POLITECNICO
DI TORINO

Master double degree course in
Computer Engineering

Master Degree Thesis

Explainable Reinforcement Learning for Risk Mitigation in Human-Robot Collaboration Scenarios

Supervisors
Prof. Paolo Garza
Prof. Iolanda Leite

Candidate
Alessandro Iucci

Ericsson Research AI Supervisors
Rafia Inam
Alberto Hata
Ahmad Terra

April 2021

Explainable Reinforcement Learning for Risk Mitigation in Human-Robot Collaboration Scenarios

ALESSANDRO IUCCI

Degree Programme in Computer Science and Engineering
Date: April 5, 2021
Supervisor: Paolo Garza, Iolanda Leite
Industrial Supervisor: Rafia Inam, Alberto Hata, Ahmad Ishtar Terra
Examiner: Danica Kragic Jensfelt
School of Electrical Engineering and Computer Science
Host company: Ericsson AB
Swedish title: Förklarbar förstärkningsinlärning inom människa-robot sammarbete för riskreducering

Explainable Reinforcement Learning for Risk Mitigation in Human-Robot Collaboration Scenarios / Förklarbar förstärkningsinlärning inom människa-robot sammarbete för riskreducering

Abstract

Reinforcement Learning (RL) algorithms are highly popular in the robotics field to solve complex problems, learn from dynamic environments and generate optimal outcomes. However, one of the main limitations of RL is the lack of model transparency. This includes the inability to provide explanations of why the output was generated. The explainability becomes even more crucial when RL outputs influence human decisions, such as in Human-Robot Collaboration (HRC) scenarios, where safety requirements should be met.

This work focuses on the application of two explainability techniques, "Reward Decomposition" and "Autonomous Policy Explanation", on a RL algorithm which is the core of a risk mitigation module for robots' operation in a collaborative automated warehouse scenario. The "Reward Decomposition" gives an insight into the factors that impacted the robot's choice by decomposing the reward function into sub-functions. It also allows creating Minimal Sufficient Explanation (MSX), sets of relevant reasons for each decision taken during the robot's operation. The second applied technique, "Autonomous Policy Explanation", provides a global overview of the robot's behavior by answering queries asked by human users. It also provides insights into the decision guidelines embedded in the robot's policy. Since the synthesis of the policy descriptions and the queries' answers are in natural language, this tool facilitates algorithm diagnosis even by non-expert users.

The results proved that there is an improvement in the RL algorithm which now chooses more evenly distributed actions and a full policy to the robot's decisions is produced which is for the most part aligned with the expectations. The work provides an analysis of the results of the application of both techniques which both led to increased transparency of the robot's decision process. These explainability methods not only built trust in the robot's choices, which proved to be among the optimal ones in most of the cases but also made it possible to find weaknesses in the robot's policy, making them a tool helpful for debugging purposes.

Keywords

Explainable Reinforcement Learning, Human-Robot Collaboration, Risk Mitigation, Reward Decomposition, Autonomous Policy Explanation, Collaborative Robots

Sammanfattning

Algoritmer för förstärkningsinlärning (RL-algoritmer) är mycket populära inom robotikområdet för att lösa komplexa problem, att lära sig av dynamiska miljöer och att generera optimala resultat. En av de viktigaste begränsningarna för RL är dock bristen på modellens transparens. Detta inkluderar den oförmåga att förklara bakomliggande process (algoritm eller modell) som genererade ett visst returvärde. Förklarbarheten blir ännu viktigare när resultatet från en RL-algoritm påverkar mänskliga beslut, till exempel i HRC-scenarier där säkerhetskrav bör uppfyllas.

Detta arbete fokuserar på användningen av två förklarbarhetstekniker, "Reward Decomposition" och "Autonomous policy Explanation", tillämpat på en RL-algoritm som är kärnan i en riskreduceringsmodul för drift av samarbetande robotars på ett automatiserat lager. "Reward Decomposition" ger en inblick i vilka faktorer som påverkade robotens val genom att bryta ner belöningsfunktionen i mindre funktioner. Det gör det också möjligt att formulera en MSX (minimal sufficient explanation), uppsättning av relevanta skäl för varje beslut som har fattas under robotens drift. Den andra tillämpade tekniken, "Autonomous Policy Explanation", ger en generellt prespektiv över robotens beteende genom att mänskliga användare får ställa frågor till roboten. Detta ger även insikt i de beslutsriktlinjer som är inbäddade i robotens policy. Ty syntesen av policybeskrivningarna och frågornas svar är naturligt språk underlättar detta en algoritmdiagnos även för icke-expertanvändare.

Resultaten visade att det finns en förbättring av RL-algoritmen som nu väljer mer jämnt fördelade åtgärder. Dessutom produceras en fullständig policy för robotens beslut som för det mesta är anpassad till förväntningarna. Rapporten ger en analys av resultaten av tillämpningen av båda teknikerna, som visade att båda ledde till ökad transparens i robotens beslutsprocess. Förklaringsmetoderna gav inte bara förtroende för robotens val, vilket visade sig vara bland de optimala i de flesta fall, utan gjorde det också möjligt att hitta svagheter i robotens policy, vilket gjorde dem till ett verktyg som är användbart för felsökningsändamål.

Nyckelord

Förklarbar förstärkningslärande, Mänskligt-robot-samarbete, Riskreducering, Reward Decomposition, Autonomous Policy Explanation, Samarbetsrobotar

Acknowledgments

I would like to express my uttermost gratitude to my supervisors Rafia Inam, Alberto Hata and Ahmad Terra from Ericsson for their persistent guidance and help. I acknowledge the great effort that they have given to my thesis work.
In the same way, I would like to thank professors Iolanda Leite and Paolo Garza for their extensive supervision and for being always available to answer my questions.

This thesis, more than just representing a project I have been working on, comes as a symbol representing the end of a journey. We could consider this journey to have started when I was 6 years old, but I won't go that back in time and I will limit the focus to the last two and a half years.
There are things to learn in life: some of them are learned studying on books, or with someone teaching you in school, but some others are learned the hard way, with hands-on experience, and life itself becomes the only teacher you can blame for not preparing you enough.
An enemy I thought I already defeated in the previous levels of this long game called life is loneliness. But just like Majin Buu keeps coming back with a different mutation every time we think Goku & friends finally defeated him (weird example but it's the only one I could think of), we must accept that sometimes villains do come back in a different form and we must understand how to handle it or eventually get defeated.
I had to learn loneliness, and I'm most definitely still learning it. I learned that it comes when you are locked up alone in your Björksätra room for three months because there is a pandemic messing up the world outside, but I also learned that it hits way harder when you're in the middle of a dancefloor full of people having the time of their lives, and you just don't seem to fit in.

And just like we can't learn light without darkness, just like we can't learn happiness without sadness, just like we can't learn peace without war, I couldn't learn loneliness without the incredible people I met in the process. I will not go ahead and formulate kind words for every person because I am intrinsically incapable of being kind and I would end up choosing horrible sentences from "quotes.net" which will sound just like the movie you've seen 1000 times before. But I still want to keep some sort of traditional format to this section and at least mention (in a completely random order, or if anything just resembling chronological order) groups of people, mostly using the names of the WhatsApp chats we have in common. So let's just go ahead and say

thank you to my little drama club with the connected “giocatori anonimi”, to “L’associazione”, to the “Suore” and to my hometown “AMIGHI”. Thanks to the “Collegio” people (whichever floor they belong to). Thanks to “ESN Turin” and to all the people who came and went from MAIN and International Reception. Thanks to the Bros. Thanks to “Ghibli” cause that will unavoidably be a part of me, no matter what, and thanks to the people who bear the burden of my complaints on a daily basis, starting from the ones who convinced me to go to the gym to the people who still second my nonsense addiction to board games.

At the end of the saga, Majin Buu actually becomes a precious ally, and I firmly believe that in the same way, loneliness is not just a villain. If we embrace it, there are times when loneliness is the only one we want to keep us company. We must then distinguish “loneliness” to “feeling lonely”, which is just a version upgraded with boredom, with that feeling of actually wanting to be around people and have fun, and if I learned this is just by missing (or not) the people mentioned above.

Last but not least, I would like to spend some words to thank my family, which is a complete novelty for me since we are not very expansive people. In the years, I have seen many who usually thank their families for their unconditional support throughout their studies, but I strongly believe that it is easy to support someone when you simply agree with their choices. This task becomes way harder when your decisions are different from what they would have expected them to be. I got lucky because, one way or another, they allowed me to always do what I felt was best for me, independently from whether they actually approved it or not, and this kind of trust is something way more valuable than unconditional support. So thanks to my father, my sister, my grandma and my great-aunt for always caring. And lastly thanks to my mother from whom I inherited the quality to be stubborn enough to always stand up for myself.

Stockholm, April 2021
Alessandro Iucci

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

AI Artificial Intelligence

BDI Behavioural Divergences Identification

CNN Convolutional Neural Network

DASRI Dominant-Action State Region Identification

DNF Disjunctive Normal Form

DQN Deep Q-Network

FCN Fully-Connected Network

FOV Field of View

HRA Hybrid Reward Architecture

HRC Human-Robot Collaboration

MDP Markov Decision Process

ML Machine Learning

MLP Multi-Layer Perceptron

MSX Minimal Sufficient Explanation

RDX Reward Difference Explanation

RL Reinforcement Learning

ROS Robot Operating System

SBC Situational Behavior Characterization

SRLC State Region to Language Conversion

XAI Explainable Artificial Intelligence

XRL Explainable Reinforcement Learning

Chapter 1

Introduction

The technological improvements that have been developed during the years allow us nowadays to have working environments where humans and robots collaborate to perform tasks in the so-called Human Robot Collaboration HRC, one of the fundamental elements in industry 4.0.

The reason why such collaboration is advisable is that humans and robots have relevant behavioral differences which make one prevail over the other under different circumstances. For example, while humans are smarter and more flexible both in their thinking and movements compared to robots, the robots have some unique characteristics like higher speed, power, and precision as well as being indefatigable.

However, while such collaboration is undoubtedly beneficial, sharing the same workplace leads to interactions between robots and humans that, if not controlled and managed correctly, can harm both of these two kinds of workers. This raises several challenges, especially when we think about the safety issues that may occur. For this reason, having a risk management strategy in place for the robots' actions becomes crucial to grant a safe collaborative environment.

This thesis is an extension of the work by Inam et al. [5, 6], which aimed to produce a risk identification and evaluation strategy as well as safety assessment in an automated warehouse with collaborative robots, and Terra et al. [4], which built AI algorithms for risk mitigation to increase the safety of the operations. Multiple algorithms based on two techniques were tested and compared in [4]: one is based on a fuzzy logic system and several others based on reinforcement learning but with different architectures.

This work focuses on extending the reinforcement learning implementation by adding explainability. One of the biggest disadvantages of reinforcement

learning algorithms is that they are often used as a black box. This is a consequence of their complex architecture, usually a neural network with a considerable amount of variables, which makes it hard not only to understand what the algorithm is doing and what did it learn but also to spot aberrant behaviors and debugging them. This is why explainability becomes a fundamental requirement to build trust between the system and the users, but also provide a powerful debugging tool for unexpected behaviors. This goal becomes even more important when it comes to delegating the safety of the operations to a machine learning system.

1.1 Objective of the thesis

The research question that will be answered in this thesis is: *How can the RL algorithm used to implement risk mitigation for HRC scenarios be made more explainable and interpretable?*

To achieve this goal, Explainable Reinforcement Learning (XRL) is used in this work, which is experiencing huge growth in the past few years because there is a raising need for transparent algorithms [2], so that it is possible to fully understand their operation, averting the risk of using them as a black box system.

Several different methods can be used to make a process more transparent and this work will focus on two different techniques defined as "Reward Decomposition" [3] and "Autonomous Policy Explanation" [1].

The main feature of the reward decomposition [3] method is that it embeds explainability in the learning process by dividing the typical reinforcement learning reward into different types of partial rewards, each one depending on some aspects of the environment and/or the robot operations. Some examples can be the proximity of the robot to an obstacle, the robot's linear speed, the rotational speed, direction, and more. The reason why this makes it all more interpretable is that this reward classification provides, via graphical and textual representations, some insight into the robot's operation:

- It can be used to analyze why the robot chose a certain action and if it was the right one in that situation.
- It becomes possible to understand which factors impacted the most the robot's action choice.
- For each decision, it becomes clearer which types of reward, among those that make up the decomposed reward, are actually relevant at that

moment in time and which are, on the other hand, negligible.

- The division of the learning process by branching the single learning neural network into multiple parallel ones, one for each type of reward, allows each of them to focus on a single aspect, therefore speeding up the learning process.

The advantage of the autonomous policy explanation, on the other hand, is to have a global explanation of what has been learned by the robot during the training process. The most considerable feature of this method is that it gives an explanation of the policy in the form of human readable text so that even non-expert users can be able to easily understand it asking question focused on what they want to analyze and getting answers from the algorithm. This helps to build trust between human users and the robot and it allows a better collaboration.

Having the full explanation of the policy learned by the robot in the form of a text makes it possible to compare the user expectation to the agent learning process so that faulty behaviors are easily detected.

To use such techniques on the previously developed algorithm, some improvements in the previous version of the algorithm were also needed because the choices made by the robot did not look very dynamic. Indeed, the robot tended to adopt an overly safe approach by constantly slowing down the speed and therefore compromising the efficiency of the operation.

1.2 Scope and Limitations

This degree project is the continuation of the work of [5], [6] and [4] * whose aim is to create risk management strategies for an automated warehouse scenario. The scene graph generator, the risk analysis components, and the risk assessment module have been already implemented in [6, 5] and serve as input for the risk mitigation algorithm. The risk mitigation algorithm has already been implemented in [4] but substantial changes have been made during the development of this project, mainly focused to get the algorithms to make smarter decisions. The low-level implementation of the navigation module of the robot is outside the scope of this thesis as it is already provided in a third-party library used in the project.

* More information can be found at https://github.com/EricssonResearch/scott-eu.

1.3 Ethical and Societal Aspects

Given the purpose of this thesis work, it is relevant to focus on the impact that it has from an ethical and societal point of view.

Indeed, when delegating safety aspects to algorithms, we must take into account the implication of this choice. In this case, without a proper risk mitigation strategy, the robot may harm humans or objects during its operation causing physical and financial damage and the question would come down to whose responsibility would it be. Making sure that the risk mitigation module is actually behaving as we would expect is the first step to guarantee safety and build trust in the users about the algorithm itself. This is one of the main reasons for the application of the explainability techniques illustrated in this work. Transparency in such Artificial Intelligence (AI) algorithms becomes a needed requirement to ensure an ethical approach and make them accepted by the scientific society.

HRC is also a very precious tool in industry 4.0 because it allows to automatize repetitive tasks delegating them to robotic components making the whole work chain more efficient. It is also relevant to notice that such repetitive tasks have a negative impact when it comes to the employee experience and automating them could help to make it more stimulating.

All of the results and theories presented in this work have been backed-up with scientific evidence to guarantee an ethical research approach. All of the code is available https://github.com/EricssonResearch/scott-eu

1.4 Structure of the thesis

Chapter 2 presents relevant background information about: the need for explainable techniques; the different approaches that can be taken; and the reasons why reward decomposition and autonomous policy explanation were chosen. In addition, this chapter briefly discusses the previous projects which formed the starting point for this thesis.

Chapter 3 presents the methods used to solve the problem and provide interpretability. This chapter also discusses the changes needed from the previous version of the reinforcement learning algorithm.

Chapter 4 discusses the obtained results and the experiments that have been carried on during the project. Lastly, the Chapter 5 will provide the conclusions about the whole work and future improvements that could be implemented.

Chapter 2

Related Work and State of the Art

2.1 Background on Explainable Reinforcement Learning

This section examines why explainability in AI and more specifically in RL is increasing in importance lately and explores taxonomy of the possible different classes of methods that allow improving interpretability focusing on the ones chosen to be implemented on the current project.

2.1.1 Reinforcement Learning

Reinforcement Learning (RL) is a technique that consists of an autonomous agent that learns by trial-and-error to find the optimal solution to a problem. The learning process starts with the agent performing randomly chosen actions so that the algorithm can explore all the possible choices in several different scenarios. The choice will therefore impact the subsequent state and, based on how desirable the reached state is, the agent receives a reward which is set by the designer of the algorithm (see Figure 2.1) [2]. The more the training goes on, the more the robot will start using what it learned instead of randomly chosen actions, because the probability of choosing random actions instead of relying on the algorithm slowly decays with the iterations. Since the agent will have as a goal to maximize the reward, the agent will gradually learn to avoid disadvantageous states to prioritize the high reward ones. The algorithm will therefore come up with a learned policy that will aim to achieve the highest possible cumulative reward and select the best actions in every situation. If

the next state depends only on the current state and the performed action but not on past states, the RL algorithm satisfies the Markov property and it is therefore called a Markov Decision Process (MDP).

Deep RL is a variant of simple RL where, given the high complexity of the space of environment states, the table that is used in RL to store all the possible rewards for each state is replaced by a neural network which approximates the function of the table. More details will be given in Section 3.2.1

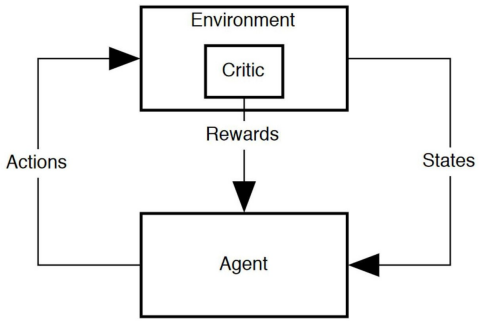


Figure 2.1 – Reinforcement Learning Process. Based on the environment state, the agent chooses actions which will change the state of the environment, and depending on the reached state, the agent receives a reward feedback. Diagram taken from [2].

2.1.2 The reasons for explainability

As previously discussed in Chapter 1, Machine Learning has allowed us a great technological improvement, but it presents a fundamental drawback: it is often used as a black box. Indeed, the more powerful and flexible these systems become, the more transparency on the decision process is affected. This trade-off is often referred to as readability-performance trade-off [2]. But why is it so important to explain why the agent takes some decisions? The first reason is psychological: the more we can understand the decisions taken from an agent, the more we can trust the model we built, and the users will be more inclined to accept the model. Transparency also ensures that the decisions of the agent are fair and ethical, therefore, to use a system with confidence, it needs to be trusted and consequently, it needs to be transparent so that its decisions can be justifiable. It is also important to consider the legal aspects related to systems that have a risk component like the one we have under analysis in this work.

There are also relevant practical aspects because even if the efficiency of Machine Learning (ML) based systems increases more and more, the lack of interpretability of the decision process reduces its usefulness: the effectiveness of a system does not exclude that the system may be faulty and considering how AI systems are becoming more and more autonomous, transparency becomes fundamental, especially when an agent learns by itself like in RL.

Last but not least, RL models are difficult to debug for developers because they rely on several different aspects: the environment and the design of the reward function, the algorithm used to train the policy, the encoding of the observations, and the neural network architecture. An improved explainability would therefore help speed up the resolution of issues and at the same time speed up development in RL methods. [7]

2.1.3 Explainable Artificial Intelligence Taxonomy

Explainable Artificial Intelligence (XAI) methods can be divided into categories mainly based on two aspects. The first division depends on at what time the needed information is extracted from the model. This division characterizes a method as a transparent model (or intrinsic model) [7] if the explanation comes directly from how the model was built, meaning that they have a transparent architecture that makes them explainable without the need of any external post analysis and it can be somehow considered self-explanatory.

Post-hoc explanation models [2], on the other hand, are characterized by creating a secondary explanatory model after the training of the primary model that can provide some insight into the decision process. To put it in simpler words, such a model can be considered an additional tool that is added to our black box which can upgrade it allowing it to be more transparent.

We must still keep in mind the existence of the readability-performance trade-off, which applies also when choosing one or the other approach: indeed, transparent models usually have a simpler architecture and this lowers the overall performance of the model, offering as a payback very good explanations; by contrast, post-hoc models usually do not affect at all the accuracy of the original opaque model, but it becomes considerably harder to retrieve satisfying explanations.

The second differentiation between XRL models comes from the different scope of the explanation which can be either global or local [2]: while global models try to explain the general behavior of the model, trying to understand which is the logic behind its decisions, the local models aim to explore the reasoning behind a specific instance of the process or a set of related instances.

Consequently, global interpretability will increase the level of confidence in the whole model, while local techniques will lead the user to trust a specific prediction.

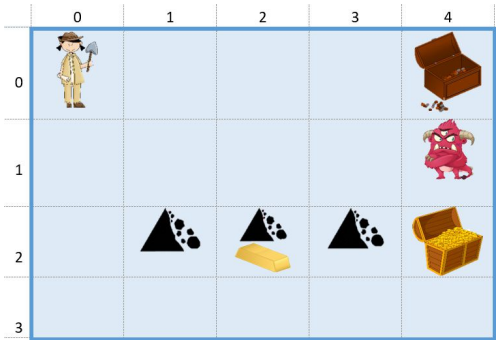


Figure 2.2 – CliffWorld application of reward decomposition. Each element in the grid represent a type of reward with associated values: cliffs are -10 points, empty treasure chest is 1, monster is -2, gold bar is 10, and full treasure chest is 15. Figure taken from [3].

2.1.4 Reward Decomposition

One of the explainability methods suitable for this risk mitigation project is RL via reward decomposition [3], whose main principle is to divide the reward given by the reward function in RL in a sum of meaningful different reward types. It consists of using a Deep Q-Network (DQN) which is modified so that a vector-valued reward function is defined where each component represents the reward for a certain type. Following the same philosophy, the Q-function is also decomposed in the same way and each component corresponds to action values that only consider a single reward type. The overall reward can be computed summing all the partial rewards, but the decomposition allows the model to focus on the best policy providing at the same time an explanation, making clear which of the reward types did the algorithm consider more important when choosing that action.

There are two kinds of explanations that can be extracted from this model:

- Reward Difference Explanation (RDX): it is useful to understand whether and why an action had an advantage or disadvantage over another action;
- MSX: it allows the users to identify reduced sets of the most relevant reward types that lead the agent to take a certain decision, at the same

time identifying the negligible components of the overall reward.

These two types of explanations are complementary because, while RDX provides an understanding of the choice made between several alternatives, MSX sets are useful to select the most relevant reward decomposition among the possible ones. The mathematical principles behind these two explanation methods are presented in detail in Section 3.3.1, while results and examples on our scenario are presented in Chapter 4

In the paper by Juozapaitis et al. [3], this approach is implemented and tested in two applications.

- CliffWorld, a grid-world where each cell can contain cliffs, monsters, gold bars, or the treasure. The possible actions of the agent are to move to north, south, west, or east. To each of the possible elements that are in the grid, corresponds a certain reward as explained in Figure 2.2. The goal of this game, which is one of the most famous basic experiments for RL, is to make the character learn by itself how to reach the treasure and avoid each penalizing element like the monsters or the cliffs.

- Lunar Lander: a game where the agent needs to control a rocket that should land safely on the moon ground. The possible actions are to fire one of the three engines (main, left, or right) or to perform no action. The decomposition of the rewards is made considering fuel cost, safe landing, crashing penalties, or obstacle avoidance. Each of the episodes ends with either a safe landing or crashing and then the game is reset. The agent learns by itself through RL how to control the rocket and which actions are convenient to ensure safe landing and avoid possible obstacles on the ground. This is another example of a classic simple RL game.

Both these applications represent a simpler scenario compared to our risk mitigation and collision avoidance module because the possible actions are restricted to a smaller set of possible choices (while, as we will see in Chapter Chapter 3, we will have 12 possible actions), there are well defined possible rewards and the two presented applications can present a much smaller number of possible environmental states.

2.1.5 Autonomous Policy Explanation

Autonomous Policy Explanation is a method elaborated by Hayes and Shah [1] that allows using algorithms and other side systems to synthesize human-readable policy descriptions. The users can therefore analyze the behavior

of the system by asking the robot targeted structured queries. This type of explanation is provably effective on several RL applications including deep reinforcement learning with Q-function approximation, and allows fault diagnosis also by non-expert users.

This tool does not only provide a way to verify shared expectations between a human and its human co-workers, but it is also powerful to debug complex systems because the generated explanations can identify undesired or misapplied behaviors.

This approach consists, as the first step, to learn a domain model of the system operations from real or simulated demonstrations and, using those statistics, building a behavioral model that approximates the robot's logic.

Using Boolean algebra over predicates to encode state regions in natural language, it is possible to create four algorithms that enable an agent to answer behavior-related questions. What these algorithms make possible is to:

- Identify environmental conditions under which the robot will perform certain actions by answering the question "When do you __?"
- Identify which actions will the robot perform under a specified set of environmental conditions by answering the question "What do you do when __?"
- Create an explanation for why a specified action did not occur by answering the question "Why did not you __?"

The Autonomous Policy Explanation method has proved to be successful in all the three applications tested in the paper [1] which were: a) a delivery task where a robot must pick and deliver parts avoiding human-only areas in a GridWorld (a 2D rectangular grid with an agent starting at one grid square and trying to move to another grid square located elsewhere. The agent is only allowed actions of moving in up, down, left, right directions by 1 grid square) b) a stabilization task where a robot must stabilize a pole placed on a mobile cart and c) a parts inspection task, where a robot controlled by a stock feed signal must locate and inspect parts on a conveyor belt.

2.2 Background on Risk Mitigation

This section discusses the risk management framework which this work is based on.

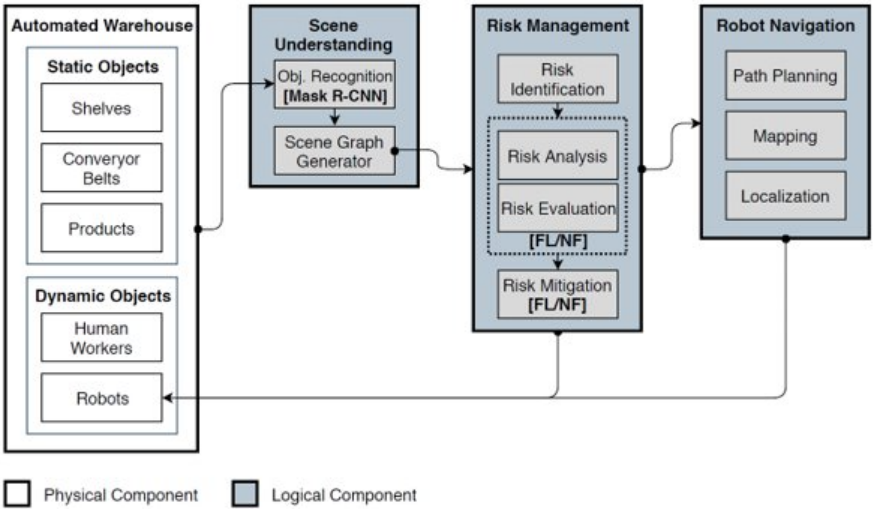


Figure 2.3 – Complete process for Risk Management.

2.2.1 Risk Management

Inam et al. [6] [8] implemented a risk management system for an automated warehouse scenario according to the specification in ISO 31000:2018: *Risk Management Guidelines*. The complete process can be seen in Figure 2.3 and consists of 4 primary phases:

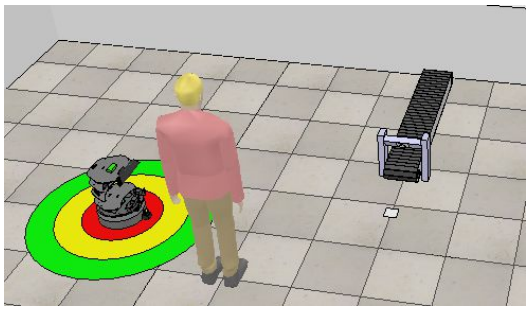
1. Hazard and Risk identification: The first step is to identify all the possible threats in a HRC scenario, considering all the possible consequences that could harm humans or damage other objects, and to do so, the HAZOP (HAZard Operability) [9] comes handy because it was developed specifically for human-robot collaboration scenarios. After modeling these scenarios by use case, sequence, and state-machine diagrams, it is possible to generate the possible deviations of the system to finally remove meaningless or redundant deviations to keep only relevant hazards

2. Risk Analysis: the robot needs an understanding of the surrounding environment and uses the scene graph built thanks to its sensors [10]. This type of representation is chosen because it considers and encapsulates the relevant semantic relationships between the elements in the environment other than the type, distance, direction, and speed of them.

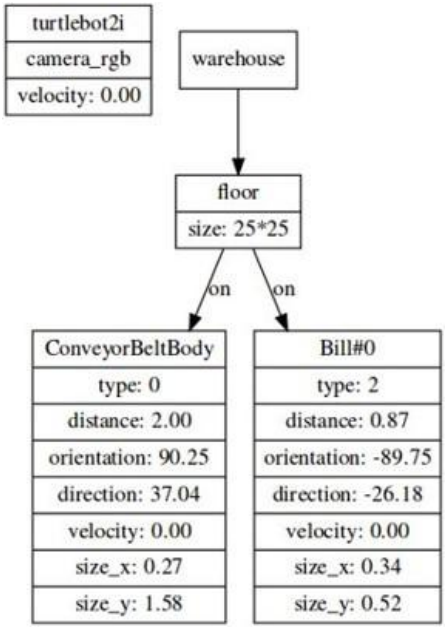
3. Risk Evaluation: this module computes the potential hazard that could harm the robot or the obstacles. Based on this, different concentric

safety zones are created around the robot depending on the previous analysis. The safety zones are: critical when an obstacle is very close characterized by the color red, warning when the obstacle is close characterized by the color yellow, and safe when the obstacle is far characterized by the color green.

4. Risk Mitigation: this last module, whose implementation will be better discussed in the Section 3.2, should be able to use the risk levels and perform actions to maximize the safety in situations where it is relevant.



(a) Scenario with the robot meeting with a dynamic obstacle (human) and a static one (conveyor belt).



(b) Scene Graph generated according to the situation presented in Figure 2.4a

Figure 2.4 – Scene Graph Generation Example. Figures taken from [4].

A practical example can be seen in Figure 2.4 where a robot finds in its path and Field of View (FOV) one walking human and a conveyor belt: a static and a dynamic obstacle. The situation is presented in Figure 2.4a and the reconstruction made by the scene graph is presented in Figure 2.4b. The most important information extracted from the scene graph representation is the risk value assigned to the two obstacles: indeed the obstacles are characterized by a risk value that goes from a minimum of 0 to a maximum of 4 and a

classification of the type of obstacle (static, dynamic, human). The value assigned to the conveyor belt is 0, the minimum, because it is a static obstacle and the value assigned to the human is 2, high but not the highest probably because the human is still and does not move, but potentially it could in an unpredictable way.

2.2.2 Artificial Intelligence for Risk Mitigation

As anticipated in the previous subsection, after performing the risk assessment and scene understanding, there needs to be a module that uses those representations to ensure a safer operation of the robot. This risk mitigation module can be implemented using AI because, thanks to machine learning, it is possible to make the robot learn a complex decision-making process by itself in a short time. Terra et al.[4] took care of implementing this module using two different approaches: a fuzzy logic system and a reinforcement learning-based algorithm.

Since the fuzzy logic system is self-explanatory and interpretable, there is no need to implement any kind of explainability in that case. However, the great disadvantage is that all the fuzzy logic rules have to be generated manually, which is a laborious process. The solution is to use the RL based on Q-Learning, which seems to be promising and more adaptable, but it serves the risk mitigation purpose as a black box. It seems therefore advisable to apply some explainability techniques there could not only make the algorithm more transparent but also help debug and improve the results.

This work focuses on the Multi-Layer Perceptron (MLP) implementation, without digging in more complex versions of the RL algorithm (i.e. the one based on the Convolutional Neural Network (CNN) or the hybrid one) because this makes the training faster and the interpretability easier to achieve, while the transposition of these methods on more complex architectures can be left to a further research phase.

2.3 Computation Architecture

The RL algorithm presented in the work of Terra et al.[4] showed to demand high computational power, mostly because it needs the information coming from the scene graph generator, which can be considered the bottleneck of the process. This becomes a problem if we consider that the robot has a constrained processing unit and it is difficult to scale up without over-complicating the computation process by making more processors in a cluster

run concurrently. Another limitation to be considered is that the robot has to carry the computation unit along with its power supply, and this would physically make the robot difficult to handle from a hardware point of view, becoming it heavier and far more complex.

The solution to this architectural problem is to decentralize the process, offloading the heavier computations on more powerful systems to reduce the latency of response and keep it within a reasonable delay interval and this problem has been solved on a parallel thesis work.

2.3.1 ROS

ROS is an open-source robot operating system that provides the tools to establish a structured communication protocol using a modular structure. This framework allows having multiple processes running on the same host but also multiple hosts communicating using a peer-to-peer topology.

The basic idea of ROS is to decentralize the processes by creating multiple *nodes*, each one delegated to a specific task or process e.g. one node to manage input sensor, one for the navigation, one for the motors control. All of these nodes register themselves in a centralized unit called *master* which works as a node discovery system so that the nodes can find one another and communicate peer-to-peer. The master knows which nodes should be involved in a certain communication, and therefore can send the right lists to the requesting nodes because each of them can subscribe to the *topics* they are interested in receiving updates about, and each message always has the indication of the belonging topic(s) [11]. To give a practical example, “Node A” can register to the master, marking the topics it wants to be updated about the “Sensor Input”. When , registered to the master, requests the list of nodes interested in “Sensor Input” topic, he gets in the list also “Node A” and from that moment on, “Node A” and “Node B” can exchange messages peer-to-peer about the “Sensor Input” topic

In this project, every module shown in Figure 2.3 was implemented as a ROS node so, for example, the risk mitigation node receives information from the risk evaluation node and computes the action to be taken, which is sent to the nodes that take care of the robot speed and navigation.

2.3.2 Final remarks

All of the above explainability methods will be adapted with slight modifications and integrated into the scenario of an automated warehouse, which is a more

complex environment than the ones tested in the cited researches [3] [1]. This thesis will therefore contribute to improving the risk mitigation module applied to the robot's navigation, highlighting undesirable behaviors so that they can be fixed, but also providing an explanation of what the robot learned.

Chapter 3

Methods

This chapter discusses the technical implementation details of the risk mitigation and explainability modules in this master thesis. It will start outlining the simulation environment and details of its design in Section 3.1, while the design of the reinforcement learning algorithm and the changes made to the original algorithm will be discussed in Section 3.2. For the explainability part, the reward decomposition approach is introduced in Section 3.3, and the principles and implementation behind autonomous policy explanation will be described in Section 3.4.

3.1 Implementation Design

In this section the robot, the simulated environment, and how the risk mitigation module influences the navigation will be discussed.

3.1.1 Robot Control System

The robot used for this research project is of the model Turtlebot 2i*. This robot is relatively low cost and has a manipulator arm that can be used for material transportation inside the warehouse environment.

Turtlebot 2i is a differential-drive robot, which means that the navigation in the environment is done by controlling independently the two wheels, left and right. Moreover, for this robot, a navigation module is already present, which allows the robot to move into the warehouse environment by using the occupancy grid map. This module works thanks to the trajectory planner,

* More details about this robot can be found at https://www.trossenrobotics.com/interbotix-turtlebot-2i-mobile-ros-platform.aspx

which given the map and a goal, can find the optimal path to reach the goal from the robot position, and a local planner which controls the speed of the robot while it follows the path.

The robot also has a laser scanner, which is useful to locate obstacles so that the planners can recompute a different path to avoid the obstacle. To minimize the modification to the original implementation, the risk management modules are built on top of this system. Specifically, the risk mitigation module, indeed, outputs a speed scale for each wheel to be applied to the command produced by the navigation module. The control diagram of the robot is shown in **??**.

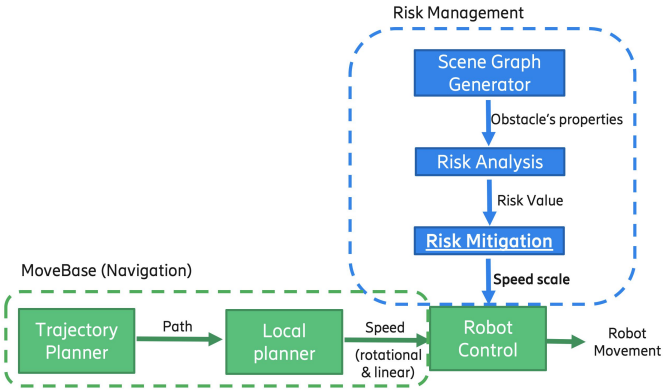


Figure 3.1 – Control Diagram of the Robot Operation taken from [4].

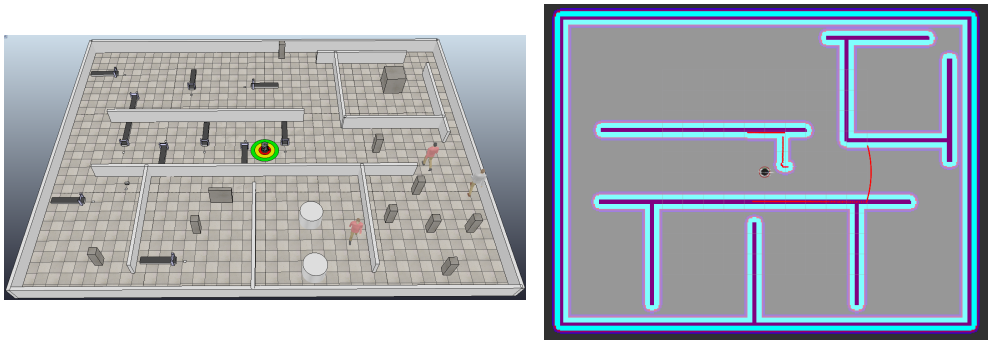
3.1.2 Simulated Scenario

Testing the robot's algorithm in the simulated environment can speed up the development process by dispensing physical setup and avoid damages or accidents. This is the reason why, continuing with the same strategy of the previous works [4] [6], V-REP was chosen, a software that allows to create a simulated warehouse and to add many elements such as conveyor belts, boxes, product, shelves and dynamic objects like humans.

The navigation module also gets as input an occupancy map that contains all static known obstacles, mostly just walls, so that the robot can find a path avoiding them to get to the goal. All extra obstacles besides the walls, which can be humans or objects placed by humans in the warehouse, are considered unknown and they are locally detected by the robot and taken into account by the scene graph generator as soon as the robot moves close enough to identify them with its detection equipment.

The robot can get information about its surrounding from a laser sensor placed on it called LIDAR. The data provided by this sensor takes only into account the distances of the objects but not the risk value assigned to each of them, and this is the reason why the sensor information needs to be combined with the scene graph generator information. At the same time, the scene graph needs the LIDAR information because it is more accurate than its monocular camera without any depth sensor.

During its operation, both training or when it already learned from the training phase, the navigation module is only aware of the walls of the environment, while all the other obstacles will be discovered gradually by the robot. The robot will therefore have to face every time different situations depending on the distance from the obstacles, the direction, the type of obstacle, the risk value, or its speed. During the training phase, the robot starts from the center of the warehouse and moves to a random position, and once it got there, it gets another goal to reach. The position is only reset in the occurrence of a collision with an obstacle. In Figure 3.2 there are the simulated warehouse environment with all the obstacles represented, and the occupancy grid map which is the starting knowledge for the navigation module.



(a) Simulated Warehouse with complete set of obstacles.

(b) Occupancy Grid Map

Figure 3.2 – Scene Graph Generation Example [4].

3.2 Risk Mitigation Algorithm

The risk mitigation algorithm is built on top of the navigation stack to reduce the potential danger of a human-robot collaboration scenario and ideally making it collision-free.

3.2.1 Deep Q-Learning

The technique used to build the RL algorithm is called Deep Q-Learning and it comes from an advanced version of Q-Learning. [12]

Q-Learning [12] is a RL algorithm that, given a state, measures how advantageous is each possible action. Indeed, Q stands for quality and the algorithm's goal is to choose the best action in each situation the agent is facing. To learn the best choices, at the beginning of its learning, the agent takes random actions and it uses a Q-table to record the state, the chosen action, and the corresponding reward given by a specific function coded by the developer. The choice of random actions characterizes the "exploring" phase of the algorithm.

After the exploration phase, in which the robot covers a variety of situations and records them in the Q-Table, it goes into the "exploiting" phase where it uses the data collected in the Q-table to ensure a good quality of the actions. The balance between these two phases is controlled by a parameter ϵ which represents the probability of choosing a random action against making a choice based on the Q-table, and it decays at each iteration of the algorithm.

Ensuring a good quality of the chosen action means maximizing the future reward. The issue with this approach is that it is required to record all possible combinations of states and actions and match them with their reward to have the certainty that the decision will be the best possible one. This implies exponential growth of the Q-Table as the number of states and actions increases and this is not sustainable in complex systems like the one we are considering.

To overcome this limitation, Deep Q-Learning [13] has the same approach in terms of functioning, but substitutes the Q-table with a neural network. The most relevant architectural changes that make the substitution possible are:

- Experience replay: a technique used to keep track of recent experiences of the agent, storing the state, the action, the reward, and the next state. During training, some experiences are drawn randomly from this set of saved ones and this allows the robot to learn not only from the latest conditions but also from past experience.

- Target network: a secondary network that has the same architecture as the primary one, but with fixed weights. It is periodically updated from the primary network during the training and it allows to compute the loss of the actions. It is useful to avoid unstable training scenarios.

3.2.2 Reinforcement Learning Algorithm

This section will be about some of the characteristics of the RL algorithm and the differences of this version with the previous one developed by Terra et al. [4] will.

The reason why such modifications proved necessary is that in the previous version, which is explained in detail in [4], the choices made by algorithm turned out to be not very dynamic and excessively cautious. This becomes a problem because if the actions chosen by the algorithm are not very dynamic, it means that it is not learning how to distinguish different situations and chooses instead always the overly safe option even when it is not convenient, and this compromises efficiency of the operations.

Actions

The purpose of the risk mitigation module is to reduce the risk in dangerous situations allowing the robot to work in safer conditions. The way this is achieved is by adjusting the robot's wheels' speed, applying individual speed scaling factors to the output speed of the navigation module. As an example, if the module outputs the couple [0.0, 0.4] it means that the left wheel will be stopped, while the right wheel speed will be multiplied by a factor of 0.4 with respect to the output speed produced by the navigation module.

Originally, the possible output actions of the module were four discrete speed scaling values, which would be applied independently for each of the two robot's wheels: the four values were 0.0, 0.4, 0.8, and 1.2. This resulted in 16 different outputs of the algorithm, considering all the possible combinations. This choice made possible the comparison with the fuzzy logic system [4], which had the same output set. However, when testing out this implementation, it turned out that the robot tended to prefer actions with the same speed scaling for both the wheels and to be very cautious constantly choosing to slow down the robot with speed scales of 0.0 (therefore stopping the robot) or 0.4.

Even when choosing the same speed scaling for both wheels, the robot does not always go straight because the navigation module is also interfering in the robot's control and makes it turn when necessary. It, therefore, seems that the risk mitigation module does not have much influence on the change of direction, and for this reason, in this new implementation, the outputs have been simplified only allowing a linear speed scaling, meaning forcing the same speed scaling for both wheels. Besides, the range of scaling was extended to

12 possible values:

$$0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.7 \quad 0.8 \quad 0.9 \quad 1.0 \quad 1.1 \quad 1.2 \quad 1.3 \quad 1.4$$

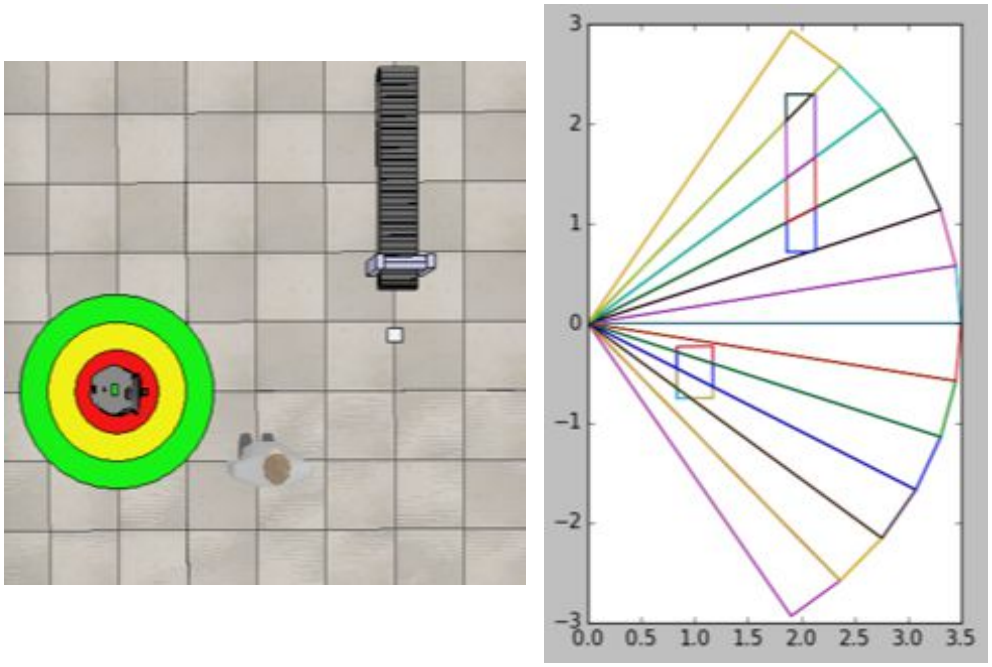
State

The action to be taken is chosen based on the situation that the robot is facing at each moment. Therefore, there needs to be a way to acquire the current state which serves as input to the network. While in the fuzzy logic implementation in [4] the focus was only on the nearest obstacle, the RL implementation considers all of the obstacles in front of the robot's FOV using the information obtained from scene understanding. and this makes There is still the need to convert the information of the scene graph into some easier representation such as a two-dimensional map. Since the robot has a FOV of $114°$ wide and a depth-sensing of 3.5 meters, it is possible to divide the field of view into 12 direction zones, getting for each one of the shortest distance to the closest obstacle. An example is shown in Figure 3.3. These 12 distances represent the first 12 inputs that are given the algorithm, but there additional ones that are added to provide more information:

13. Robot's linear translational speed
14. Robot's angular or rotational speed
15. The maximum risk value of the obstacles detected in the robot's field of view
16. Radius of the warning zone that is characterized by the color yellow
17. Radius of the safe zone that is characterized by the color green

In the previous implementation, the last two inputs of the state were generated by the safety analysis module, whose goal is to generate the radius of the different safety zones by evaluating several aspects such as the speed of the robot to generate three concentric round areas around the robot corresponding to different levels of safety, which can be seen in Figure 3.3a. In the current implementation, the three zones radiuses are not computed dynamically but have fixed values.

This choice was made because the reward function, which will be analyzed in Section 3.3, has different ways to compute the rewards depending on the zone where the nearest obstacle is placed. The dynamic computation of the radiuses reduces their size when the robot slows down, and the



(a) Top view of a possible scenario.

(b) 2D map of the FOV zone represented in Figure 3.3a with the detected obstacles.

Figure 3.3 – A possible scenario with obstacles and the respective translation into a 2D map. Figures taken from [4]

robot reasonably tends to slow down when getting closer to obstacles. The consequence of this behavior was that the obstacles were often in the safe zone or even outside it, and the robot could not learn properly how to deal with obstacles in critical and warning zone, and only focused on the most frequent situations, which were the safer ones, making the algorithm choose less appropriate actions.

To overcome this problem, additionally, to the fixed size safety zones, the risk mitigation module activation was restricted just to situations where there is actually a relevant risk. Indeed, as previously discussed in Section 2.2.1, this module should only influence operations in dangerous situations and do nothing otherwise. For this reason, its intervention is only requested when there is at least an obstacle in one of the three zones; if the obstacles detected by the scene graph are outside the safe zone, the risk mitigation module is disabled and lets the navigation module take over completely.

3.3 Reward Decomposition Principles

Reward decomposition is the first explainability technique implemented on this risk mitigation module, and it focuses on keeping the reward components divided by type and not having them mixed in only one lump-sum scalar reward.

This technique is inspired by other two:

- The explainability techniques that go under the name of hierarchical reinforcement learning which originated from temporal reward decomposition [14].
- If considered the different types of reward as agents, reward decomposition becomes just another interpretation of multi-agent reinforcement learning [15].

Architecture

This technique implies modifying the architecture of the model so that for each reward type, there is a distinct Q-Learning pipeline.

Terra et al. [4] compared three different Q-Network architectures: a fully connected network; a 1D convolutional neural network that convolves the input through the temporal dimension; and a hybrid network that combines both of previous networks. The experimental results showed that the hybrid model did not perform as well as expected, the CNN-based delivered safer actions with a significant reduction in efficiency, while the Fully-Connected Network (FCN)-based was more efficient but less safe.

From that, the simplification of the FCN architecture was approached instead of CNN, because it is easier for this kind of network to learn. Moreover, there is no evidence in the literature that how convolution through time could impact the effectiveness of the explanation produced by reward decomposition.

Reward Types

The reward decomposition technique requires finding different types of rewards (R) that can produce a meaningful categorization for what should be evaluated by the robot at each step. For this reason, the following decomposition into five types was produced:

1. *Obstacle reward*: focuses on the position of the nearest obstacle in the robot's FOV, giving penalties when the robot is getting too close to an obstacle.

Algorithm 1 : Decomposed Reward Q-Learning [3]

s_t: state at time t
a_t: action at time t
R: set of types of reward
Q^t: Q-function at time t
Q_r^t: component of type r of the Q-function at time t
$\vec{v}_t$: vectored reward at time t
$v_{t,r}$: component of type r of reward at time t
γ: discount factor
$Q \overset{\alpha}{\leftarrow} x$ is shorthand for $Q \leftarrow (1-\alpha)Q + \alpha x$

$s_0 \leftarrow$ Initial State
$a_0 \leftarrow \epsilon(Q^0, s_0)$
$t = 0$
repeat
  $(s_{t+1}, \vec{v}_t) \leftarrow \text{Act}(a_t)$
  $a_{t+1} \leftarrow \epsilon_t(Q^t, s_{t+1})$ # ϵ-greedy exploration
  for all $r \in R$ **do**
    $a' \leftarrow \arg\max_a \sum_r Q_r^t(s, a)$
    $Q_r^{t+1}(s_t, a_t) \overset{\alpha}{\leftarrow} v_{t,r} + \gamma Q_r^t(s_{t+1}, a')$
    $t \leftarrow t + 1$
  end for
until convergence

2. *Speed reward*: focuses on the linear speed of the robot, considering when it is coherent with the risk level of the situation. This means that the robot should not be too fast in dangerous situations, but it also should not be too slow when it is safe to keep a reasonable speed.
3. *Goal reward*: this reward type keeps the focus on whether the robot is moving towards the goal or if it is stuck or keeping a very cautious pace when not needed.
4. *Collision reward*: it only considers whether there has been a collision, giving a considerably negative reward in that case.
5. *Direction reward*: focuses on the direction of the rotational speed of the robot, taking into account where the closest obstacle is located and if the robot is turning towards it or away from it

Decomposed Reward Q-Learning

The classical Q-Learning algorithm needs to be adapted to support optimization with the reward decomposition approach, because in this case we do not have just one reward, but several sub-rewards. Juozapaitis et al. [3] created a version of the Q-Learning algorithm, giving in their paper proof of its convergence and soundness. The pseudo-code of the algorithm is shown in Algorithm 1. There have been several attempts to create algorithms that could support reward decomposition such as Hybrid Reward Architecture (HRA) [3]. Those algorithms had however the problem that, since there is a different Q-Network for each of the reward types, each of these branches of the network tended to the optimization of the single branch for the single reward type, without taking into account the influence of the other types. The reason for this new algorithm is ensuring the convergence to the overall optimal policy and not to the optimization of each reward sub-policy.

Reward Function

The reward function has been completely re-written compared to the implementation in [4] as now each type of reward should have its own reward function. This is easily achieved using a simple integer index that indicates which type of reward is being computed at that moment. The complete reward function can be found in Appendix A, and it takes mostly into account the nearest obstacle's distance, the speed of the robot, the distance traveled from the last computation, and the rotational speed of the robot.

3.3.1 Explainabilty with Reward Decomposition

After examining all the characteristics of reward decomposition, we can now use them to achieve interpretability for the decisions taken by the robot.

There are two kinds of explanation we can extract from this decomposition: RDX and MSX.

Reward Difference Explanations

This kind of explanation gives information to help us gain an understanding of why, being the robot in state s, the action a was preferred to all the other possible actions. It is possible to analyze all the rewards that the robot would have got with each of the 12 actions, and apply RDX in a pairwise fashion between actions a_1 and a_2. This means computing the differences between all

the reward types represented by all the elements of the decomposed Q-vectors: $\Delta(s, a_1, a_2) = \vec{Q}(s, a_1) - \vec{Q}(s, a_2)$. [3]

Each component can be analyzed to understand if action a_1 had an advantage or disadvantage over a_2 for that type of reward, and it can all be visualized as a bar chart as shown in Chapter 4.

Minimal Sufficient Explanations

When there are many reward types to consider, it is hard to detect which are the ones that most influenced the choice. For this reason, we can identify the minimal sufficient explanations MSX$^+$ and MSX$^-$, two smaller sets of relevant reasons for the choice. The union of these two sets is ideally smaller than the full RDX and represents a more compact explanation.

Let's define d, the disadvantage of a_1 over a_2 as $d = \sum_c I\left[\Delta_c(s, a_1, a_2)\right] \cdot |\Delta_c(s, a_1, a_2)|$ being I the identity function [3], which is basically the total magnitude of the differences for which a_2 is preferred, or also the module sum of all the negative values in RDX. Given d, MSX$^+$ is defined as the smallest cardinality set of postive reasons in RDX whose sum overcomes d :

$$\text{MSX}^+ = \arg\min_{M \in 2^C} |M| \quad \text{s.t.} \quad \sum_{c \in M} \Delta_c(s, a_1, a_2) > d$$

After defining MSX$^+$, we can define MSX$^-$ as the relevant disadvantages of a_1 over a_2, which means finding the subset of the rewards contributing to d which make all of the reasons in MSX$^+$ necessary. If we take the magnitude of all the advantages in MSX$^+$ and remove the smallest contribution, we have v:

$$v = \sum_{c \in \text{MSX}^+} \Delta_c(s, a_1, a_2) - \min_{c \in \text{MSX}^+} \Delta_c(s, a_1, a_2)$$

and we can define MSX$^-$ as:

$$\text{MSX}^- = \arg\min_{M \in 2^C} |M| \quad \text{s.t.} \quad \sum_{c \in M} -\Delta_c(s, a_1, a_2) > v$$

preferring larger disadvantages to smaller ones [3].

3.4 Autonomous Policy Explanation

Reward decomposition combines transparent architecture with a post-hoc analysis approach to explain locally the actions chosen. To have a rather global explanation understandable even by non-expert users instead, we can rely on "Autonomous Policy Explanation" which generates explanations in natural language. This section will discuss how the principles of this technique have been adapted to our scenario to create this kind of explanation.

3.4.1 Mapping into Natural Language

To formulate explanations in natural language we need to map the relevant input and output parameters into natural language categories, to reduce the state space that would otherwise be too large.

The input features are the distance and the position of the nearest obstacle and the robot speed. The only output is the robot speed scale. To keep a reasonable amount of states, the mapping was performed only on speed scaling, speed, and the nearest obstacle's distance, therefore ignoring its direction, which would have increased too much the complexity.

The mapping is as follows:

- *Nearest obstacle's distance*: the categorization was made based on the three safety zones around the robot, which as explained in the previous section, were kept with fixed radius.
  - *Critical*: Distance ≤ 0.295 m
  - *Warning*: 0.295 m $<$ Distance ≤ 0.622 m
  - *Safe*: Distance > 0.622 m
- *Robot's speed*: the robot speed has been categorized by taking into account that the minimum speed is 0.0, meaning that the robot cannot go backwards, and the theoretical maximum speed induced by the navigation module is 0.55 m/s.
  - *Slow*: Speed ≤ 0.2 m/s
  - *Medium*: 0.2 m/s $<$ Speed ≤ 0.4 m/s
  - *Fast*: Speed > 0.4 m/s
- *Speed scaling*: this categorization is just a four classes division of the possible output speed scaling of the risk mitigation module

- *Stop*: 0.0
- *Slow down*: 0.2 0.4 0.6 0.7 0.8
- *Keep the same speed*: 0.9 1.0 1.1
- *Speed up*: 1.2 1.3 1.4

Mapping into natural language is also used for the questions that can be asked and the answers, which are built using a series of templates and binary classifiers that will be discussed in the following subsections.

3.4.2 Behavioural Modelling

We use the MDP approach to build the domain and to extract a policy from the robot's behavior. Indeed, after a reasonable simulation time, we can extract a policy from the analysis of the robot state and actions and each time step by observing mainly two aspects:

- The transitions between states and their probability;
- The probabilities of every possible action class at each state.

The analysis of these data in our case scenario will be analyzed in Chapter 4.

3.4.3 Language grounding

As anticipated earlier, to create a human readable explanation, the information is transformed into natural language. To do so, two main components are used: communicable predicates and Disjunctive Normal Form (DNF) clauses.

Communicable Predicates

Communicable predicates are simply a way of encoding information about the state of the robot into a boolean feature vector with the DNF clauses [1].

In this case, the implementation of this technique requires to define the following six boolean predicates, each one with a function questioning the state of the robot:

1. IsObstacleCritical: true if the nearest obstacle to the robot is in the critical zone, false otherwise.
2. IsObstacleWarning: true if the nearest obstacle to the robot is in the warning zone, false otherwise.

3. IsObstacleSafe: true if the nearest obstacle to the robot is in the safe zone, false otherwise.

4. IsRobotSlow: true if the speed of the robot is classified as slow, false otherwise.

5. IsRobotAtMediumSpeed: true if the speed of the robot is classified as medium, false otherwise.

6. IsRobotFast: true if the speed of the robot is classified as fast, false otherwise.

Besides this function, identified by the field "verify" in the predicates, they also contain human readable sentences based on its boolean value, e.g. if the function in IsObstacleCritical turns out to be true, the correspondent sentence will be "the nearest obstacle is in the critical zone", otherwise "the nearest obstacle is not in the critical zone". The full example of the dictionary data structure connected to the predicate "IsObstacleCritical" is shown in 3.1.

$$\text{IsObstacleCritical} = \begin{cases} \text{true}: & \text{“the nearest obstacle is in the critical zone”} \\ \text{false}: & \text{“the nearest obstacle is not in the critical zone”} \\ \text{verify}: & \text{lambda s : s[“speed”] == “Fast”} \end{cases} \tag{3.1}$$

Disjunctive Normal Form Clauses

DNF clauses are the form used for representing states which are made of a combination of predicates into a Boolean vector representation.

Since there are six predicates, we can easily build a six elements vector where each of them is 1 if the correspondent predicate is true, and 0 if it is false. Since the first and last three predicates are mutually exclusive and they cannot coexist at the same time, at most one among the first three elements and one among the last three can be set to 1. One easy example is that the state of the robot having a fast speed and with the nearest obstacle at a critical distance will be encoded with the sequence $100\ 001$.

There is also the need for the Quine-McCluskey [16] algorithm to perform Boolean logic minimization on these Boolean formulas. This is necessary, as will be discussed in the next subsection, because state regions will need to be summarized in the most concise possible way to provide an effective

explanation. For example, if the detected regions are:

$$\begin{aligned} f(s) = & [\{\text{Distance: Critical, Speed: Slow}\}, \\ & \{\text{Distance: Critical, Speed: Medium}\}, \\ & \{\text{Distance: Critical, Speed: Fast}\}] \end{aligned}$$

corresponding to:

$$f(s) = [\{100\ 100\}, \{100\ 010\}, \{100\ 001\}]$$

needs to be summarized in the form: "100 - - -" where the symbol "-" means that that predicate is not relevant and can assume every value as long as it is a valid combination. $f(s)$ can therefore be summarized as "The nearest obstacle is in the critical zone".

3.4.4 Queries For Policy Explanation

We can now use all the tools mentioned above to ask questions to the robot and give us insight about under which conditions it will perform a certain action, or, alternatively, which actions will be performed under specified conditions according to the policy. It is also possible to gain an understanding of the differences between the actual robot behavior against the expected behavior.

To answer these questions, we need four algorithms, which have been adapted for this project from [17], and each of those will help answer a specific type of question. The pseudo-code for each of the algorithms can be found at Appendix B. To see how every question can be answered step-by-step it is very useful to have a concrete example to rely on: for this reason, a high-level insight on how each of the four algorithms works is provided with actual examples from our scenario in Section 4.2.

Identification of conditions for actions to take place

Identifying the conditions under which an action takes place means answering the question that corresponds to the template "When do you {action}?" e.g. in our scenario "When do you speed up?". To answer this question, we need to combine the action of two of the four algorithms.

The first step is to identify the state regions where the action specified is most likely to happen by finding all the target states and non-target states, and this is performed by the Dominant-Action State Region Identification (DASRI) algorithm.

After identifying the relevant state regions, the State Region to Language Conversion (SRLC) algorithm allows us to summarize the regions encoding them in Boolean expressions and using the Quine-McCluskey [16] algorithm to minimize the complexity of the answer.

It is therefore possible to generate a natural language response with the template "I perform {action} when {summary of target state region}".

Explanation of action against expectation

Sometimes there is the need to understand why at a certain time the robot did not behave as we would have expected. The Behavioural Divergences Identification (BDI) algorithm helps us answer the question that corresponds to the template "Why did not you {action}?" e.g. "Why did not you stop?". The algorithm can find the differences between the current state of the robot and the states where that action is supposed to happen according to the policy. This description is given by the template "I did not {action} because {difference between state the robot finds itself in and state regions where the action is supposed to be executed}. I {action} when {not performed action state region summary}."

Identification of situational behavior

This third question type can be considered the opposite of the first one because in this case, we extract using Situational Behavior Characterization (SBC) algorithm the policy strategy the robot would have in a given state. The question template is "What do you do when {state region description}" e.g. "What do you do when you have a fast speed?".

This is the most useful type of question to inspect if the policy learned by the robot matches our expectations, and the answer template will be similar to the one provided by the SRLC algorithm, "I perform {target action(s)} when {summary of defined state region}", with the difference that we have a reversed approach in this case.

Chapter 4

Results and Analysis

This chapter discusses and presents the results of reward decomposition and autonomous policy explanation in human-robot collaboration scenarios. In Section 4.1 the focus will be on the analysis of the explanations given by the reward decomposition, while autonomous policy explanation results will be analyzed in Section 4.2. As discussed in Chapter 2, these explanation methods not only give us insight into the reasons behind the choices made by the robot, but the analysis of those results is a powerful debugging tool to fix possible unexpected behaviors.

One of the most difficult challenges when building a RL algorithm is to design the reward function and these two methods played an important role in its refinement. The current function version chooses a dynamic range of actions instead of being stuck on the same overly safe choice as in the previous versions [4].

The data analyzed in the following sections have been collected over the robot's operation over a couple of days. During this time, the robot, while learning the policy, was assigned every time a randomly chosen goal, and once it reached that goal, it was assigned another one until completion. The warehouse scenario was not always the same, but it changed every time between two possible options, with an unchanged position of the walls but different positions for static obstacles. Only the actions actually chosen using the network were taken into account in this analysis, while the randomly chosen ones, which are required by Q-learning, have been excluded.

4.1 Reward Decomposition

Reward decomposition [3], as described in Chapter 2 can be considered as a hybrid method of explainability because despite being a transparent algorithm, which provides self-explanation thanks to its architecture, it also provides post-hoc explainability through the analysis of the rewards and sub-rewards given to the robot during its operation.

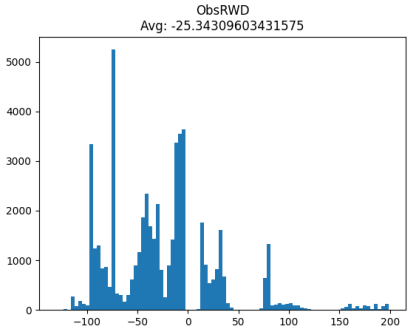
It is actually considerably difficult trying to understand which factors impacted the robot's choice the most in a classic RL algorithm. The reason is that they are all blended together in just a cumulative reward which gives us no information about how it was obtained. In this case, instead, it is possible to understand and weigh the importance of each of the reward types through graphical explanations.

4.1.1 Reward Types Balance

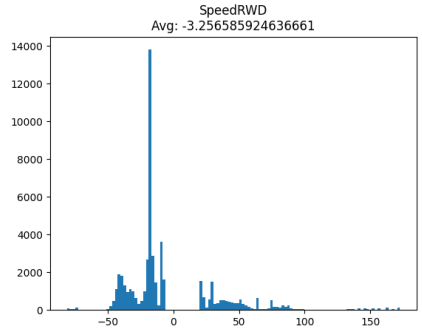
Reward decomposition [3] presents some challenges when considering the splitting of the rewards because, when writing the reward function, all rewards fall evenly into the same range of possibilities otherwise, some reward types which have higher magnitude may cause the others to be negligible in the total reward sum for instance if the "Obstacle Reward" falls in the range $[-500, 500]$ while the "Speed Reward" has a range of $[-50, 50]$, it is very likely that in most cases the first type of reward would outpace the second, often making it have no impact on the final choice decision.

This challenge exists even when not using the reward decomposition method because the final reward in classic RL methods is still a composition of several factors, but in our case, we have the opportunity to control it better and balance out how much every factor is taken into account. It is not always required or even convenient to balance out the reward types: for example, in our scenario, the "Collision Reward", has purposefully a different scale and approach compared to the others, which on the contrary are scaled to have equal importance.

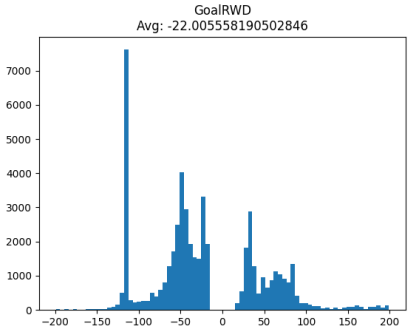
In Figure 4.1 it is possible to visualize the distributions and averages for each kind of reward assigned by the reward function during the robot's operation. The plot represents histograms where, on the x-axis, each bin represents the frequency of the choice of that value for that reward type. As mentioned above, the goal here is to verify that none of the reward types has a range of values too wide or too narrow compared to the others, and so that all of them fall in the same range. This is necessary to avoid that some reward



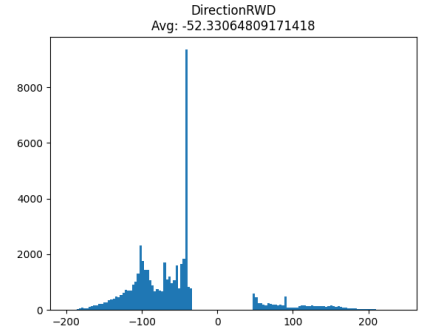
(a) Distribution of Obstacle Reward.



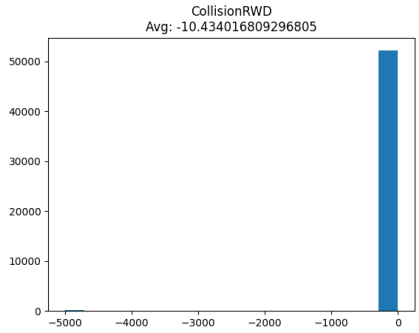
(b) Distribution of Speed Reward.



(c) Distribution of Goal Reward.



(d) Distribution of Direction Reward.



(e) Distribution of Collision Reward.

Figure 4.1 – Distributions and averages of the five reward types assigned by the reward function during the robot's operation. The histograms represent on the y-axis, for each reward type, how many times a value on the x-axis has been selected by the function

types overrule the others by having a bigger magnitude just as a consequence of how the reward function was designed. It is possible to verify that the rewards, with the exception of the collision reward which can be considered as a special

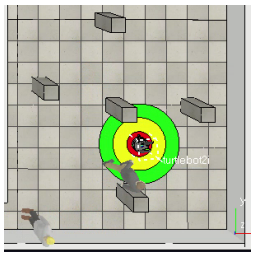
case, are approximately all distributed over an interval between $[-150, 200]$ and therefore our goal of balancing them out is achieved. The averages seem quite balanced too without remarkable disparities between them.

The collision reward, which is a special case, has only two values: 0 if there is no collision, or -5000 if a collision happened.

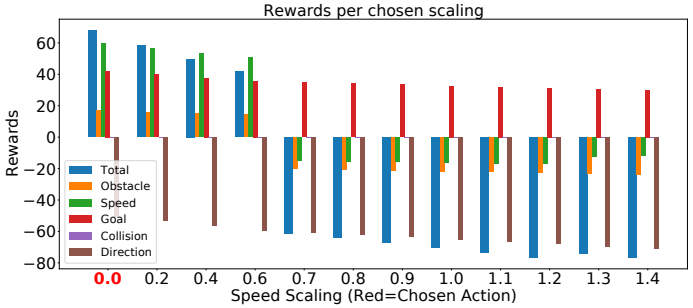
4.1.2 Action Comparison

Reward decomposition makes it possible to compare the rewards of all the possible actions, meaning the speed scaling factors presented in Section 3.2, between them, evaluating what would have been the total reward if the robot had chosen each of the twelve possible actions but also going on a deeper level, comparing the single types of reward. We can therefore have a graphical representation of the rewards which is a valuable tool to help understand why the robot takes some decisions and provide explainability of the algorithm.

In Figure 4.2 it is illustrated an example of the graphical representations we can get with reward decomposition. In the plot Figure 4.2a there is the analyzed warehouse scenario and in Figure 4.2c we have the most relevant values that characterize the rewards. In Figure 4.2b we have a plot that compares rewards for each possible action. There are six bars: the first blue



(a) Warehouse scenario



(b) Graphical representation of rewards per each action

Nearest Obs. Distance	Linear Speed	Rotational Speed
Critical	Medium	Counterclockwise

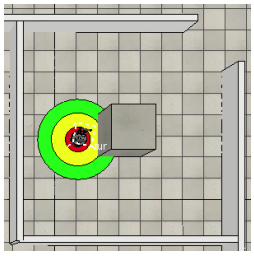
(c) Relevant values

Figure 4.2 – Warehouse scenario, relevant values and graphical representation of rewards per each action when the algorithm chooses the optimal action

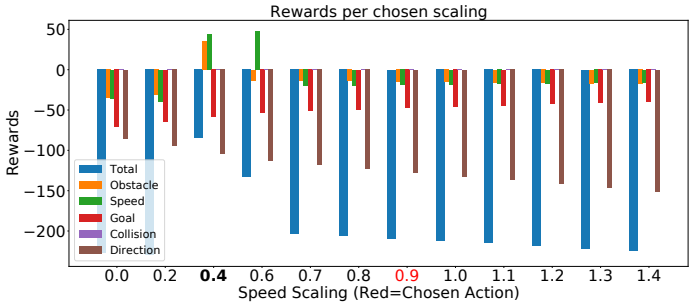
one represents the total reward, sum of all the five rewards, while the sub-rewards are individually represented by one bar each. Respectively in the order, there are obstacle reward, speed reward, goal reward, collision reward, and direction reward. Moreover, on the x-axis, the action with the best total reward is marked in bold and the action chosen by the algorithm for that iteration is shown in red.

In this case, it is possible to see that the robot is in a difficult situation because it is surrounded by obstacles, one of which with a high risk value, the human, which has unpredictable behavior. It is therefore reasonable that the robot chooses to stop with a speed scaling factor of 0.0 which, in this case, also represents the optimal action.

But the robot does not always choose the action that maximized the reward given by the reward function as shown in Figure 4.3. Indeed, in this case the robot needs to reach a goal outside that room, and is therefore looking for a way out, which is in the top right corner of the Figure 4.3a. In this case the algorithm, as we can see from Figure 4.3b, chooses to almost keep the same speed with a speed scaling of 0.9 instead of the optimal action represented by a speed scaling factor of 0.4. Slowing down quite drastically is reasonable because the robot is fast and rotating in the direction of the way out from the room, which is however also the direction of the obstacle, so the robot is basically going towards the obstacle at a fast pace. The reason why slowing



(a) Warehouse scenario



(b) Graphical representation of rewards per each action

Nearest Obs. Distance	Linear Speed	Rotational Speed
Warning	Fast	Clockwise

(c) Relevant values

Figure 4.3 – Warehouse scenario, relevant values and graphical representation of rewards per each action when the algorithm does not choose the optimal action

down of a factor of 0.4 is better than stopping is that the obstacle is still in the warning zone, so with the right rotation direction and a reasonable pace it is still possible for the robot to avoid the obstacle without the need to stop, therefore avoiding compromising too much the efficiency.

Top	**1**	**2**	**3**	**4**	**5**	**6**
Freq % Undertrained	16.00%	22.67%	31.67%	41.34%	46.67%	55.00%
Freq % Compl. Training	29.58%	37.17%	44.27%	51.03%	57.40%	63.71%
Freq % After Training	44.48%	47.99%	62.37%	67.66%	76.57%	80.85%

Top	**7**	**8**	**9**	**10**	**11**	**12**
Freq % Undertrained	62.00%	69.67%	76.67%	85.33%	91.33%	100%
Freq % Compl. Training	69.65%	75.52%	81.93%	87.81%	93.13%	100%
Freq % After Training	82.37%	86.42%	88.73%	93.82%	95.32%	100%

Table 4.1 – Progressive ranking of actions chosen by the robot

It becomes therefore interesting to understand how often the robot chooses the optimal action, where optimal action means the action among all possibilities which maximizes the total reward. In Table 4.1 it is presented the progression of the ranking of the actions chosen by the robot in three cases: considering statistics of the early training stages (undertrained), considering statistics on the whole training process (complete training) and lastly considering statistics of the robot's operation after training. The table presents a progression of how many times the robot chooses the action belonging to the set of the top 1 choice, top 2 choices, top 3 choices, and so on up to the final top 12 choices which represents the 100% of the iterations. If we examine as an example the top 3 column, in the first case the robot chose for 31.67% of the iterations an action among the best 3 possibilities in terms of total reward, while in the second case it increased to 44.27% of the times chose an action among the best 3 possibilities and in the last case, after training is completed, this value boosts to 62.37%. Theoretically, if the robot chooses random actions, the progression increases between one column to the other

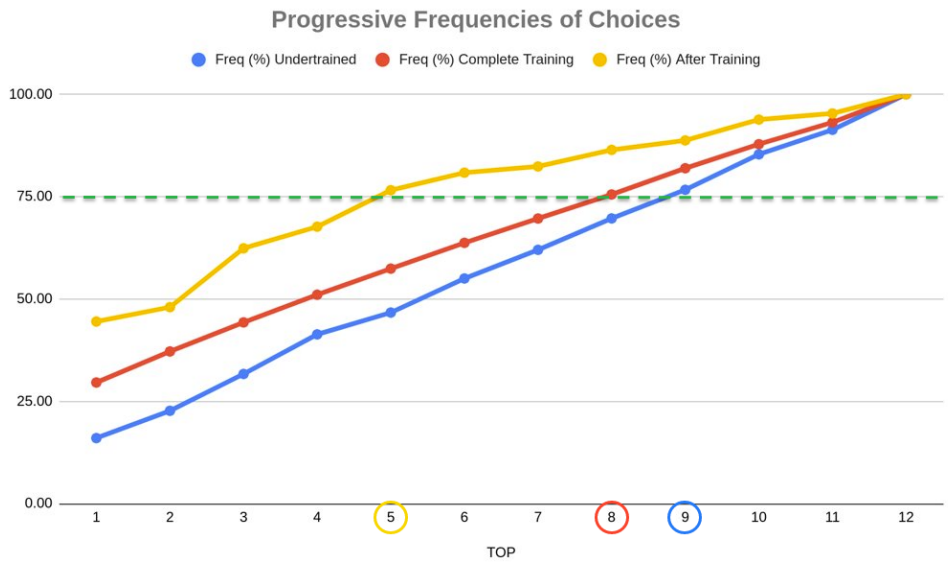


Figure 4.4 – Plot of values in Table 4.1 with highlight on 75% threshold reach

of around $100\%/12 = 8.33\%$. From this table, observing the differences between the three cases and keeping in mind what would happen if the choices were random, we can see a remarkable improvement given by an increased training time, especially if we consider the after training model. Even only considering the optimal action (top 1 column), it is clear that it is chosen with an almost doubled frequency comparing the trained to the under-trained case, and it triplicates considering the after training case, reaching almost 50% of the choices. All the values have been plotted in Figure 4.4, highlighting that 75% of the choices in the after training case are among the top 5 choices, while for the complete training they are among the best 8 and for the undertrained case among the best 9. This analysis helps the users trust the model and the RL algorithm because it becomes clear that the robot learned how to behave in most situations. Even though there is room for improvement, if we consider the complexity of the states and possible situations the robot may face, this result is encouraging.

4.1.3 RDX and MDX

The previous plots in Figure 4.2 and Figure 4.3 which consider all the possible actions can be useful to have an overview of the options given to the robot. However, if we want to have a deeper analysis and focus on specific actions,

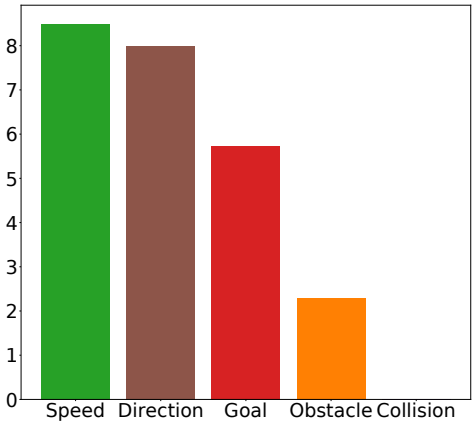
we rely on pairwise action comparison, which means comparing the chosen action to one alternative specified by the user. The full picture is given by RDX, which allows checking the differences between all the types of rewards in descending order. However, sometimes it is useful to find a set of meaningful types of rewards, excluding the ones which do not have much impact on the choice, and MSX is the tool that allows us to select minimal sets of relevant reasons.

The possible scenarios when using RDX and MSX can be summarized in four categories whose RDX diagrams are shown in Figure 4.5:

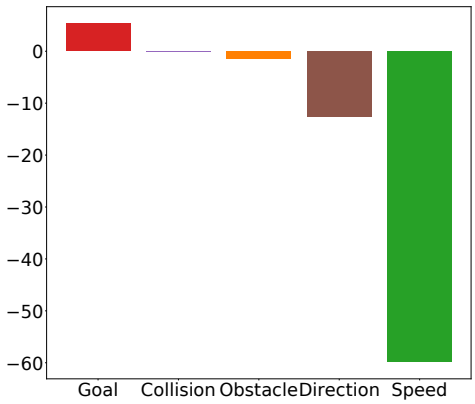
1. The action chosen by the robot is better than the compared action for each one of the reward types: Figure 4.5a, represent the RDX when we compare the speed scaling of 0.0 chosen by the algorithm to an alternative selected by the user of 0.7 for the situation presented in Figure 4.2. All the components of the RDX are positive except for the collision reward which is 0. In this case, the action chosen performs better than the compared one for all the reward types and therefore there is no meaning in MSX exploration.

2. The action chosen by the robot is worse than the compared action if we consider the sum of the five reward types: Figure 4.5b, represents the RDX when we compare the speed scaling of 0.9 chosen by the algorithm to an alternative selected by the user of 0.6 for the situation presented in Figure 4.3. The user alternative would have actually performed better than the one chosen from the algorithm, therefore, if we look at the definition, even in this case MSX exploration has no meaning.

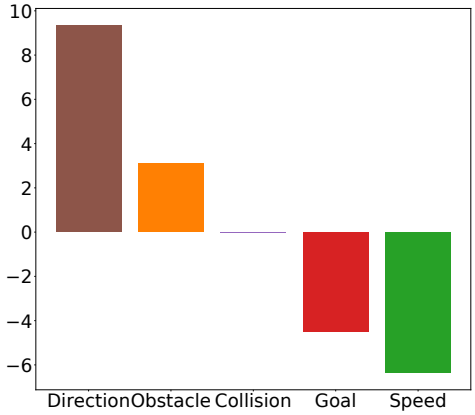
3. The MSX is meaningful but does not provide any compression with respect to the RDX: we have this case presented in Figure 4.5c, which does not correspond to any of the possible comparisons in Figure 4.2 or Figure 4.3, but it is an example case taken from another iteration. Indeed, this is a rather rare case because we usually get a reduced set of rewards when applying MSX to cases in which is meaningful. In this case, the MSX+ set is represented by both direction and obstacle reward because none of them individually would be enough to overcome the disadvantage represented by goal and speed rewards. At the same time, the MSX- set is represented by both goal and speed reward because none of them individually can overcome the direction reward, which, if we look again at the definition of how we create MSX-, is the only remaining component if we remove from the MSX+ set the least important element



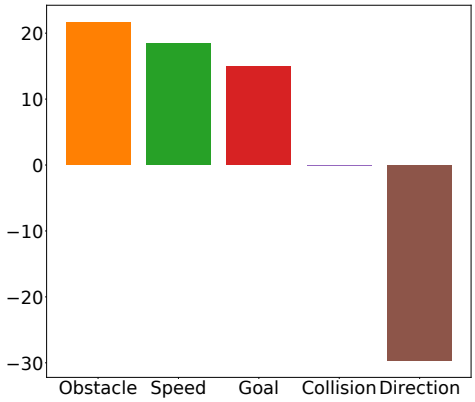
(a) Case 1: The action chosen by the robot is better than the compared action for each one of the reward types



(b) Case 2: The action chosen by the robot is worse than the compared action for the sum of the five reward types



(c) Case 3: The MSX is meaningful but does not provide any compression with respect to the RDX



(d) Case 4: The MSX is meaningful and provides a reduced set with respect to the RDX set.

Figure 4.5 – RDX graphical representation of the four possible scenarios. The x-axis presents the RDX components in descending order, the y-axis values correspond to the reward differences between the two action alternatives for each of the reward types.

(obstacle reward in this case). In this case, the MSX sets do not provide any compression with the exclusion of the collision reward because all components were determining to the choice made by the algorithm.

4. The MSX is meaningful and provides a reduced explanation with respect to the RDX set: Figure 4.5c, represent the RDX when we compare the speed scaling of 0.9 chosen by the algorithm to an alternative selected

by the user of 0.2 for the situation presented in Figure 4.3. There is just one very negative component in the RDX. In this case, the MSX+ is only represented by the obstacle and speed rewards because they manage to overcome the disadvantage created by the direction reward. This means that the impact of the goal reward, in this case, is negligible. The MSX- is therefore only represented by the direction reward, which, as explained in case 3, can overcome the advantage of the obstacle reward. It is also very common that MSX+ set is composed of only one reward because sometimes one component of RDX has so much impact that it becomes the only relevant one. In that case, as a consequence, MSX- is an empty set.

4.2 Autonomous Policy Explanation

Autonomous policy explanation, as anticipated in the previous chapter, heavily relies on the state diagram of the robot's operation and on understanding which actions are preferred by the robot at each state. The state has been reduced to the consideration of only two factors:

- Nearest obstacle's distance which has been divided into 3 categories: safe, warning, or critical zone.
- Robot's speed which has been divided into 3 categories too: fast, medium or slow speed

In the same way, also the possible actions have been categorized in stop, if the speed scale is 0.0, slow down, if the speed scale is between 0.2 and 0.8, keep the same speed if the speed scale is between 0.9 and 1.1, and speed up if the speed scale is higher than 1.2.

The same data that has been described at the beginning of the chapter is also used in this section to extract information about the robot's operation.

4.2.1 State Diagram and Action Selection Frequency

The algorithms presented in Chapter 3 need information on the algorithm running statistics and transitions between states to work, and therefore it is useful to analyze these statistics before analyzing the behavior of the autonomous policy explanation algorithms.

State Diagram

The state diagram is a directed graph that shows the probability of transition between one state and all the others. Each node represents a possible state, and in this case, since we have 3 possibilities for the nearest obstacle distance and we have 3 options for the speed of the robot, we have $3^2 = 9$ possible states, which will become the nodes of our graph. In our scenario, it is very likely that the robot does not change its state at each iteration but may need several iterations to actually switch to a new one. For this reason, the majority of transitions between iterations do not result in any state transition, but that does not represent the intention of the algorithm. Indeed, even if the robot is slow and the intention is to speed up, while the chosen action actually represents what the algorithm is trying to achieve, there may be no state transition up until several iterations later. This is the reason why to create the state diagram, only the transitions between different states were considered, excluding self-transitions.

The state diagram generated from the robot's operation can be seen in Figure 4.6. The nodes' state descriptions have been shortened for better readability, and they have been ordered so that on the top layer we have slow speed, in the middle medium speed, and at the bottom fast speed. In the same way, we have the states with the nearest obstacle in the safe zone on the left side, in the warning zone in the middle, and in the critical zone on the right side. The numbers on the directed edges represent the probability (in percentage) that from one state the robot would end up in another state. As an example, if we look at the top of the diagram, we can see that of all the transitions from "Warning-Slow" to another state, 30.63% of the times it switched to the state "Safe-Slow". In the same way, from "Safe-Slow", 40.21% of the times the robot switches to "Warning-Slow", while 32.99% of the times it switches to "Safe-Fast". For simplicity, not all the transitions have been included in the graph, but only the ones with probability >30%.

It is interesting to notice that there aren't many long edges in the graph, which means that the transitions are usually quite gradual, with the exception of the transitions "Safe-Slow" to "Safe-Fast" and "Critical-Fast" to "Safe-Fast", which are very desirable transitions because in the first case we want the robot to speed up when it has no obstacle around to keep a good efficiency, but we also want the robot to promptly move away from dangerous situations like the one represented by "Critical-Fast".

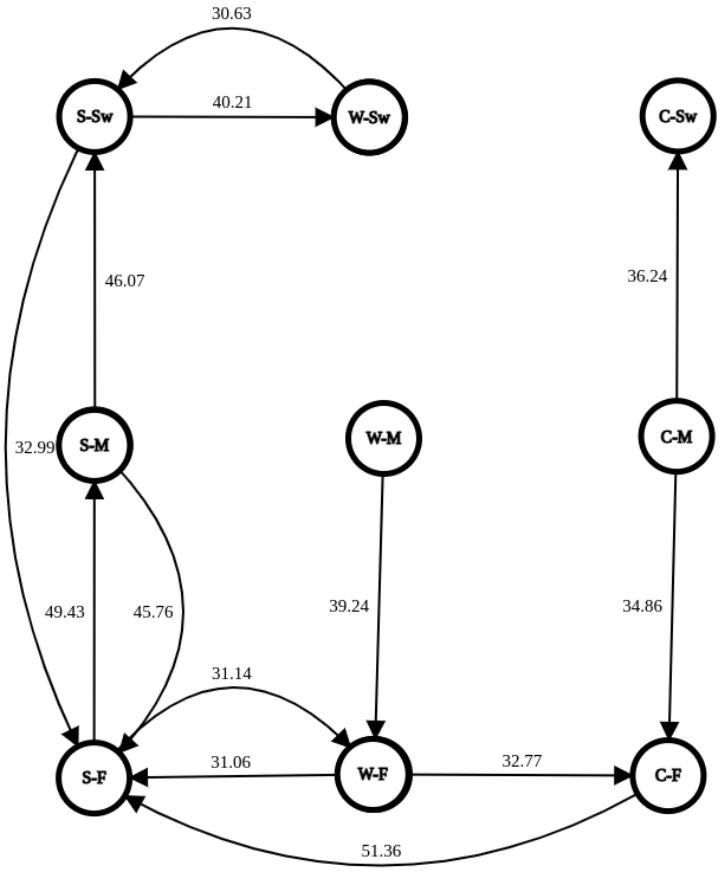


Figure 4.6 – State Diagram derived from robot's operation (S = Safe zone, W = Warning zone, C = Critical zone, Sw = Slow, M = Medium, F = Fast).

Action Selection Statistics

Not only the state transition analysis is important to understand the robot's behavior, but it is also desirable to understand which are the actions selected from the robot in each situation. Therefore, for each of the algorithm's iterations, after mapping the nearest obstacle distance, the speed, and the action to the relative classes, it is possible to analyze for each of the 9 possible states, how often each action has been selected. The detailed analysis of these statistics is in Table 4.2: the frequency percentage is relative to each state, and the most selected one is highlighted in bold. The table also presents a comparison between the trained model and an under-trained model (in the column Freq. UT), which was only trained for 1000 iterations. This comparison it is useful to highlight how the policy changes and gets better training the model for an adequate amount of time. The majority of the most

NO Distance	Speed	Action	Freq.	Freq. UT
Critical	Slow (S_0)	**Stop**	**38.13% (1057)**	6.98%
		Slow Down	25.14% (697)	30.27%
		Keep the same speed	17.71% (491)	**32.56%**
		Speed Up	19.01% (527)	30.23%
	Medium (S_1)	**Stop**	**57.31% (905)**	11.11%
		Slow Down	18.18% (287)	11.11%
		Keep the same speed	13.81% (218)	**55.56%**
		Speed Up	10.7% (169)	22.22%
	Fast (S_2)	**Stop**	**47.39% (2212)**	3.03%
		Slow Down	22.73% (1061)	36.36%
		Keep the same speed	15.32% (715)	15.15%
		Speed Up	14.57% (680)	**45.45%**
Warning	Slow (S_3)	Stop	14.88% (1475)	6.1%
		Slow Down	**31.03% (3075)**	**36.62%**
		Keep the same speed	28.62% (2836)	29.58%
		Speed Up	25.47% (2524)	27.7%
	Medium (S_4)	Stop	11.17% (135)	15.0%
		Slow Down	**30.27% (366)**	15.0%
		Keep the same speed	30.02% (363)	**35.0%**
		Speed Up	28.54% (345)	**35.0%**
	Fast (S_5)	Stop	13.06% (770)	7.95%
		Slow Down	**31.32% (1847)**	**35.23%**
		Keep the same speed	27.0% (1592)	32.95%
		Speed Up	28.62% (1688)	23.86%
Safe	Slow (S_6)	Stop	5.92% (590)	7.29%
		Slow Down	30.31% (3019)	31.17%
		Keep the same speed	**32.71% (3258)**	**34.01%**
		Speed Up	31.05% (3092)	27.53%
	Medium (S_7)	Stop	7.27% (239)	11.43%
		Slow Down	30.57% (1005)	**31.43%**
		Keep the same speed	29.93% (984)	25.71%
		Speed Up	**32.24% (1060)**	**31.43%**
	Fast (S_8)	Stop	5.58% (723)	8.3%
		Slow Down	28.28% (3662)	**36.1%**
		Keep the same speed	30.04% (3890)	35.02%
		Speed Up	**36.11% (4676)**	20.58%

Table 4.2 – Statistics on the actions selected by the robot for each state

frequent choices seem reasonable for the first model: when an obstacle is in the critical area, the robot understands that is a risky situation and stops most of the times, while with the nearest obstacle in the warning zone, it tends to slow down. At the same time, some situations are still not crystal clear to the robot, like for example the situation when the robot is slow but the nearest obstacle is in the safe zone: the robot tends to keep the same speed while it would be best to speed up to ensure a good efficiency. It is also relevant to notice that, especially for the cases where the nearest obstacle is in the safe zone, the differences in frequency between the possibilities "slow-down", "keep the same speed" and "speed up" are not very marked.

While for the first model the choices match for the most part the expectations, for the under-trained model there are some clear divergences: for example, when the obstacle is in the critical zone and the robot is fast, the policy suggests that the action to be taken would be to speed up, which is definitely the worst possibility to ensure safety.

4.2.2 Policy Explanation

Using the tools explained above and the four algorithms explained in Appendix B we can make the robot answer the questions illustrated in Chapter 3 which make us analyze the policy learned by the robot. The policy is not followed by the robot at every iteration, but it describes what the robot will most likely do during its operation. The four algorithms are a tool to explore the whole policy and get explanations about what the robot learned, focusing only on the parts that are more interesting for the user. In the following, there will be examples of questions and answers that can be provided with this method, and there will be a comparison with what would have been the answers to the same questions in the case of the under-trained model with only 1000 iterations, the same analyzed in Table 4.2.

In which state does the robot perform a certain action?

This is the first kind of question we will answer and to do so we need to use the DASRI algorithm. The system will ask the user to select the action to analyze. To see the steps of the algorithm takes to answer the question, we will use as an example the question "In which state does the robot stop?". The first step is, using the statistics collected and analyzed in the previous section, understanding for each state which is the most likely action. Once we found the most likely actions for each of the states, if the state has as preferred action the one selected by the user, it goes in the target states set, otherwise in the

non-target states set. Here is the complete example for our case, where the states from S_0 to S_8 follow the same progression of Table 4.2, therefore $S_0 =$ "Critical-Slow", $S_1 =$ "Critical-Medium" up until $S_8 =$ "Safe-Fast"

1. Find the most likely action for each state	S_0 : stop, S_1: stop, S_2: stop, S_3: slow down, S_4: slow down, S_5: slow down, S_6: keep the same speed, S_7: speed up, S_8: speed up
2. For each state, if the analyzed action is the most likely, add the state to the target states, otherwise to the non-target states	Target states: S_0, S_1, S_2 Non-target states: S_3,S_4,S_5,S_6,S_7,S_8

This result means that the states where the robot is most likely to speed up are from S_0 to S_2 which corresponds to the states where the nearest obstacle is at critical distance, independently from the speed it has. In the case of the under-trained model, as we can analyze from Table 4.2, stopping is never the most frequent action, and so all the states go in non-target states set. It is therefore easy to understand that the under-trained model would have safety issues when the nearest obstacle is in the critical zone, because in that case we would want the robot to stop, especially if at fast or medium speed.

When does the robot perform a certain action?

This question is strongly related to the previous one and therefore in the system they are combined in the same function. The output to the first question serves as input to the SRLC algorithm which can give as an answer not just states, but an actual summary of what they represent in natural language. The first step is to convert the target states found with the previous algorithm in binary representations using the six predicates illustrated in Chapter 3: IsObstacleCritical, IsObstacleWarning, IsObstacleSafe, IsRobotSlow, IsRobotMedium, and IsRobotFast. After codifying the states, we need to minimize them using the Quine-McCluskey algorithm in a concise representation that best covers the target state region, and as the last step, we convert the binary representation back into natural language.

1. Convert target states to binary representations based on predicates	$S_0 = [100\,100]$ $S_1 = [100\,010]$ $S_2 = [100\,001]$

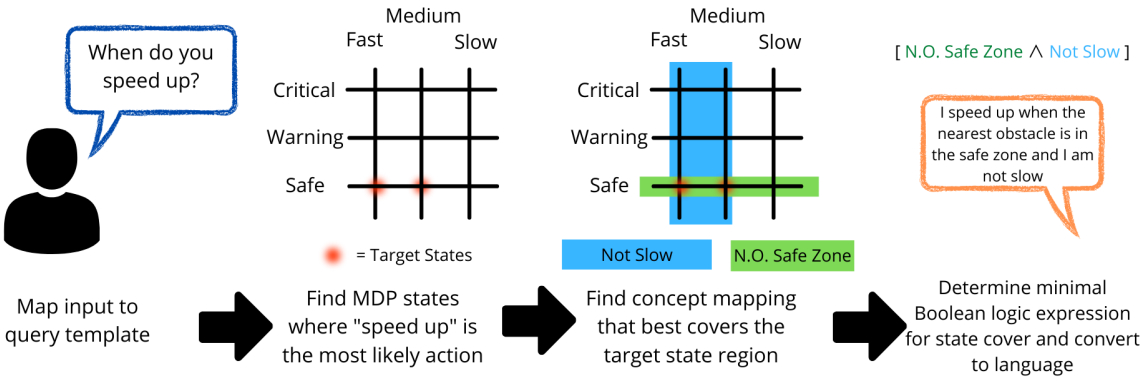


Figure 4.7 – Approach for mapping an action query to a policy explanation. Figure adapted from [1]

2. Apply Quine-McCluskey [16] algorithm to find the minimal representation	Minimize: [[100 100], [100 010], [100 001]] into: [100 - - -]
3. Convert the binary representation back into natural language	"The nearest obstacle is in the critical zone"

A similar process combining the first two algorithms, but asking the robot when does it speed up, is shown in Figure 4.7.

Why does not the robot perform a certain action in a given state?

This question provides a more targeted insight on a specific iteration of the robot's operation comparing what is the action that the robot chose against what it should have chosen according to the policy. The system will ask the user to input the progressive number of the iteration, and, after showing the state of the robot at that time and the chosen action, it will ask the action to compare with it. The example that will be shown corresponds to an iteration the nearest obstacle is in the safe zone and the robot is at medium speed, at its choice is to speed up. We may want to ask the robot why it did not slow down since it is at medium speed.

1. Perform DASRI algorithm on the chosen action to find target state and non-target states	Target states: S_3, S_4 and S_5 Non-target states: $S_0, S_1, S_2, S_6, S_7, S_8$
2. Apply SRLC algorithm to find differences between the state of the robot and the target states	State of the robot: Safe-Medium Target states: The nearest obstacle is in the warning zone for each of the speeds Difference: The obstacle is in the safe zone
3. Apply SRLC algorithm to find differences between the target and non-target states	Minimization of "slow down" states: the nearest obstacle is in the warning zone
4. Result	"I did not slow down because the nearest obstacle is in the safe zone. I slow down when the nearest obstacle is in the warning zone"

We must take into account that when applying this algorithm there may be the case in which the action selected by the robot is not the one that the robot would have chosen if it had followed the policy, and in that case, this algorithm would fail because the autonomous policy explanation only takes into account the overall learned policy. To have a detailed insight into such cases, the suggestion is to use other methods like the reward decomposition presented above.

What does the robot do when certain predicates are met?

This is the last and probably the most useful of the four options we have to analyze the learned policy, answered using the SBC algorithm. The system will ask the user to choose a state to analyze, first asking which is the distance of the nearest obstacle and then asking the speed of the robot. The user can even decide to select just one of the two state components and leave the other one undefined. In the example, the question that will be answered is "What does the robot do when it is at a fast speed?"

1. Convert user input to binary representation	"Fast speed" : [- - - 001]
2. Find all states that match the user input	Matching states: S_2 =[100 001] S_5 =[010 001] S_8 =[001 001]
3. Find actions that the robot would choose in each of the states	S_2 = Stop S_5 = Slow down S_8 = Speed up
4. For every found action, use SRLC algorithm to minimize the conditions of the corresponding states versus all the other states, omitting the part related to the user input	"I will stop if the nearest obstacle is in the critical zone. I will slow down if the nearest obstacle is in the warning zone. I will speed up if the nearest obstacle is in the safe zone."

While this answer is in line with what we would expect the robot to do, asking the same question to the under-trained model, the answer is "I will speed up if the nearest obstacle is in the critical zone. I will slow down if the nearest obstacle is not in the critical zone". This answer highlights a dangerous behaviour, especially in the first part when considering that at fast speed and with an obstacle in the critical zone, the robot would speed up.

Complete policy

The last algorithm also allows us to not specify any predicate and leave everything undefined. In that case, we would have the natural language version of the entire policy. In our case, the entire policy produced by the algorithm is the following:

> I will stop if the nearest obstacle is in the critical zone. I will slow down if the nearest obstacle is in the warning zone. I will keep the same speed if the nearest obstacle is in the safe zone and I am slow. I will speed up if the nearest obstacle is in the safe zone and I am not slow.

This format can easily help the developers spot any anomaly in the robot's behavior. In this case, for example, keeping the same speed when the obstacle is far and the robot is slow does not seem like a good strategy to have to favor

the efficiency of the operation. But these tools, as mentioned multiple times above, are also useful for debugging purposes. In this case, we know that the reward function should be modified in the branches corresponding to slow speed and safe zone, perhaps giving a penalty in that case. The resulting policy in our case, which was the result of several rounds of tuning for the reward function, seems to be quite satisfactory and reflecting almost completely the behavior that we would expect from the robot.

Chapter 5

Conclusions and Future work

5.1 Conclusions

The purpose of this thesis was to find methods that would allow developers and users to gain some insight into the algorithms that guided the robot's behavior ensuring safe operations. This is particularly useful if we consider the nature of the RL algorithms which are black-box systems that learn on their own and our only way to control them is through proper architectures and reward function tuning, but also extremely challenging. When dealing with safety in the workplace in a human-robot collaboration scenario, it becomes fundamental to gain some knowledge about what the robot learned and how it would deal with risky situations. This goal has been achieved in this work with the use of two different techniques that act on different granularity: reward decomposition lets us explore locally each action to understand which were the most determinant factors that impacted the robot's choice, while autonomous policy explanation gave us a global post-hoc explanation about the overall robot behavior, making it possible even for non-expert users to get answers thanks to the natural language output. The use of both these methods builds trust in the RL model and it becomes a powerful tool for debugging purposes, showing the developer aberrant behaviors and allowing the developer to easily understand where to intervene to solve them. The refinement of such sophisticated algorithms is not an easy task and the journey to the perfect policy is facilitated by the two presented explainability techniques: one proof of this can be how the robot's choices became somehow more dynamic and smarter. Indeed, the policy produced by autonomous policy explanation tells us that the robot overall chooses proper actions depending on the situation it is in, while the dynamicity can be seen in Table 5.1 which shows for each

action, the number of times that has been chosen by the robot. The distribution results quite even with the exception of actions 0.0 and 1.4 which, being the extremes, are naturally giving higher rewards respectively in very risky or safe situations. It is relevant to remember that the robot before [4] tended to choose very frequently the same slowing down action to be very safe, but not very efficient.

Action	**0.0**	**0.2**	**0.4**	**0.6**	**0.7**	**0.8**
Frequency	8106	3524	3511	4469	3515	3426

Action	**0.9**	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**
Frequency	3498	3726	3697	4165	3624	6972

Table 5.1 – Frequency of actions chosen by robot

5.2 Future work

There is still plenty of room for improvement in this project and here are some of the modifications that can be made to try to improve both the operational and the explanation parts.

The first substantial change that should be implemented is to integrate the navigation module of the robot with the risk management and risk mitigation modules. This change would be particularly beneficial because, as it is now, the risk mitigation operates only after the navigation module has made some choices to optimize the robot operation, and the risk mitigation is only intervening on a choice that should be already optimized based on the parameters of the navigation module. In this way, these two modules which are the core of the robot operation, may end up interfering with each other in many situations e.g. if the navigation module believes it is okay to have a medium speed in a risky situation, the risk mitigation module will slow it down, but at the next iteration the navigation module would still apply a medium speed and the risk mitigation will have to intervene again and again on the same situation. The integration would therefore help avoid this kind of inconsistent behaviors ensuring coherent actions at every iteration.

Another improvement that can be done is again related to the navigation module: for this project, we switched from the previous ROS Kinetic release to the next generation ROS Melodic. Having to switch ROS version, some parameters of the navigation and their integration in the module changed,

especially if we consider the introduction of plugins. The consequence was having to re-optimize all the parameters, but even with this change, the robot movements resulted jerkier and less smooth than in the previous version. This is another reason why the integration of the two modules mentioned above could result quite beneficial.

For what concerns the algorithmic part, the main improvements that could be done are two:

- Improving the reward function: as mentioned multiple times before, this is a process that requires time and a lot of tuning to be able to find the perfectly balanced reward function, and even if the results are not perfect yet, so far it seems to behave quite appropriately.
- Try different architectures: in this thesis, only the MLP architecture was considered, which is simpler than a CNN. It could be beneficial to try and explore other architectures, even if the training may require more time to start producing satisfactory results.

In reward decomposition, more categories of rewards could be found that look more into detail of the operation. The advantage of having the reward split into even more types is that we get an even deeper insight, and RDX and MSX will gain even more importance in the analysis than with just 5 types. For autonomous policy explanation, it seems somehow limiting to restrict the policy to the consideration of only the most chosen action for each state because, looking at Table 4.2, sometimes the difference of frequency between two actions in the same state is not as remarkable as in some other cases. Another improvement that could be implemented is to include in the policy something similar to the direction reward as a variable e.g. categorize the direction where the robot is turning in two classes like “towards the nearest obstacle” or “away from the nearest obstacle”, even though this may over-complicate the model without giving much more insight from the explainability point of view.

Besides these implementation improvements, which will help us answer better the question about why did the robot choose a certain action, what is still left to explore is how to fix unexpected behaviors once they have been detected using these methods, and the future research should go in the direction of linking the explanation provided by XRL methods to what is needed to fix unwanted behaviors.

References

[1] B. Hayes and J. A. Shah, "Improving robot controller transparency through autonomous policy explanation," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '17. New York, NY, USA: Association for Computing Machinery, 2017. doi: 10.1145/2909824.3020233. ISBN 9781450343367 p. 303–312. [Online]. Available: https://doi.org/10.1145/2909824.3020233

[2] E. Puiutta and E. Veith, "Explainable reinforcement learning: A survey," *arXiv preprint arXiv:2005.06247*, 2020.

[3] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable reinforcement learning via reward decomposition," in *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*, 2019.

[4] A. Terra, H. Riaz, K. Raizer, A. Hata, and R. Inam, "Safety vs. efficiency: Ai-based risk mitigation in collaborative robotics," in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, 2020. doi: 10.1109/ICCAR49639.2020.9108037 pp. 151–160.

[5] R. Inam, E. Fersman, K. Raizer, R. Souza, A. Nascimento, and A. Hata, *Safety for automated warehouse exhibiting collaborative robots*, 06 2018, pp. 2021–2028. ISBN 9781351174664

[6] R. Inam, K. Raizer, A. Hata, R. Souza, E. Forsman, E. Cao, and S. Wang, "Risk assessment for human-robot collaboration in an automated warehouse scenario," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018. doi: 10.1109/ETFA.2018.8502466 pp. 743–751.

[7] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," 2020.

[8] A. Hata, R. Inam, K. Raizer, S. Wang, and E. Cao, "Ai-based safety analysis for collaborative mobile robots," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019. doi: 10.1109/ETFA.2019.8869263 pp. 1722–1729.

[9] J. Guiochet, "Hazard analysis of human–robot interactions with hazop–uml," *Safety Science*, vol. 84, pp. 225 – 237, 2016. doi: https://doi.org/10.1016/j.ssci.2015.12.017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092575351500346X

[10] M. Y. Yang, W. Liao, H. Ackermann, and B. Rosenhahn, "On support relations and semantic scene graphs," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 131, pp. 15 – 25, 2017. doi: https://doi.org/10.1016/j.isprsjprs.2017.07.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271617300746

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[12] "Simple reinforcement learning: Q-learning | by andre violante | towards data science," https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56, (Accessed on 01/21/2021).

[13] "Deep q-learning | an introduction to deep reinforcement learning," https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/, (Accessed on 01/21/2021).

[14] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[15] J. R. Kok and N. Vlassis, "Sparse cooperative q-learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 61.

[16] E. J. McCluskey, "Minimization of boolean functions," *The Bell System Technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956. doi: 10.1002/j.1538-7305.1956.tb03835.x

[17] “Ha800-hri / hayes-shah · gitlab,” https://gitlab.tue.nl/ha800-hri/hayes-shah, (Accessed on 01/26/2021).

Appendix A

Reinforcement Learning Reward Function in Python

```
if i == 0:  # obstacle reward
    if nearest_obstacle_distance <= r_critical:
        if scaled_speed > 0.2:
            reward = -20 * maxRisk * (scaled_speed_factor + 1)
        else:
            reward = +20 * maxRisk / (scaled_speed_factor + 1)
    elif nearest_obstacle_distance <= warning_zone:
        if scaled_speed <= 0.2:
            reward = -50 / maxRisk / (scaled_speed_factor + 1)
        elif scaled_speed > 0.25:
            reward = -7 * maxRisk * (scaled_speed_factor + 1)
        else:
            reward = +20 * maxRisk * (scaled_speed_factor + 1)
    elif nearest_obstacle_distance <= clear_zone:
        if scaled_speed <= 0.25:
            reward = -65 / maxRisk / (scaled_speed_factor + 1)
        elif scaled_speed > 0.35:
            reward = -5 * maxRisk * (scaled_speed_factor + 1)
        else:
            reward = +60 * (scaled_speed_factor + 1)
    else:
```

```
        if scaled_speed <= 0.35:
            reward = -75 / maxRisk / (scaled_speed_factor + 1)
        else:
            reward = +75 / maxRisk * (scaled_speed_factor + 1)

elif i == 1:  # speed reward
    if scaled_speed >= 0.35:
        if nearest_obstacle_distance > clear_zone:
            reward = +65 / maxRisk * (scaled_speed_factor + 1)
        else:
            reward = -20 * maxRisk / (scaled_speed_factor + 1)
    elif scaled_speed >= 0.2 and scaled_speed < 0.35:
        if nearest_obstacle_distance <= clear_zone and nearest_obstacle_distance > r_critical:
            reward = + 25 * maxRisk * (scaled_speed_factor + 1)
        elif nearest_obstacle_distance > clear_zone:
            reward = -65 / maxRisk / (scaled_speed_factor + 1)
        else:
            reward = -15 * maxRisk * (scaled_speed_factor + 1)
    elif scaled_speed < 0.2:
        if nearest_obstacle_distance < r_critical:
            reward = +50 / maxRisk / (scaled_speed_factor + 1)
        else:
            reward = -25 * maxRisk * (scaled_speed_factor + 1)

elif i == 2:  # goal reward
    if distance > 0.013:
        if nearest_obstacle_distance > clear_zone:
            reward = +75 / maxRisk * (scaled_speed_factor + 1)
        else:
            reward = -50 * maxRisk / (scaled_speed_factor + 1)
    elif distance >= 0.003 and distance < 0.013:
        if nearest_obstacle_distance > r_critical and nearest_obstacle_distance < clear_zone:
```

```
54              reward = +50 * maxRisk / (
    scaled_speed_factor + 1)
55          elif nearest_obstacle_distance < r_critical
    :
56              reward = -50 * maxRisk * (
    scaled_speed_factor + 1)
57          elif nearest_obstacle_distance > clear_zone
    :
58              reward = -75 / maxRisk / (
    scaled_speed_factor + 1)
59      elif distance < 0.003:
60          if nearest_obstacle_distance < r_critical:
61              reward = +50 * maxRisk / (
    scaled_speed_factor + 1)
62          else:
63              reward = -75 / maxRisk * (
    scaled_speed_factor + 1)
64
65  elif i == 3:  # collision reward
66      if done: # collision happened
67          reward = reward - 5000
68
69  elif i == 4:  # direction reward
70      if direction <= 3:  # obstacle on the left
71          if rotSpeed > 0.10: # turning clockwise
72              reward = +75 * maxRisk * (
    scaled_speed_factor + 1)
73          else:
74              reward = -60 * maxRisk * (
    scaled_speed_factor + 1)
75      elif direction > 3 or direction <= 7:  #
    obstacle in the center
76          if rotSpeed < 0.10 or rotSpeed > -0.10: #
    basically not rotating
77              reward = -60 * maxRisk * (
    scaled_speed_factor + 1)
78          else:
79              reward = +75 * maxRisk * (
    scaled_speed_factor + 1)
80      elif direction > 7:  # obstacle on the right
81          if rotSpeed < -0.10: # turning counter-
    clockwise
82              reward = +75 * maxRisk * (
    scaled_speed_factor + 1)
83          else:
84              reward = -60 * maxRisk * (
    scaled_speed_factor + 1)
```

Appendix B

Autonomous Policy Explanation Algorithms [1]

Dominant-Action State Region Identification Algorithm. Adapted from [1]

Input: Behavioural Model $G = \{V, E\}$, Target Action a
Output: Set of target states T_{π^a} where a is the dominant action, Set of non-target states $T_{\pi^{*/a}}$
$T_{\pi^a} = \{\,\}$
$T_{\pi^{*/a}} = \{\,\}$
for all $s \in V$ **do**
  $a^* \leftarrow$ most frequent action executed in s
  if $a^* == a$ **then**
    $T_{\pi^a} \leftarrow T_{\pi^a} \cup s$
  else
    $T_{\pi^{*/a}} \leftarrow T_{\pi^{*/a}} \cup s$
  end if
end for
return $T_{\pi^a}, T_{\pi^{*/a}}$

State Region to Language Conversion Algorithm. Adapted from [1]

Input: Target state set T, Non-target state set $\bar{T}$, set of predicates P
Output: String representation for clauses in DNF Formula of S grounded in elements of P
assert $T \cap \bar{T} = \emptyset$
to_include $\leftarrow \{\}$
to $\leftarrow \{\}$
for all $t \in T$ **do**
  state_val $= 0$
  i = 0
  for all $p \in p$ **do**
    state_val $| = (p(t) == True) << i$++
  end for
  to_include $\leftarrow$ to_include $\cup$ state_val
end for
for all $t \in \bar{T}$ **do**
  state_val $= 0$
  i = 0
  for all $p \in P$ **do**
    state_val $| = (p(t) == True) << i$++
  end for
  to_exclude $\leftarrow$ to_exclude $\cup$ state_val
end for
qm_minimization $\leftarrow$ Quine_McCluskey(ones = to_include, zeros = to_exclude)
clauses $\leftarrow$ [ ]
for all minterm $\in$ qm_minimization **do**
  string $\leftarrow$ " "
  for all literal $\in$ minterm **do**
    if literal is False **then**
      string+=p.negative_predicate
    else
      string+=p.positive_predicate
    end if
  end for
  clauses.append(string)
end for
return "or".join(clauses)

Behavioral Divergences Algorithm. Adapted from [1]

Input: Behavioural Model $G = \{V, E\}$, Target Action a, Previous state s_p, Distance threshold D_{const}
Output: Explanation of the difference between the current state and state region where a is performed, explanation of where a is performed locally
$T_{\pi^a} = \{\,\}$
$T_{\pi^{*/a}} = \{\,\}$
for all $D \in \{1, ..., D_{const}\}$ **do**
  for all $s \in \{v \in V \mid distance(v, s_p) \leq D\}$ **do**
    $a^* \leftarrow$ most frequent action executed in s
    if $a^* == a$ **then**
      $T_{\pi^a} \leftarrow T_{\pi^a} \cup s$
    else
      $T_{\pi^{*/a}} \leftarrow T_{\pi^{*/a}} \cup s$
    end if
  end for
end for
expected_region $\leftarrow$ describe$(T_{\pi^a}, T_{\pi^{*/a}}, C)$
current_region $\leftarrow$ describe$(\{s_p\}, T_{\pi^a}, C)$
return diff(expected_region, current_region), expected_region

Situational Behavior Characterization Algorithm. Adapted from [1]

Input: Behavioural Model $G = \{V, E\}$, Communicable Predicates P, State region description d, Max number of actions $cluster\_max$
Output: Explanation of policy in d per each action and its corresponding state region
$S \leftarrow dict()$
DNF_description $\leftarrow$ DNF_conversion(d, P)
for all $s \in \{v \in V \mid test\_dnf(v, DNF\_description)\ is\ True\}$ **do**
  $S\left[\pi(s)\right] \leftarrow S\left[\pi(s)\right] \cup s$
  if $|S| > cluster\_max$ **then**
    return "Error: Too many actions"
  end if
end for
for all $a \in S$ **do**
  descriptions $[a] \leftarrow$ describe$(S\left[a\right], V \setminus S\left[a\right], P)$
end for
return descriptions