the intelligent approach, since it includes randomness on its decisions.

On the other hand, Figures 27(c) and 27(d) represent the frequency of approved and rejected cars, both using RL and not, respectively. It is worth to notice that the supervisory method sends more cars to customization than the intelligent method, and consequently receives more rejections, since its rejection probability is higher than cars type 1. This justifies the lower reward received.

| Controllable Actions | | | Uncontrollable Actions | | |
|---|---|---|---|---|---|
| **Action** | **Reward** | **Probability (%)** | **Action** | **Reward** | **Probability (%)** |
| $s_I$ | -10 | Not considered | $f^1_{QN}$ | -20000/-40000 | 30 |
| $s^1_O$ | -10 | Not considered | $f^2_{QN}$ | -20000 | 10 |
| $s^2_O$ | -10 | Not considered | $f_I$ | -10 | Not considered |
| $s^1_P$ | -10 | Not considered | $f^1_O$ | -10 | Not considered |
| $s^2_P$ | -10 | Not considered | $f^2_O$ | -10 | Not considered |
| $s^1_Q$ | -10 | Not considered | $f^1_P$ | -10 | Not considered |
| $s^1_{Qr}$ | -10 | Not considered | $f^2_P$ | -10 | Not considered |
| $s^2_Q$ | -10 | Not considered | $f_{Ri}$ | -10 | Not considered |
| $s^2_{Qr}$ | -10 | Not considered | $f_{Ro}$ | -10 | Not considered |
| $s_{Ri}$ | -10 | Not considered | $f_S$ | -10 | Not considered |
| $s_{Ro}$ | -10 | Not considered | $f^1_{QY}$ | 30000 | 70 |
| $s_S$ | -10 | Not considered | $f^2_{QY}$ | 20000 | 90 |
| | | | $f^1_{Qr}$ | -1000 | 30 |
| | | | $f^2_{Qr}$ | -1000 | 10 |

**Table 13 – Rewards and probabilities adopted for case 5, fixing all values while varying the cost of rejection of cars type 1.**