QoS-aware resource and energy management for autonomous mobile

Dinh Khanh Ho

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Gestion des ressources et de l'énergie orientée qualité de service pour les systèmes robotiques mobiles autonomes

Dinh Khanh HO

CEA-LIST

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur
Dirigée par : Benoît MIRAMOND, Michel AUGUIN
Co-encadrée par : Karim BEN CHEHIDA
Soutenue le : 09 janvier 2020

Devant le jury, composé de :
Stéphane DONCIEUX, Professeur, ISIR
Jean-Philippe DIGUET, Professeur, Lab-STICC
Lionel LAPIERRE, MC-HdR, LIRMM
David FILLIAT, Professeur, U2IS
Karim BEN CHEHIDA, Docteur, CEA-LIST
Benoît MIRAMOND, Professeur, LEAT
Michel AUGUIN, Professeur, LEAT

Gestion des ressources et de l'énergie orientée qualité de service pour les systèmes robotiques mobiles autonomes

Jury :

Président du jury
David FILLIAT, Professeur, U2IS, ENSTA Paris

Rapporteurs
Stéphane DONCIEUX, Professeur, ISIR, Université Pierre et Marie Curie
Jean-Philippe DIGUET, Professeur, Lab-STICC, Université Bretagne Sud

Examinateurs
Lionel LAPIERRE, MC-HdR, LIRMM, Université de Montpellier
Benoît MIRAMOND, Professeur, LEAT, Université Côte d'Azur
Michel AUGUIN, Professeur, LEAT, Université Côte d'Azur
Karim BEN CHEHIDA, Docteur, CEA-LIST

UNIVERSITÉ CÔTE D'AZUR

Abstract

Information Technology Doctoral School

Doctor of Philosophy

QoS-Aware Resource and Energy Management for Autonomous Mobile Robotic Systems

by Dinh-Khanh HO

Mobile robotic systems are becoming more and more complex with the integration of advanced sensing and acting components and functionalities to perform the real required missions. For these technical systems, the requirements are divided into two categories: functional and non-functional requirements. While functional requirements represent what the robot must do to accomplish the mission, non-functional requirements represent how the robot performs the mission. Thus, the quality of service and energy efficiency of a robotic mission are classified in this category. The autonomy of these systems is fully achieved when both functional and non-functional requirements are guaranteed without any human intervention or any external control. However, these mobile systems are naturally confronted with resource availability and energy capacity constraints, particularly in the context of long-term missions, these constraints become more critical. In addition, the performance of these systems is also influenced by unexpected and unstructured environmental conditions in which they interact. The management of resources and energy during operation is therefore a challenge for autonomous mobile robots in order to guarantee the desired performance objectives while respecting constraints. In this context, the ability of the robotic system to become aware of its own internal behaviors and physical environment and to adapt to these dynamic circumstances becomes important.

This thesis focuses on the quality of service and energy efficiency of mobile robotic systems and proposes a hierarchical run-time management in order to guarantee these non-functional objectives of each robotic mission. At the local management level of each robotic mission, a MISSION MANAGER employs a reinforcement learning-based decision-making mechanism to automatically reconfigure certain key mission-specific parameters to minimize the level of violation of required performance and energy objectives. At the global management level of the whole system, a MULTI-MISSION MANAGER leveraged rule-based decision-making and case-based reasoning techniques monitors the system's resources and the responses of Mission Managers in order to decide to reallocate the energy budget, regulate the quality of service and trigger the online learning for each robotic mission.

The proposed methodology has been successfully prototyped and validated in a simulation environment and the run-time management framework is also integrated into our real mobile robotic system based on a Pioneer-3DX mobile base equipped with an embedded NVIDIA Jetson Xavier platform.

Keywords: robotic run-time adaptation, multi-objective decision-making, reinforcement learning, case-based reasoning, artificial intelligence, quality of service, energy management, non-functional requirements, mobile robotics.

Résumé

Gestion des ressources et de l'énergie orientée qualité de service pour les systèmes robotiques mobiles autonomes

Les systèmes robotiques mobiles autonomes deviennent de plus en plus complexes avec l'intégration de composants de capteurs et d'actionneurs et de fonctionnalités avancées pour effectuer les missions réelles. Pour ces systèmes techniques, les exigences sont divisées en deux catégories : les exigences fonctionnelles et les exigences non-fonctionnelles. Alors que les exigences fonctionnelles représentent ce que le robot doit faire pour accomplir la mission, les exigences non-fonctionnelles représentent la façon dont le robot exécute la mission. Ainsi, la qualité de service et l'efficacité énergétique d'une mission robotique sont classées dans cette catégorie. L'autonomie de ces systèmes est pleinement atteinte lorsque les exigences fonctionnelles et non-fonctionnelles sont garanties sans aucune intervention humaine ni aucun contrôle externe. Cependant, ces systèmes mobiles sont naturellement confrontés à des contraintes de disponibilité des ressources et de capacité énergétique, notamment dans le cadre de mission à longue durée, ces contraintes deviennent plus critiques. De plus, la performance de ces systèmes est également influencée par des conditions environnementales inattendues et non structurées dans lesquelles ils interagissent. La gestion des ressources et de l'énergie en cours de mission est donc un défi pour les robots mobiles autonomes afin de garantir les objectifs de performance souhaités tout en respectant les contraintes. Dans ce contexte, la capacité du système robotique à prendre conscience de ses propres comportements internes et de son environnement physique et à s'adapter à ces circonstances dynamiques devient importante.

Cette thèse porte sur la qualité de service et l'efficacité énergétique des systèmes robotiques mobiles et propose une gestion hiérarchique en cours d'exécution afin de garantir ces objectifs non-fonctionnels de chaque mission robotique. Au niveau de la gestion locale de chaque mission, un MISSION MANAGER utilise un mécanisme de prise de décision fondé sur l'apprentissage par renforcement pour reconfigurer automatiquement certains paramètres clés propres à la mission afin de minimiser le niveau de violation des objectifs de performance et des objectifs énergétiques requis. Au niveau de la gestion globale de l'ensemble du système, un MULTI-MISSION MANAGER s'appuie sur des règles de prise de décision et des techniques de raisonnement par cas pour suivre les ressources du système et les réponses des MISSION MANAGERs afin de décider de réallouer le budget énergétique, de régler la qualité du service et de déclencher l'apprentissage en ligne pour chaque mission robotique.

La méthodologie proposée a été prototypée et validée avec succès dans un environnement de simulation et le cadre de gestion est également intégré dans notre système robotique mobile réel basé sur une base mobile de Pioneer-3DX équipée d'une plate-forme embarquée de NVIDIA Jetson Xavier.

Mots clés : robotique mobile, auto-adaptation, prise de décision, apprentissage par renforcement, raisonnement par cas, intelligence artificielle, qualité de service, management d'énergie, exigences non-fonctionnelles.

Declaration of Authorship

I, Dinh-Khanh HO, declare that this thesis titled, "QoS-Aware Resource and Energy Management for Autonomous Mobile Robotic Systems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Karim Ben Chehida and my thesis directors, Benoit Miramond and Michel Auguin, for having entrusted me with this thesis subject and for having guided me during these three years with their great availability and insightful advice. The thesis was not always smooth and many difficult times came, but they were always at my side to encourage me and lead me on the right paths.

I sincerely thank Stéphane Doncieux and Jean-Philipe Diguet for having reported my thesis with very detailed and constructive remarks. I would also like to thank Lionel Lapierre and David Filliat for agreeing to participate in the jury of my thesis defense.

Many thanks to my colleagues at the L3A laboratory who helped me and made me have a good time during my thesis. Thank to Thomas, head of the laboratory, for warmly welcoming me into the laboratory and helping me with many activities during my thesis. Thanks to Maroun for your help in finding a robot experimentation room. Thanks to Philippe for helping me prepare the devices and equipment for my robotic experimentation. I would also like to thank my co-office friends, Michal, Imane and Alix for sharing many funny moments as well as difficulties during my thesis. It is a great pleasure to stay with you in the L3A laboratory after my thesis.

Finally, I would like to give a special thank to my parents, my older brother, for always encouraging me in difficult times. This thesis is also a great gift that I would like to give them.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
CBR	**C**ase-**B**ased **R**easoning
CNN	**C**onvolutional **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
DQL	**D**eep **Q**-**L**earning
DQN	**D**eep **Q**-**N**etwork
DRL	**D**eep **R**einforcement **L**earning
LIDAR	**L**aser **I**maging **D**etection **A**nd **R**anging
MAPE	**M**onitoring, **A**nalyzing, **P**lanning, and **E**xecuting
MAPE-K	**M**onitoring, **A**nalyzing, **P**lanning, and **E**xecuting - **K**nowledge
MODM	**M**ulti-**O**bjective **D**ecision-**M**aking
MDP	**M**arkov **D**ecision **P**rocesses
MLP	**M**ulti-**L**ayer **P**erceptron
NFR	**N**on-**F**unctional **R**equirement
ODA	**O**bserve, **D**ecide, and **A**ct
QoS	**Q**uality **o**f **S**ervice
RL	**R**einforcement **L**earning
ROI	**R**egion **O**f **I**nterest
ROS	**R**obot **O**perating **S**ystem
RTA	**R**un-**T**ime **A**daptation
SAS	**S**elf-**A**daptive **S**ystems
SGD	**S**tochastic **G**radient **D**escent
SOCh	**S**tate **O**f **Ch**arge

Dedicated to my family…

Chapter 1

Introduction

Contents

The first chapter of our thesis is organized as follows. The general context of our research is presented in Section 1.1. In Section 1.2, the research problem is stated with the necessary terminologies and three main research questions. The overview of our approach and contributions is then described in Section 1.3. Section 1.4 ends this chapter by introducing the structure of our thesis.

1.1 General Context

Autonomous mobile robotic systems are continually developing and growing. Today they are applied to a broad range of real life missions such as security surveillance, environmental monitoring, delivery, search and rescue missions, etc. These autonomous mobile systems can be deployed in many domains such as service, road, field, air, marine and space [Kunze et al., 2018] (Figure 1.1):

- **Service robotics**: relate to the robots that work in a house, in an office, in a hospital, in a museum, etc to help or provide people the information in their daily tasks. Some examples of these robots are PR2 robot[1], NAO humanoid robot[2] and STRANDS robot[3];
- **Autonomous vehicles**: are deployed on the road with a known example of self-driving cars such as Waymo car[4] and EZ10 shuttle[5];

[1]See http://www.willowgarage.com/pages/pr2/overview
[2]See https://www.softbankrobotics.com/emea/en/nao?q=emea/fr/nao
[3]See http://strands.acin.tuwien.ac.at/
[4]See https://waymo.com/
[5]See https://easymile.com/application-map-easymile/

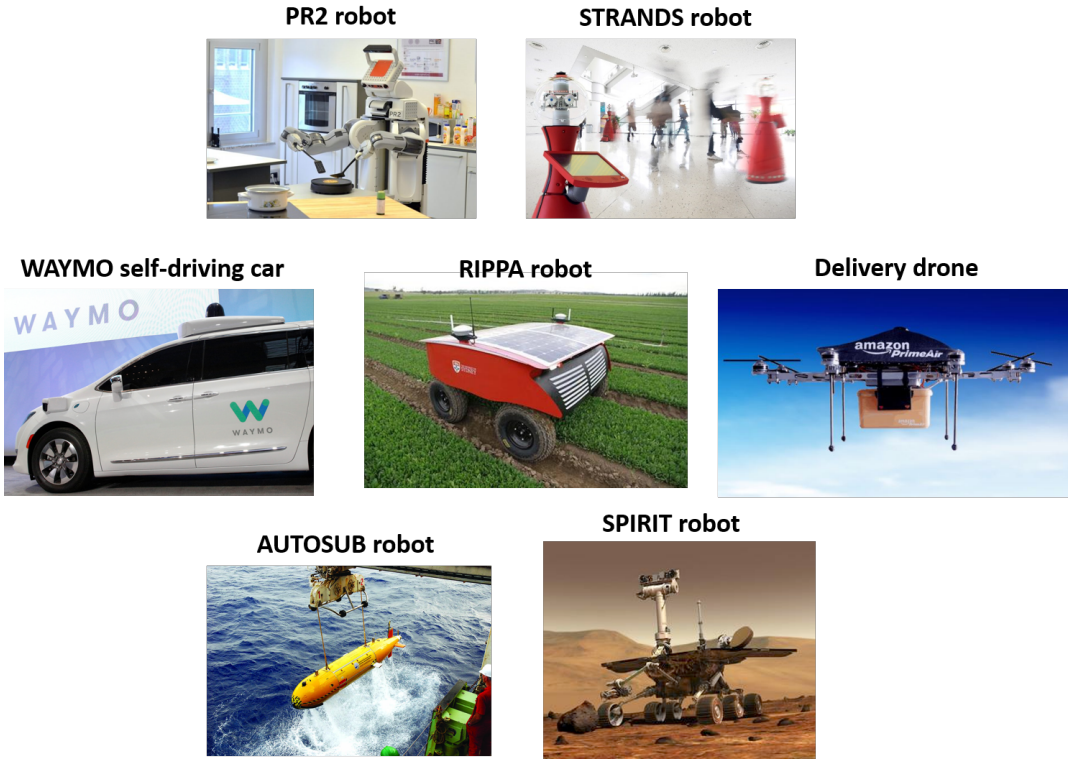


FIGURE 1.1: Example of mobile robotic systems in the domains of service (PR2 and STRANDS robot), road (WAYMO self-driving car), field (RIPPA robot), air (delivery drone), marine (AUTOSUB robot) and space (SPIRIT robot).

- **Field robotics**: are the robotic systems working in construction, agriculture, mining, etc such as RIPPA robot[6];
- **Aerial robotics**: mean the robots that can fly in the air such as Parrot drones[7];
- **Marine robotics**: are underwater or surface vehicles such as Autosub robot[8];
- **Space robotics**: mean the robotic systems used to explore the space such as Spirit robots[9].

For each robotic domain, the level of autonomy required is different due to the different functionalities and operational environments. For example, for marine and space robotics, the communication requirement, operating time and capability of autonomously returning to the docking station to be recharged should be prioritized, while for self-driving cars, the safety requirement should first be emphasized to avoid accidents with surrounding vehicles and pedestrians. Indeed, a vast amount of research and effort of the robotics community has concentrated on developing the robotic functionalities, the application algorithms such as visual perception, mapping, path planning, path following, etc for guaranteeing the autonomy of these systems [Sprunk, 2015; Chikurtev, 2015; Abdulmajeed and Mansoor, 2014]. However, in the context of mobile robotics, two important aspects such as resource constraints

[6]See https://sydney.edu.au/news-opinion/news/2015/10/21/rippa-robot...
[7]See https://www.parrot.com/fr/drones
[8]See https://noc.ac.uk/facilities/marine-autonomous-robotic-systems/autosubs
[9]See https://www.nasa.gov/mission_pages/mer/index.html

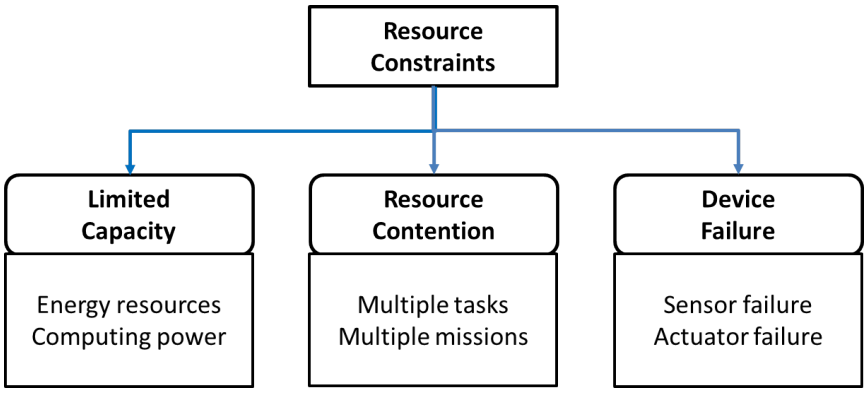


FIGURE 1.2: Resource constraints in mobile robotics: limited capacity of computing and energy resources, resource contention between robotic missions, and device failure such as sensors, actuators' failure.

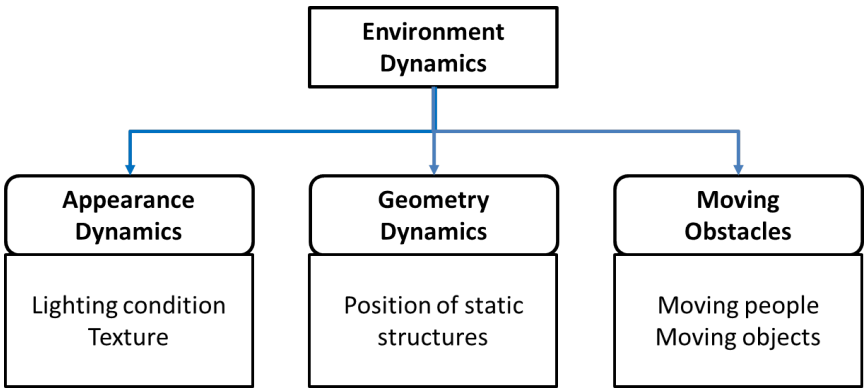


FIGURE 1.3: Environment dynamics in mobile robotics: appearance dynamics such as changing lighting condition or texture, geometry dynamics such as changing position of static structures, and moving obstacles.

and environment dynamics can unexpectedly influence the robotic functional autonomy:

- **Resource constraints in mobile robotics**. The first aspect relates to the fact that the mobile systems can carry on limited and/or unreliable resources and energy capacity (Figure 1.2). The inefficient utilization of these resources can significantly degrade the system performance even to the point that the autonomous functionalities cannot be completed. The limited onboard computing resources may also require the installation of a remote computing workstation or require research on cloud robotics where the robots use the computing services of cloud platforms when the onboard computing resources cannot meet the heavy computing demands [Li et al., 2016; Hu et al., 2017]. Another approach tackling the limitation of resources of a single robot is to deploy robot swarms and resolve the multi-robot collaboration problem for accomplishing the missions [Yan, Jouandeau, and Cherif, 2013; Lee, Tarokh, and Cross, 2010]. In addition, in the context of multiple tasks or missions on the same robot, these limited resources are shared and the resource conflicts between them can occur. Device failure is another important issue for robotic resources, sensors and actuators. This issue leads to another research branch to guarantee the fault tolerance or to make the robotic system more dependable [Gspandl et al.,

2012; Cui et al., 2014; Guiochet, Machin, and Waeselynck, 2017; Hireche et al., 2018];

- **Environment dynamics in mobile robotics**. The second aspect influencing the robotic autonomy is that the mobile robots operate mostly in unstructured and unknown a priori environments (Figure 1.3). The environment dynamics can be divided into three main categories as proposed by the work in [Sprunk et al., 2014]: appearance dynamics such as lighting conditions and texture, geometry dynamics such as position of (static) objects, and moving obstacles such as moving people and other moving objects. These variations of the environment can lead to unpredictable behaviors of the robotic systems [Brugali, Capilla, and Hinchey, 2015].

Consequently, a question arises in the mobile robotic context: **How are these robotic functionalities performed?** This is another aspect of technical systems called **non-functional properties** or **quality models** that are defined by the standard ISO 25010[10]. Two main models are defined by this standard:

- **Product quality model**: means the intrinsic quality characteristics of the product (software or system). Eight product quality properties proposed by the ISO 25010 such as functional suitability, performance efficiency, reliability, security, usability, maintainability, compatibility and portability are shown in Figure 1.4;
- **Quality in use model**: means the quality characteristics of the product (software or system) when it is used under a specific context. Five quality in use properties such as effectiveness, efficiency, satisfaction, freedom from risk and context coverage are depicted in Figure 1.5.

While the product quality model is generally used to characterize and evaluate the quality of the software or system at the design time before using it in real life, the quality in use model is used to characterize and evaluate at the execution time. In this case, the system may face contexts that were not anticipated at the design

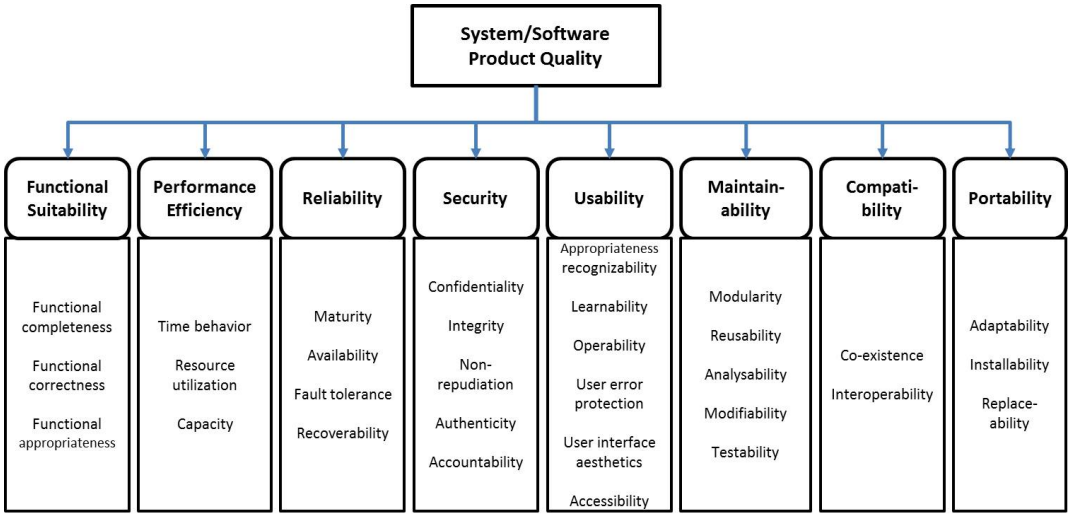


FIGURE 1.4: Product quality model proposed by the ISO 25010. It evaluates the intrinsic quality characteristics of the software or system.

[10]See https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

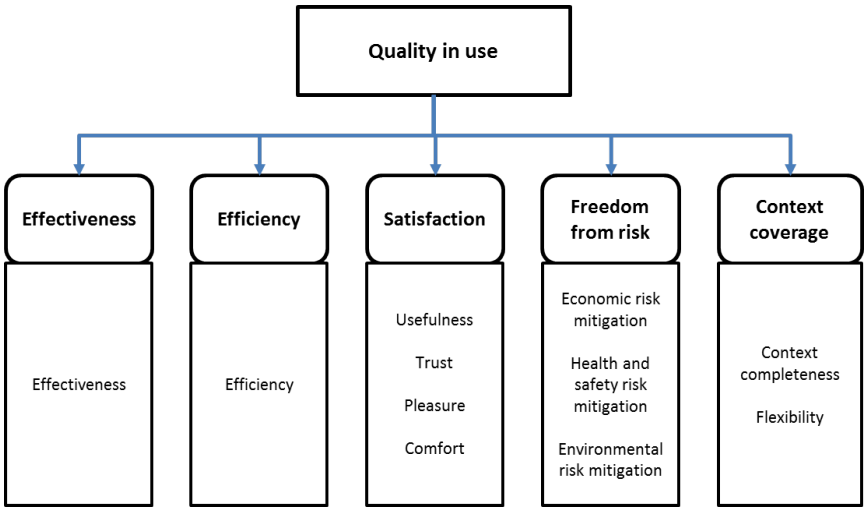


FIGURE 1.5: Quality in use model proposed by the ISO 25010. It evaluates the quality characteristics of the system or software when it is used under a specific context.

time. This quality model is therefore really our concern when we consider the run-time execution of mobile robotic missions. The definition of each characteristic in the quality in use model can be explained as follows:

- **Effectiveness**: evaluates the ability of a system or software to produce the intended or expected result. This property generally concerns the functionalities of the system or software;
- **Efficiency**: evaluates the time behavior and the resource utilization such as computing power and energy of a system or software to produce the expected result;
- **Satisfaction**: represents the user's experiences when using a system or software such as a graphical user interface and easy interaction;
- **Freedom from risk**: indicates the ability of a system or software to mitigate potential risks to financial status, people and environment;
- **Context coverage**: represents the extent to which a system or software can be used with effectiveness, efficiency, freedom from risk and satisfaction in both intended and unintended contexts of use.

In our research, we focus on the effectiveness, efficiency and context coverage of these quality characteristics to deploy the run-time adaptation (RTA) of mobile robotic missions. As with functional requirements, a set of non-functional requirements (NFRs) can be applied to the robotic missions. The two dynamic factors mentioned above such as resource constraints and environment dynamics also have an impact on these requirements.

Thus, the dynamic circumstances in the context of mobile robotics require the adaptive capabilities of robots to successfully accomplish missions taking into account desired performance objectives and constraints while minimizing human intervention (inversely proportional to the level of autonomy), particularly in restricted access environments, as well as in long-term operation. The ability of robotic systems to adapt their configuration and behavior at the execution time is therefore

called **robotic run-time adaptation** and can be considered as a subfield of **self-adaptive systems** [Macías-Escrivá et al., 2013]. Indeed, self-adaptive systems are a promising approach for systems that have capability of:

- **Self-awareness**: being aware of their internal and external contexts;
- **Self-optimization**: reasoning and making decisions based on their current operational conditions;
- **Self-reconfiguration**: interpreting given decisions to reconfigure themselves.

The prefix "self" means that the system can do itself without any human intervention or external control, and this is also how the **robotic autonomy** is defined. Based on this general context, we will then state our research problem in the next section.

1.2 Problem Statement

1.2.1 Terminologies

We provide in this section some important terminologies to facilitate the problem statement. Figure 1.6 indicates the main concepts and their relationships in a robotic system. We have a **mobile robot** that interacts with its **physical environment**. A **robot user** identifies **functional requirements** and deploys the **mission** to fulfill them. Based on these functional requirements, the robotic mission will determine what to do and use the **robot's resources** such as processing applications (or software), sensors, actuators and energy to perform the required functionalities. In addition to functional requirements, the robot user can also identify **non-functional requirements** applied to the robotic mission to determine how the robotic mission should perform functionalities. These requirements can also be identified by the

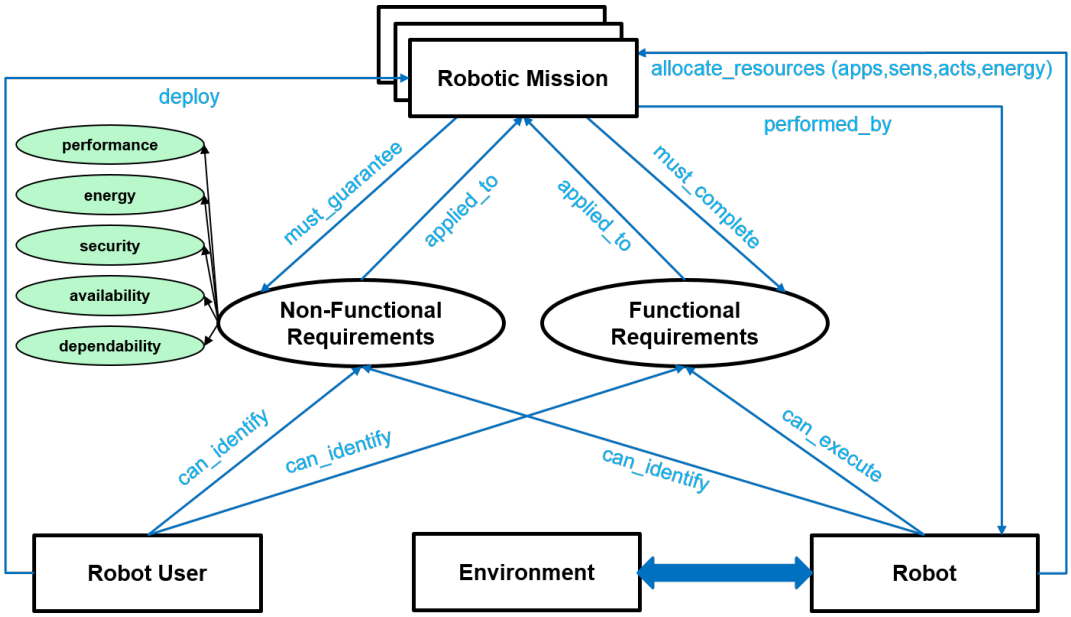


FIGURE 1.6: Relationships and roles of the concepts of robotic mission, robot user and robot. The full autonomy is acquired when the robotic mission can complete all the functional requirements while guaranteeing the non-functional requirements.

robot's constraints such as current battery state of charge or current state of computing resources. In our research, we consider the mission's quality of service and energy efficiency as non-functional requirements. The full autonomy is achieved when the robotic mission can complete all functional requirements while guaranteeing the non-functional requirements. And a **multi-mission context** means that there are several robotic missions deployed in the same time with different functional requirements and priorities on the same robotic platform.

1.2.2 Research Questions

This thesis focuses on the quality of service and energy management to enhance the autonomy of mobile robotic systems. The problem is stated by three following research questions (RQ):

- **RQ1: How does a robotic mission control its configuration and/or its behavior in order to guarantee explicitly a desired set of non-functional requirements under dynamic operational conditions?** A robotic mission is an abstract concept in our framework. It is deployed to achieve certain desired functional objectives. For each mission, a set of non-functional objectives such as performance and energy efficiency can be applied and the robotic mission must guarantee them in order to reinforce the autonomy. However, the operational conditions of mobile robotics, such as environmental conditions and resource availability, which are inherently dynamic and unstructured, make management more difficult;

- **RQ2: How does a robotic mission adapt to changing non-functional requirements?** While the robotic mission must face dynamic operational conditions to ensure the desired non-functional requirements, these requirements can be dynamically modified by the robot user or by the robot itself. For example, depending on the current level of the robot's battery, the mission's energy consumption target may be dynamically modified. This question therefore poses another challenge for the robotic mission;

- **RQ3: How does a robotic system guarantee the mission's quality of service and system energy constraints in a multi-mission context?** In a robotic system equipped with many advanced sensing, acting and computing components, not a single mission will be deployed, but several missions with different functionalities and priorities can be performed at the same time. Hence, the multi-mission context is another research issue with the challenge of ensuring the quality of service of each mission under system-level constraints.

1.3 Overview of Our Approach and Contributions

In addition to the research questions mentioned above, this section summarizes our approach and scientific contributions. Our approach is based on three main points. The first is the characterization of mobile robotic missions, which helps to understand the dynamic characteristics of mobile robotic missions and to be considered as a preliminary step towards solving the research questions. The second is considered as a local management level to each mission based on a reinforcement learning to solve the RQ1 and RQ2 research questions. And the last is a system-level management with a rule-based decision-making mechanism and a case-based reasoning

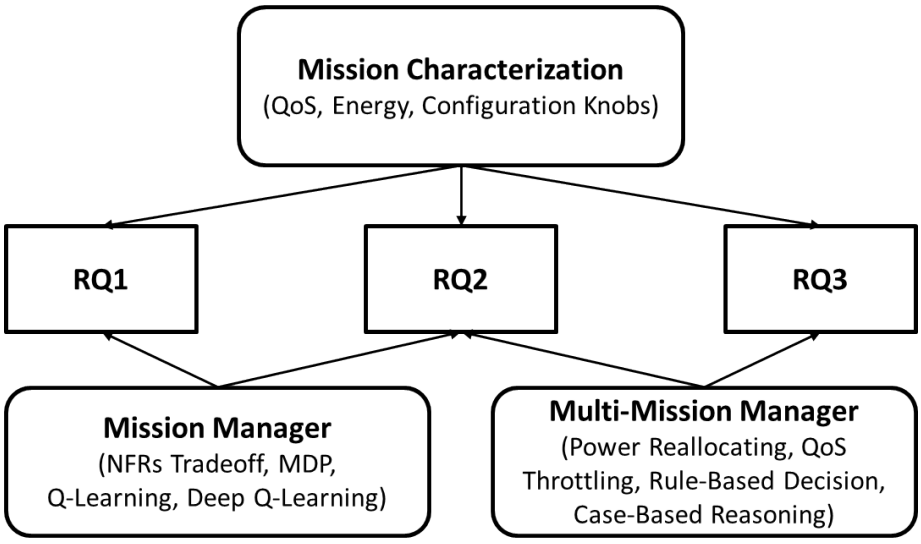


FIGURE 1.7: Overview of our approach and contributions.

technique to resolve the RQ2 and RQ3 research questions. Figure 1.7 summarizes the overview of our approach and contributions, and our scientific publications resulting from this research work are presented in Appendix E.

1.3.1 Robotic Mission Characterization

This point proposes our unified concepts to characterize mobile robotic missions. Among the concepts proposed, we highlight the one of the robotic mission's quality of service (QoS), energy consumption and dynamic configuration knobs. In the literature, a unified concept for mobile robotic missions has not been defined. We also propose a systematic approach to generalize the characterization for different robotic missions. The results of the characterization process of a robotic mission are known as a knowledge base of this mission. These concepts also support the run-time adaptation framework we propose.

1.3.2 QoS and Energy-Aware Self-Adaptive Mission Manager

The Mission Manager represents the local management at the mission level and strives to ensure a set of desired non-functional requirements (NFRs) with an emphasis on quality of service and energy efficiency objectives. The Mission Manager's run-time decision-making problem is modelled by Markov Decision Processes (MDPs). A reinforcement learning based approach is then proposed to resolve these MDPs. Two algorithms called Q-Learning and Deep Q-Learning have been implemented and validated. The results show that the Mission Manager is able to guarantee desired non-functional requirements under dynamic operational conditions. The ability of online learning to address more dynamic and unpredictable factors is also a highlight of our proposed approach.

1.3.3 Adaptive and Hierarchical Multi-Mission Manager

In the context of multiple robotic missions, many missions are performed at the same time and share system resources with limited capabilities. We propose a rule-based decision-making approach and a case-based reasoning technique to guarantee

the quality of service of each robotic mission while respecting global system constraints. Moreover, by evaluating the adaptation response of each Mission Manager, the Multi-Mission Manager can decide to launch the online learning to evolve the adaptation model or to generate a new adaptation model for each robotic mission.

1.4 Thesis Structure

The sections above provided the general context for our research. The problem is also outlined in the three research questions and the three main contributions have been identified to answer these questions. The rest of the thesis is organized as follows:

- **Chapter 2** provides certain theoretical background as well as the state of the art underlying our research. The background is articulated around three main axes: mobile robotic systems, self-adaptive systems and artificial intelligence decision-making. Finally, a discussion on the relationship with this thesis is presented;
- **Chapter 3** presents the methodology for characterization and monitoring of mobile robotic missions. Some related work is first mentioned. Next, we present the concept and systematic approach we propose to characterize the missions. The monitoring methodology is also discussed. To validate our proposed methodology, we also describe our simulation and real frameworks. Some examples of mobile robotic missions are presented at the end of this chapter and the motivation for run-time adaptation is also highlighted;
- The reinforcement learning approach for the Mission Manager is presented in **Chapter 4**. The problem of quality of service and energy management for mobile robotic missions is formally defined, and two learning algorithms called Q-Learning and Deep Q-Learning have been applied to this problem. The validation is realized with some case studies in the simulation framework. We then give an in-depth analysis based on the obtained results;
- **Chapter 5** addresses the context of multiple robotic missions. The methodology based on rule-based decision-making and case-based reasoning techniques is presented and discussed. A simulated scenario is implemented and the efficiency of our proposed methodology is rigorously analyzed;
- **Chapter 6** describes the implementation and validation of the methodology presented in this thesis in our real robotic framework. The performance of Mission Manager and Multi-Mission Manager is evaluated and discussed. And we discuss also our contribution to long-term robotic autonomy in real life;
- **Chapter 7** presents the general conclusions of this thesis as well as some perspectives;
- Finally, **Appendices A, B, C, D and E** will provide some additional information for our thesis.

Chapter 2

Research Background

Contents

This chapter discusses the basic background that underlies our research. The topics presented are multidisciplinary and span from mobile robotics, self-adaptive systems to artificial intelligence with an emphasis on the decision-making techniques. The overview and autonomy requirements of mobile robotic systems are mentioned in Section 2.1. The definition of self-adaptive systems and their taxonomy are presented in Section 2.2. The engineering approaches for planning and decision-making

in self-adaptive systems are also discussed in this section. Sections 2.3, 2.4, 2.5 detail some specific planning and decision-making techniques and they are also the core of our methodology. In Section 2.6, some previous work on performance and energy aware run-time adaptation for mobile robotic systems will be presented and discussed. Finally, we conclude this chapter in Section 2.7.

2.1 Mobile Robotic Systems

Mobile robotic systems are defined as systems capable of locomotion. They can become autonomous in unstructured environments by integrating a feedback loop of perception, control and action [Veres et al., 2011]. Some examples of autonomous mobile robotic systems are presented in Section 1.1. This section aims to provide an overview of the generic structure of mobile robotic systems and their autonomy requirements. The role of artificial intelligence for the robotic autonomy is also discussed.

2.1.1 Overview

Generic Architecture of Mobile Robotic Systems

Figure 2.1 describes the generic structure of an autonomous mobile robotic system. We divide it into three subsystems as follows:

- The first one is named the **sensing and acting subsystem**, and is composed of actuators, sensors and low-level embedded controllers that control and communicate with sensors and actuators;
- The second one is the **computing subsystem**, which includes high-level computing resources such as computing core, memory, and software;
- The last one is the **power supply subsystem**, which is considered to be an energy resource, and powers the two above subsystems.

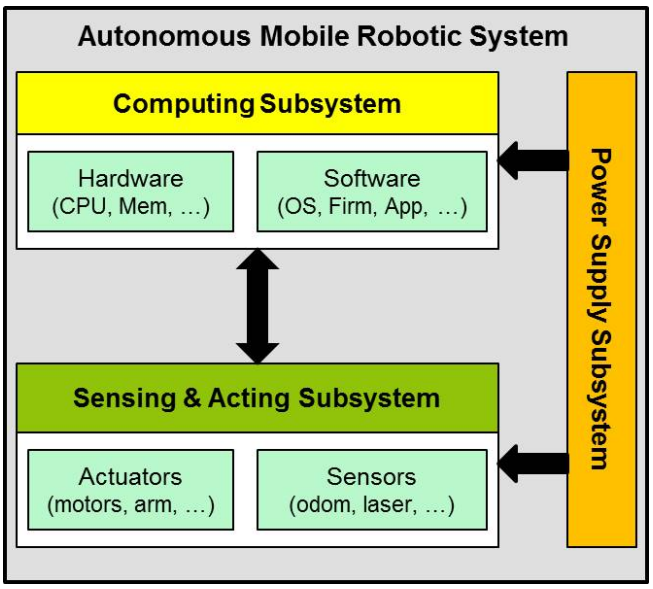


FIGURE 2.1: Generic structure of mobile robotic systems. A mobile robotic system is composed of three main subsystems: sensing and acting, computing and power supply subsystems.

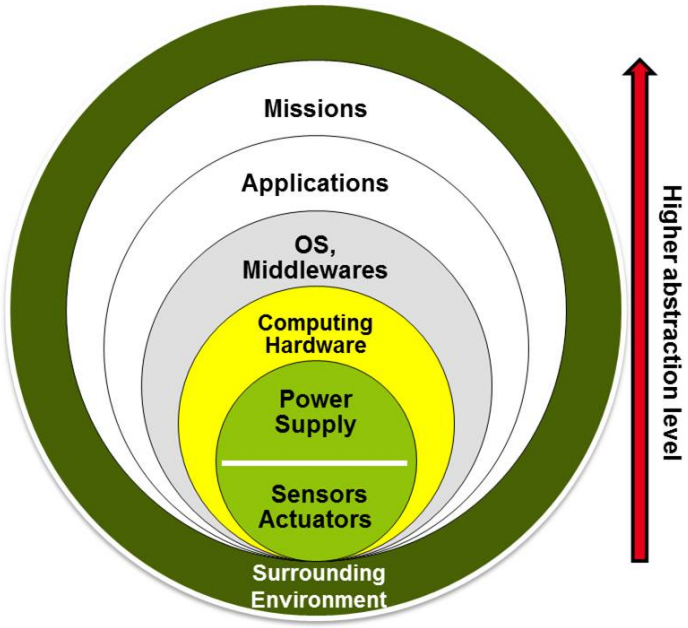


FIGURE 2.2: Abstraction level of mobile robotic systems in which the mission is at the highest level and the sensors and actuators are at the lowest one. All the system interacts directly with the surrounding physical environment.

These are the three preliminary components that make a mobile robotic system functional. The abstraction level of the robotic system is proposed in Figure 2.2 in which the robotic mission is at the highest level and it encapsulates the complex details of the robotic systems. The complexity of these systems is scaled depending on the number of components in each subsystem, from miniature mobile robots with some sensors and motors, and an embedded microcontroller-based board such as Khepera mobile robots with a diameter of $55mm$ [Floreano et al., 1999], or nano/pico aerial vehicles in Navion project[1] to autonomous cars with dozens of sensors and actuators, and onboard heterogeneous computing resources [Kato et al., 2015].

Robotic Software Systems

The survey in [Ahmad and Babar, 2016] proposed a reference model for the robotic software systems as shown in Figure 2.3. In this model, two main layers are defined:

- The **control layer** is composed of a set of drivers to interact with robotic components such as sensors and actuators. Refer to our generic structure presented in Figure 2.1, this layer is really included in the sensing and acting subsystem;
- The **application layer** takes care of the more complex functions of a robot and uses the control layer to support robotic operations. This layer is therefore included in the computing subsystem of our generic structure.

Ahmad and Babar also surveyed three main research themes on this application layer or the software architecture for robotics:

- **Robotic Modeling, Design and Programming**: This theme is known as how to design, model and program a robotic system, and this problem is still dominant in the robotics research community;

[1] See http://navion.mit.edu/

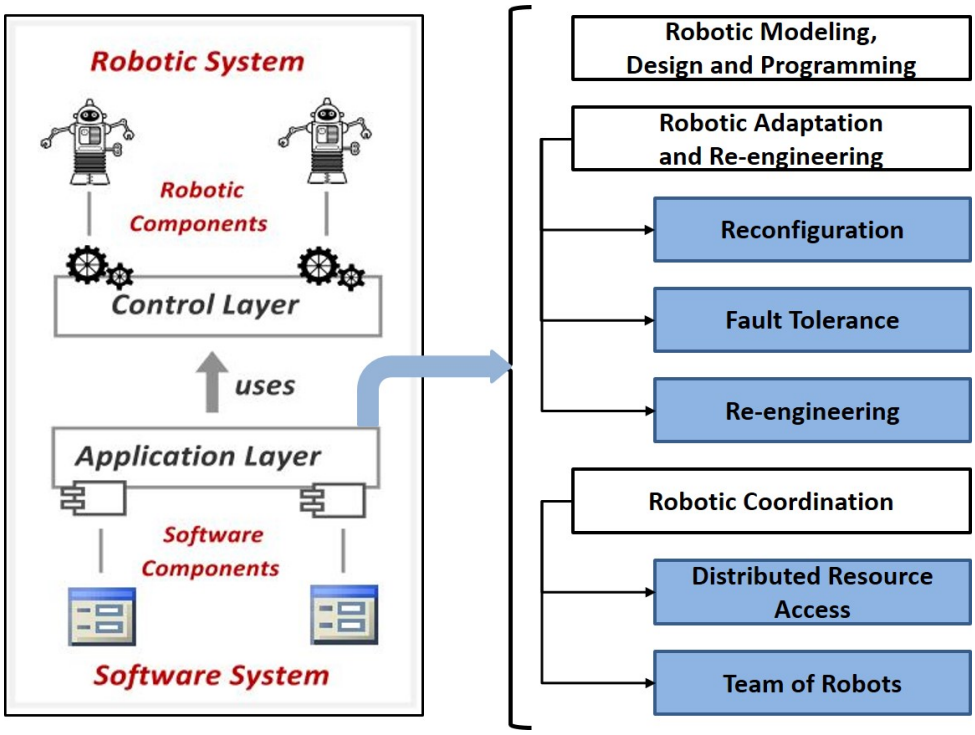


FIGURE 2.3: A reference model for robotic software systems (adapted from [Ahmad and Babar, 2016]). The three main research themes are identified: robotic modeling, design and programming, robotic adaptation and re-engineering, and robotic coordination.

- **Robotic Adaptation and Re-engineering**: This theme includes three sub-themes such as reconfiguration, fault tolerance and re-engineering. *Reconfiguration* capability means how to reconfigure the structure or behaviour of a robot so that it can adapt to many dynamic operating conditions. *Fault tolerance* capability allows the robot to continue its operations despite the presence of a failure. *Re-engineering* capability makes an existing robot evolve into a new version that better meets new requirements;
- **Robotic Coordination**: This theme relates to the problem of distributed resource access of a robotic system or the problem of coordinating a team of robots.

In fact, the second theme (robotic adaptation and re-engineering) guides our research. The objective is to allow a mobile robotic system to monitor and analyze itself its current operational conditions and dynamically modify its configuration and behavior to meet certain non-functional requirements.

Robot Operating System

Mobile robotics are systems-of-systems with many hardware and software components that communicate with each other. This complexity poses the question of system-level integration. The Robot Operating System (ROS) [Quigley et al., 2009] has a strong community support[2] and provides a common open-source framework

[2]See http://www.ros.org/

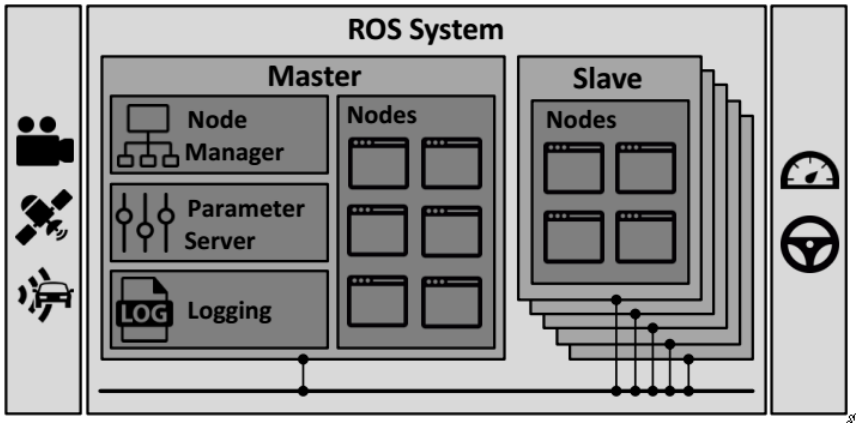


FIGURE 2.4: ROS system architecture (reprinted from [Hellmund et al., 2016]). Each hardware and software component in the system is managed by ROS nodes and all the nodes are managed by a central ROS master. After registering with ROS master, the communication between ROS nodes is direct.

dealing with this integration question in a component-based, modularized and distributed manner. In this framework, the components can be implemented or managed in the form of ROS nodes. These nodes can be launched in a distributed network of computing machines and communicate via a message-passing mechanism such as topics and services. Indeed, this publish-subscribe communication mechanism really facilitates the reusability of the robotic components.

The ROS-based system architecture is presented in Figure 2.4 with the connection between a central ROS master and many slave ROS nodes. The ROS nodes are usually configurable with a rich set of parameters, among which, some parameters can be dynamically reconfigured with the support of a *dynamic_reconfigure*[3] ROS package, and this dynamic reconfiguration of the parameters does not normally lead to service interruption.

2.1.2 Autonomy of Robotic Systems

The mobile robotic systems with the autonomy capability can act in real world environments, usually complex and changing environments for a prolonged time without any form of external control [Bekey, 2005; Tessier, 2017; Veres, 2011]. These systems must perceive their surrounding environment and their own internal behaviors, then with a decision-making or planning phase, give the appropriate reactions in order to keep the robotic system in a feasible context until the required missions are accomplished. By default, the autonomy of mobile robotic systems usually implies the autonomous functionalities, where their generic control pattern can be presented in Figure 2.5. The work in [Veres et al., 2011] surveyed and discussed the intelligent decision-making and control techniques for guaranteeing the functional autonomy. The recent innovations in sensors technology, as well as in perception technology such as computer vision, data fusion capabilities permit the mobile robots representing and understanding any dynamic and complex environments. This contributes also to the functional autonomy of these systems. Nevertheless the dynamic circumstances of onboard resources such as computing power and energy

[3] See http://wiki.ros.org/dynamic_reconfigure

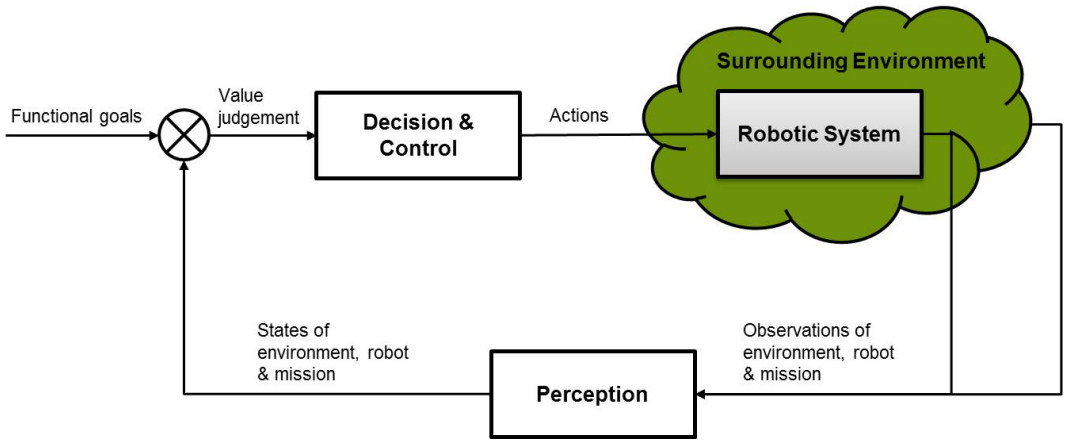


FIGURE 2.5: Autonomy for functional goals with the capability of perception and intelligent decision-making & control.

source, and unknown a priori environmental conditions can influence significantly these autonomous functionalities, even they cannot be correctly completed.

For example, the mobile robot needs a recognition task for analyzing and avoiding possible front obstacles during a specific period, but, at this moment, the onboard computing platform cannot meet this workload because of the heavy computing demands of other functionalities or other software modules, so the recognition task cannot finish before colliding with obstacles. Otherwise, if the robot is in a free space without obstacles, this recognition task should only be allocated limited computing resources and energy in order to guarantee the efficient utilization of these resources. Guaranteeing performance and energy is therefore an important challenge for enhancing the autonomy of robotic systems.

2.1.3 Artificial Intelligence for Robotic Autonomy

Artificial Intelligence (AI) is defined as the science and engineering of making intelligent machines. These machines can perceive the environment around them, in some cases behave as humans and take actions to maximize their chances of achieving their goals. AI has many applications in the field of healthcare, automotive, robotics, finance and economics, art, etc. For robotics, Kunze et al. [Kunze et al., 2018] indicate that some sub-disciplines of AI can help improve the robotic autonomy, including navigation & mapping, perception, knowledge representation & reasoning, planning, interaction and learning (Figure 2.6):

- **Navigation and Mapping** allow the mobility of mobile robots to work efficiently in unstructured and dynamic environments;
- **Perception** enables the robot to perceive and understand the scene. The perceived information is then used for robot navigation and mapping on the one hand and for semantic understanding of the scene, for example, object detection and recognition on the other hand;
- **Knowledge Representation and Reasoning** allow the robot to represent various aspects of the world and reason about them;

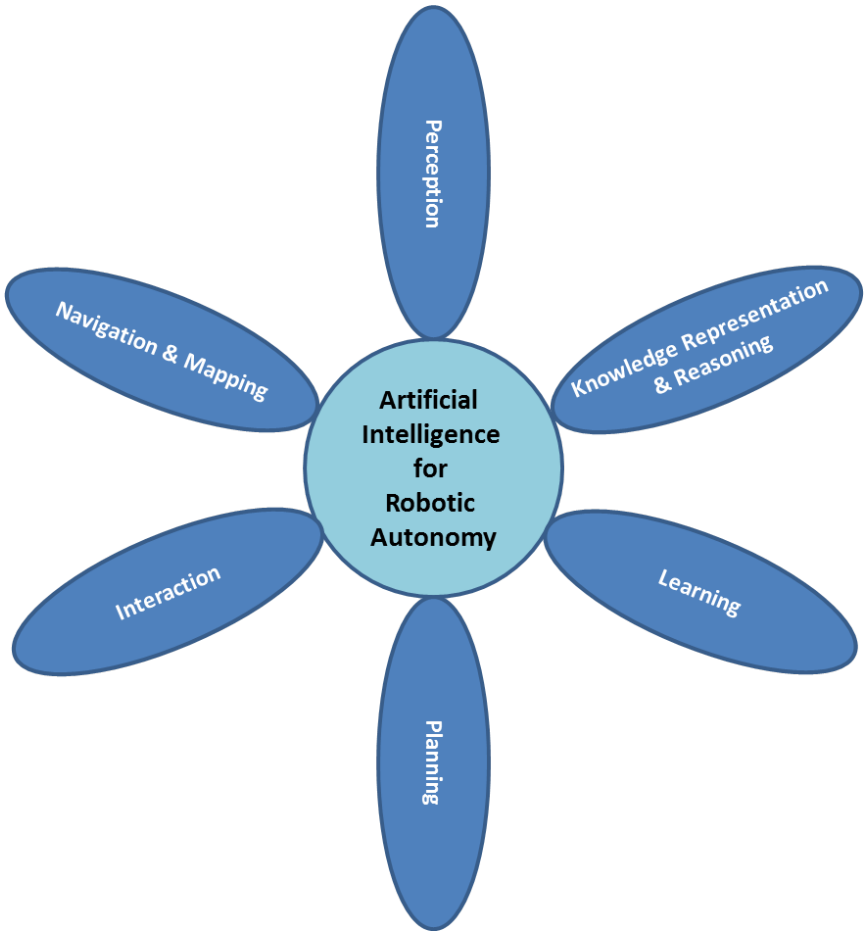


FIGURE 2.6: Some sub-disciplines of Artificial Intelligence improve the robotic autonomy.

- **Planning** concerns the execution of strategies or sequences of actions for the robot. AI planning normally focuses on the decision-making problem to decide, at a given robot's state, which action will be executed in order to achieve desired goals;
- **Interaction** allows the robot to communicate with other agents in its world such as humans or other robots;
- **Learning** allows the robot to make predictions or decisions based on sample data or experiments without being explicitly programmed. In fact, the learning technique can be used for the five topics above.

Our thesis focuses on the quality of service and energy management for the robotic systems and we identify the planning, decision-making and learning techniques in artificial intelligence that are at the heart of the methodology of system management we propose.

2.2 Self-Adaptive Systems

We seek to highlight some basic concepts of self-adaptive systems (SAS) that will be applied to discuss the state of the art robotic run-time adaptation. We mainly present the definition, taxonomy and planning strategy of these systems.

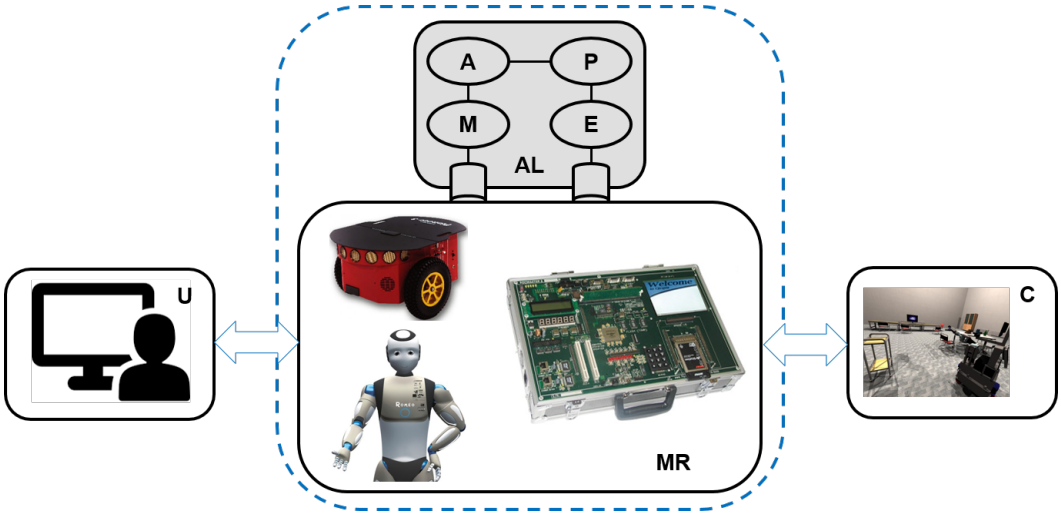


FIGURE 2.7: An illustration of self-adaptive systems (AL = Adaptation Logic, MR = Managed Resources, U = User(s), C = Context, M = Monitoring, A = Analyzing, P = Planning, E = Executing), adapted from [Krupitzer et al., 2015]. Its applications can be found in the mobile, embedded, pervasive computing systems, or in the robotic domain, etc.

2.2.1 Definition

The work in [Macías-Escrivá et al., 2013] defines **self-adaptivity** as the capability of the system to adjust its behavior in response to the operational environment. The "self" prefix means that the decision-making process is autonomous with minimal or without any form of external interventions. Figure 2.7 illustrates the concept of a self-adaptive system with connections between four main elements:

- **Users** (U) can be end-users or system operators;
- **Managed Resources** (MR) are the systems under consideration;
- **Context** (C) implies the surrounding operational environment;
- **Adaptation Logic** (AL) representing adaptivity processes.

The adaptation logic is usually represented by a known architecture composed of *Monitoring*, *Analyzing*, *Planning* and *Executing* elements (**MAPE** paradigm) with the following roles:

- **Monitoring** element collects the information from the environment and system through a (physical and/or virtual) sensors network, as well as goals specification and system requirements;
- **Analyzing & Planning** elements, also considered as decision-making phase, analyze the monitored metrics to identify the current state, reason and plan the adaptation decisions adaptive to this state;
- **Executing** elements receive the adaptation decisions from Planning element and reconfigure the managed system.

2.2.2 Taxonomy of Self-Adaptation

The work in [Krupitzer et al., 2015] introduces a generic taxonomy of self-adaptation for surveying the engineering approaches in the SAS domain. Figure 2.8 indicates the proposed taxonomy with the following main concepts:

- **Reason: Why do we have to adapt?** Basically, the self-adaptation deals with the changing circumstances caused by the change in the environmental conditions, dynamic resource availability and change caused by the users;
- **Level: Where do we have to implement change?** In a multi-layer system like robotic systems or cyber-physical systems, the adaptation can occur in a single layer, many layers or cross-layer;
- **Time: When should we adapt?** We can adapt before the change occurs following a prediction phase by the proactive approach, or adapt after the change occurs by the reactive one;
- **Technique: What kind of change is needed?** The adaptation decision can be to reconfigure the parameters of applications, hardwares, peripheral devices, etc, the system structure to combine hardware and software, or even in some cases, to reconfigure the system context;
- **Adaptation control: How to adapt?** Two approaches can be used for the adaptation logic. The internal approach intertwines the adaptation logic with the system resources and the external one isolates the adaptation logic and the managed resources. The adaptation decision can be based on the criteria such as models, rules, goals or utility. The decision plans can be determined in an offline, online or hybrid manner. More details on this concept will be presented in the following part. Finally, the adaptation control can be deployed in a centralized, decentralized or hybrid manner.

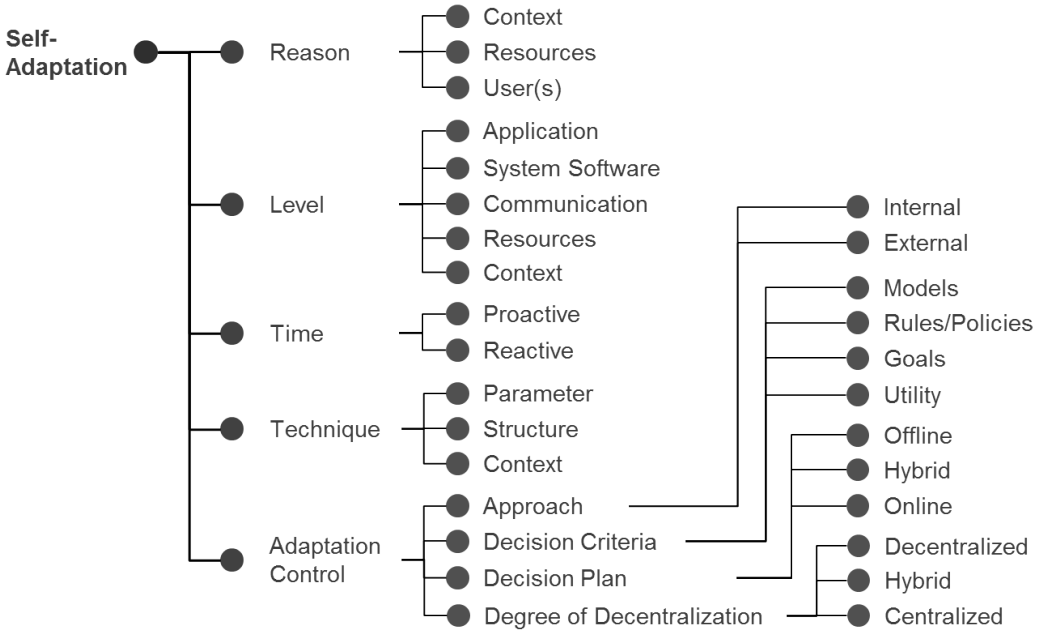


FIGURE 2.8: Taxonomy of self-adaptation (adapted from [Krupitzer et al., 2015] with an addition of *Decision Plan*).

2.2.3 Planning Strategy

Another concept associated with self-adaptive engineering that we would like to highlight is the planning strategy. This refers to the way in which the decision or adaptation plans are determined. There are three approaches: *offline planning*, *online planning* and *hybrid planning*. We summarize the qualitative assessment of these approaches in three dimensions: the inverse of costs such as time and computing overhead, reliability such as the probability of convergence of algorithms and the optimal response to dynamic run-time effects, as shown in Figure 2.9:

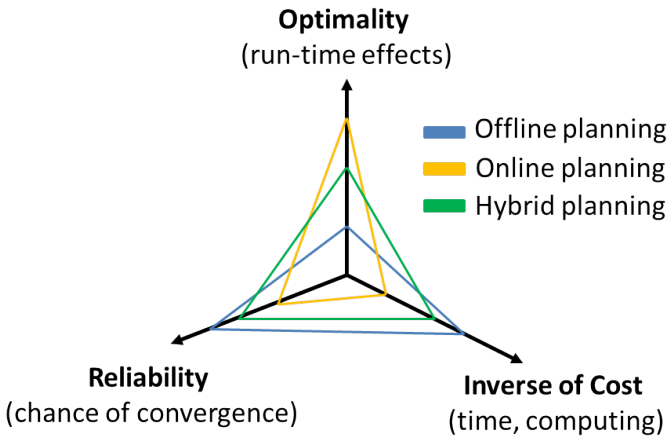


FIGURE 2.9: Planning strategy in self-adaptive systems. The three strategies are evaluated in terms of optimality, reliability and computing cost.

- The **offline planning** means that the adaptation plans are determined in the design time and are kept static during run-time. This approach has the advantage of reliability, it is easily understood and easy to implement in terms of time and computing overhead [Zhao et al., 2017]. However, some unforeseen run-time operational circumstances can be undefined in the predetermined adaptation plans, which causes a question of optimality and of handling run-time effects;
- Inversely, the adaptation plans in the **online planning** are determined at run-time based on the past experiences or events of the system, and usually by the optimization and online learning process [Kim and Park, 2009]. The online approach can give the optimal adaptation plans that efficiently deal with uncertainty and unforeseen contexts, but the main issues associated with this approach are the time constraints and the computing overhead, as well as the risk of divergence;
- For combining the advantages of two above approaches while reducing the constraints, the **hybrid planning** is proposed by predetermining an adaptation plan at the design time and evolving this plan at run-time with an online learning process [Pandey et al., 2016].

2.2.4 Decision-Making Technique

In this section, we will discuss some general categories of decision-making approaches in the state of the art self-adaptive systems (Figure 2.10) and explore the qualitative properties of these approaches:

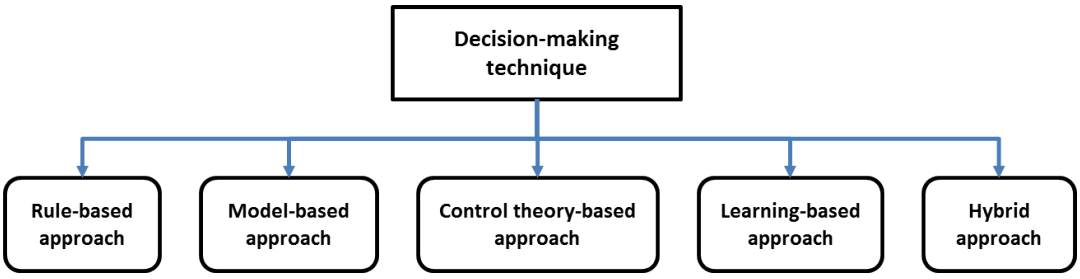


FIGURE 2.10: Decision-making techniques in self-adaptive systems.

- **Rule-based approach**: This technique seems traditional and popular in the field of self-adaptive systems with a set of rules defining the relationship between events and actions. It is represented by the event-condition-action (ECA) paradigm and by the IF - THEN clauses [Georgas and Taylor, 2009]. The set of rules is predetermined at the design time. By monitoring the current event, the adaptation decision can be chosen from the set of rules. An example of rules based on thresholds can be found in [Maggio et al., 2012]. This technique has the advantage of being easily understood and implemented. However, many dynamic factors at run-time of mobile robotic systems will lead to events or conditions that are undefined in the adaptation rule set;

- **Model-based approach**: In this approach, a set of analytical models of the managed system, the environment, as well as the quality/performance model is built during design. At the time of execution, depending on the adaptation decision criteria such as rules, objectives or utility of the system, the system architecture is chosen to work. Model construction generally begins with a few simple assumptions and goes through many trial and error calibration processes. It cannot therefore take into account all the uncertainties and the accuracy of the model becomes critical. A discussion on a roadmap of these approaches for complex software systems can be found in [France and Rumpe, 2007; Morin et al., 2009];

- **Control theory-based approach**: The engineering approach based on control strategies for self-adaptive systems has been studied by the work of [Filieri et al., 2017]. An example of this approach can be found in [Shevtsov and Weyns, 2016]. This approach is advantageous thanks to a solid mathematical basis and the ability to formally guarantee the behaviour of the controlled systems. But the system identification and controller synthesis phase in this approach is not trivial, and the accuracy of the model will define the performance of the adaptation process. In addition, the stability of the controller must be carefully considered;

- **Learning-based approach**: This approach uses a decision-making mechanism that allows the system's behaviour to be learned and adapted online on the basis of empirical data or past experiences with the system. The work of [Singh et al., 2017] has also explored this approach to managing the runtime execution of computing systems. Despite the time and high overhead costs of calculation, this approach is considered to be the most autonomous and intelligent decision-making mechanism. The recent works of [Sadighi et al., 2018] mentioned also this methodological innovation in their context of self-aware autonomous systems;

- **Hybrid approach**: In order to leverage the advantages of some decision-making approaches while reducing their disadvantages, some studies combine these approaches in the decision-making framework such as the work in [Zhao et al., 2017] combining rule-based and learning-based approach, and [Mishra et al., 2018] combining control theory and learning-based approach for their self-adaptive systems.

2.3 Case-Based Reasoning

Case-Based Reasoning (CBR) is a **problem-solving paradigm** that uses cases solved in the past to solve a new case whose solution is unknown. It is also an approach that supports the **decision-making** process in symbolic AI [Aamodt and Plaza, 1994]. A solved *case* includes a *context* describing the problem and a *solution* to successfully solve it. A new case means that the context is known but its solution is unknown. CBR uses a four-step process to propose a new case solution as denoted in Figure 2.11:

- **Retrieve**. When a new case arrives, this step seeks to find the most similar case in the case database based on certain measures of similarity;
- **Reuse**. The solution of the most similar case is then proposed to the new case. It can be reused directly or requires some transformations before being used;
- **Revise**. This step assesses the outcome of the proposed solution applied to the new case to confirm whether the new case has been successfully resolved or not;

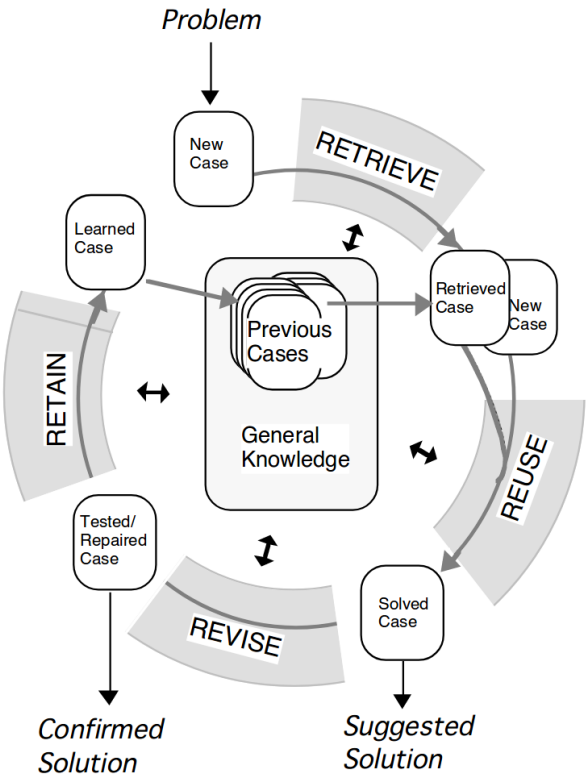


FIGURE 2.11: Case-Based Reasoning cycle uses a four-step process: Retrieve, Reuse, Revise and Retain (reprinted from [Aamodt and Plaza, 1994]).

- **Retain**. If the solution is confirmed, the new case and its confirmed solution will be updated in the case database.

Case-Based Reasoning is only a paradigm, not an implementation. Depending on the type of problem, the implementation of CBR may be different. The implementation of CBR faces some general challenges such as case representation, measurement of similarity, case base organization and solution refining. **Case representation** means how to describe a case, by feature vectors, or by texts, etc. **Measurement of similarity** calculates the similarity between cases that is required in the retrieve step. **Case database organization** is about how the case database is organized to effectively manage memory and facilitate case indexing. **Solution refining** is important to evolve the case database. All challenges require a domain-specific implementation. Recently, CBR has been widely applied in many domains such as dynamic goal management in self-adaptive systems [Zhao et al., 2017], quality of service prediction in cloud manufacturing [Liu and Chen, 2019], recommender systems [Pinto et al., 2019], etc.

2.4 Markov Decision Process

The Markov Decision Process (MDP) is a mathematical framework for modelling the problem of **sequential decision-making** under uncertainty [Pandey et al., 2016] and is generally solved by dynamic programming and reinforcement learning [Sutton and Barto, 2014]. A MDP is defined as a 5-tuple (S, A, T, R, γ), where:

- S is a finite set of states;
- A is a finite set of actions;
- T is a transition matrix representing the probability that at the given state s, taking the action a leads to the next state s', so $T(s, a, s') = P(s'|s, a)$;
- R is the immediate reward received after transitioning from state s to state s' when taking action a;
- γ is the discount rate that determine the importance weight of future rewards.

The solution of MDPs is to find an optimal policy π that maps between state space S and action space A, $\pi : S \rightarrow A$, in order to maximize the accumulative reward.

2.5 Reinforcement Learning

The reinforcement learning (RL) is one of the machine learning approaches, where the main idea is the **trial-and-error** process. An agent observes the environment state, makes some decisions and receives a reward or a reinforcement signal as an evaluation of the pair of current state and chosen action (Figure 2.12). The process is iterated until a good policy π for mapping between state and action is found. Generally, this approach is used to optimize the long-term goal as the accumulative reward, not immediate reward R_t, so the optimal policy maximizes the expected total discounted reward $\mathbb{E}\{\sum_{t=0}^{T} \gamma^t R_t\}$. The reinforcement learning is also known as a solution to resolve Markov Decision Processes, and the RL problem is usually modelled by MDPs. It can be said that the reinforcement learning is an automated process of learning a control policy between agent and environment with less prior knowledge.

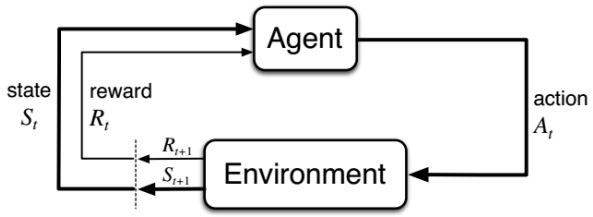


FIGURE 2.12: Agent-environment sequential interaction in a reinforcement learning: (*current state, action, reward, next state*) (reprinted from [Sutton and Barto, 2014]).

2.5.1 Q-Learning

Q-Learning is one of the algorithms resolving the model-free RL problem, of which the core is to find a Q-Table (Figure 2.13) representing the state-action value $Q(s, a)$. The iterating process is defined by

$$\delta_t = R_{t+1} + \gamma \max_A Q(S_{t+1}, A) - Q(S_t, A_t) \tag{2.1}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta_t \tag{2.2}$$

Where,

- t is the *time step*. It is also called as *decision epoch* or *adaptation time step*. At the time step t, the agent observes the state S_t and makes the action A_t. Consequently, at the next time step $(t + 1)$, the agent will obtain the reward R_{t+1} and receive the state S_{t+1}. Thus, a sequence of interaction between agent and environment is defined by $(S_t, A_t, R_{t+1}, S_{t+1})$ or (*state, action, reward, next state*);
- δ_t is defined as the *temporal difference*;
- R_{t+1} is the *immediate reward* received at the instant $(t + 1)$;
- $0 \leq \gamma \leq 1$ is called the *discount rate* that defines the importance weight of future rewards;
- $0 \leq \alpha \leq 1$ is called the *learning rate* that defines the rate of changing Q-value.

After finding Q-Table by the learning process, a mapping policy can be directly derived from Q-Table. At a given state s_i, the action a_j that makes $Q(s_i, a_j)$ maximum will be chosen to execute:

$$a_j = \arg\max_{a_j \in A} Q(s_i, a_j) \tag{2.3}$$

Q-Table (MxN Q-values)		Actions			
		a_0	a_1	...	a_{N-1}
States	s_0	$q_{0,0}$	...	...	$q_{0,N-1}$
	s_1	...	...	...	...
	...	...	...	...	...
	s_{M-1}	$q_{M-1,0}$	...	...	$q_{M-1,N-1}$

FIGURE 2.13: Q-Table approach of M states $S = \{s_0, s_1, ..., s_{M-1}\}$ and N actions $A = \{a_0, a_1, ..., a_{N-1}\}$.

Q-Table approach is only practical with finite state space and action space. When the number of states or actions is large, or in case of continuous state and action space, the Q-value should be represented by a linear approximation function or a neural network as the case of Deep Q-Learning.

2.5.2 Deep Q-Learning

As described above, the Q-Table approach has finite state space limits and may not be practical when dealing with massive state spaces. The article [Mnih et al., 2015] proposes to combine Q-Learning with deep neural networks (DNN) to address this limitation and achieve control at the human level. Now, the state that the agent perceives is continuous and is the entry of neural networks. The output of neural networks is a Q-value for each action in a finite action space (Figure 2.14). Q-Network is represented by a weight matrix θ. Thus, the learning process tries to find an optimal weight matrix θ^* that maximizes the accumulative reward. The loss function $\mathbb{L}$ for the training process of a Q-Network is defined by

$$\mathbb{L}(\theta) = [(R_{t+1} + \gamma \max_A Q(S_{t+1}, A, \theta)) - Q(S_t, A_t, \theta)]^2 \tag{2.4}$$

Where $(R_{t+1} + \gamma \max_A Q(S_{t+1}, A, \theta))$ is considered as target. We can realize that unlike supervised learning where the target is static, this target is dynamic.

The use of neural networks as function approximators for Q-Learning was applied a long time ago, but the real success is only achieved in 2015 with Mnih's article with two important contributions:

- **Experience replay**: experiences are stored in a memory and used for the mini-batch training of neural networks. This is an approach to obtain data efficiency while training neural networks;
- **Target network**: a target Q-Network with the weight matrix θ^- is kept static for a certain time and gradually copied from the current Q-Network with the weight matrix θ. This makes the learning process more stable. Thus, the loss function is now defined by

$$\mathbb{L}(\theta) = [(R_{t+1} + \gamma \max_A Q(S_{t+1}, A, \theta^-)) - Q(S_t, A_t, \theta)]^2 \tag{2.5}$$

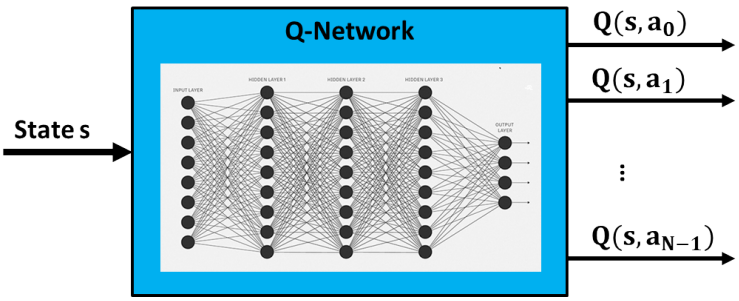


FIGURE 2.14: Deep Q-Network approach. In this approach, the state space is continuous, the action space is still discrete, and the Q-values are estimated by a Q-Network.

2.5.3 Decision Epoch and Episode

Most reinforcement learning problems can be broken down into sequences in which an agent interacts with its environment from an *initial state* until it reaches a certain *terminal state*. The initial state can be identified at the beginning of the interaction between agent and environment. The terminal states can be real or virtual. For example, in the case an agent plays a game, the terminal state can be defined when the agent wins or loses the game. Or more simply, the terminal state is reached when a waiting time has elapsed. Each sequence of agent-environment interactions between the initial and terminal states is therefore called an *episode*.

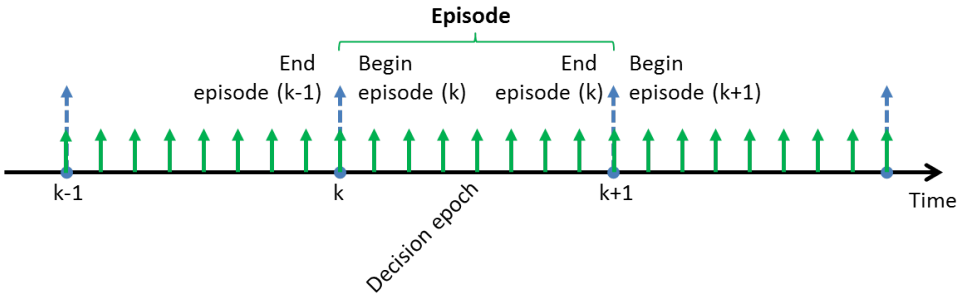


FIGURE 2.15: Illustration of episodes and decision epochs in the reinforcement learning. An episode is composed of many decision epochs.

In an episode, the decision is made at each *time step*, or at each *decision epoch*, or at each *adaptation time step*. We can use these terms interchangeably. Thus, an episode includes a number of decision epochs as depicted in Figure 2.15. The Q-Learning or Deep Q-Learning training process strives to maximize the accumulative reward $\mathbb{E}\{\sum_{t=0}^{T_{eps}} \gamma^t R_t\}$ in each episode, where T_{eps} is the number of decision epochs in this episode. And the training process is generally carried out through multiple episodes.

2.5.4 Exploration vs. Exploitation

The use of exploration and exploitation is an important factor that determines the success of reinforcement learning. While exploitation means that the agent always chooses the best decision based on current observations, exploration helps it to acquire new knowledge by sometimes using random decisions. This avoids local or short-term optimality and can lead to better decisions for the future. One of the techniques known to implement exploration and exploitation is the ϵ-greedy algorithm. This means that with a probability of $(1-\epsilon)$, the agent will choose the best decisions (exploitation), but with a low probability of ϵ, it will choose random decisions (exploration). For example, for the Q-Learning approach with the ϵ-greedy algorithm, at the time step t, the agent observes the state S_t, the action A_t is chosen by

$$A_t = \begin{cases} \text{a random action in the action space } A & \text{with probability } \epsilon \\ \arg\max_A Q(S_t, A) & \text{with probability } (1-\epsilon) \end{cases} \tag{2.6}$$

2.6 Previous Work on Performance and Energy-Aware Robotic Run-Time Adaptation

In this section, we apply the general concepts of self-adaptive systems for describing the previous approaches of self-adaptation in autonomous mobile robotic systems. While the functional requirements are normally mandatory for a robotic mission, we focus on self-adaptation for guaranteeing the non-functional requirements with a special emphasis on performance and energy goals. The aforementioned taxonomy-based analysis of previous approaches is synthesized in Table 2.1. The adaptation processes and the non-functional properties guarantee of each approach are detailed in Table 2.2. These previous work will be discussed in details and we will highlight the motivation for our research.

TABLE 2.1: Previous approaches for performance and energy aware run-time adaptation in autonomous mobile robotic systems.

Project	**Reason**	**Technique**	**Time**	**Adaptation Control**			
				Approach	**Criteria**	**Plan**	**DoD**
[Hernandez-Sosa et al., 2005]	res, ctx	prm (app)	re	ext	rules	off	hyb
[Zhang, Lu, and Hu, 2009]	ctx	prm (app,res)	re	ext	rules	off	cen
[Inglés-Romero et al., 2013]	res, ctx, u	prm (app,res)	re	ext	rules, models	off	cen
[Gherardi and Hochgeschwender, 2015]	res, ctx	str	re	ext	rules, models	off	cen
[Ondrúška et al., 2015]	res	str	pro	int	rules	off	cen
[Jaiem et al., 2016b]	res, ctx	prm (res), str	pro	int	goals, models	off	cen
[Brugali et al., 2018]	ctx	str	re	ext	rules, models	off	cen
[Pandey et al., 2018]	ctx	prm(app)	re	ext	non-functional goals	off	cen
our research objectives	res, ctx, u	prm (app, res)	pro	ext	non-functional goals	hyb	hyb

ACRONYMS: **DoD** - degree of decentralization, **res** - resources, **ctx** - context, **u** - users, **prm** - parameters, **str** - structure, **app** - applications, **re** - reactive, **pro** - proactive, **ext** - external, **int** - internal, **off** - offline, **hyb** - hybrid, **cen** - centralized.

TABLE 2.2: Previous approaches: adaptation processes and non-functional requirements guarantee.

Project	**Adaptation Processes - MAPE Functionality**			**NFRs Guarantee?**	
	Monitor	**Analyze & Plan**	**Execute**	**Performance**	**Energy**
[Hernandez-Sosa et al., 2005]	component level (period, elapsed time, CPU time), system level (computational load, battery level, load profile)	rule-based adaptation: timeout control at component level, CPU load control at system level	degrade perf level, promote perf level, operate at specific perf level by reconfiguring two knobs: frequency of operation and quality level	Yes	No
[Zhang, Lu, and Hu, 2009]	computing workload (CPU cycles) of recognition task	rule-based adaptation: optimal solution	reconfigure max robot velocity and processor frequency	No	Yes
[Inglés-Romero et al., 2013]	context-dependent metrics (battery level, distance to coffee machine, waiting time, ambient noise)	rule- and model-based adaptation: threshold-based rules, empirical equation for constraint solver	reconfigure max robot velocity and other scenario-specific parameters	Yes	Yes
[Gherardi and Hochgeschwender, 2015]	context-dependent measurements	rule- and model-based adaptation: event-based rules, feature model	reconfigure system structure	Yes	No
[Ondrúška et al., 2015]	safe distance deviation	rule-based adaptation: greedy, belief-based rules	switch on or off localization subsystem	No	Yes
[Jaiem et al., 2016b]	time, energy, safety related metrics	model-based adaptation	reconfigure max robot velocity and structure (sensors, algorithms)	Yes	Yes
[Brugali et al., 2018]	context (not yet implemented)	rule- and model-based adaptation: event-based rules, feature model, quality of service model	reconfigure structure (software, sensors, robot platform)	Yes	No
[Pandey et al., 2018]	scores of object detection algorithms	Markov Decision Process	reconfigure object detection algorithm parameters	Yes	No

2.6.1 Self-Adaptation in CoolBOT, Hernandez-Sosa et al., 2005

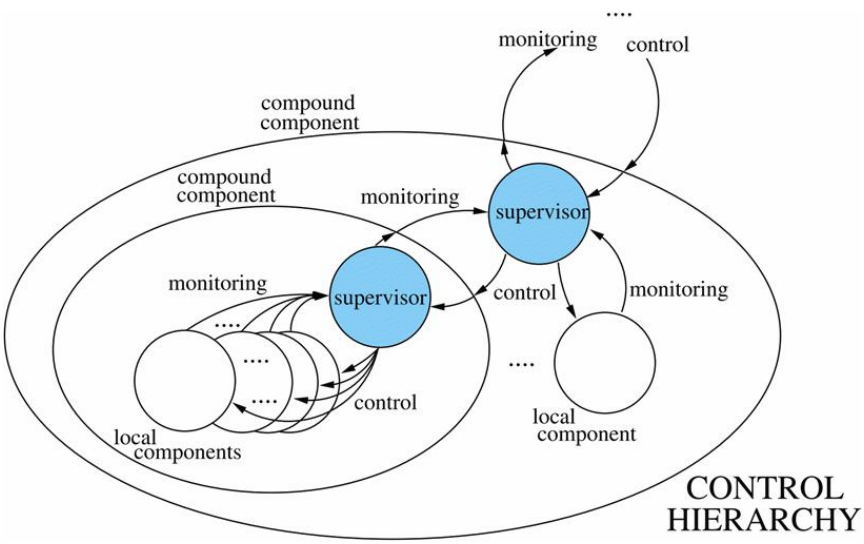


FIGURE 2.16: Component-based run-time self-adaptation: hybrid control hierarchy at local and compound robotic components (reprinted from [Hernandez-Sosa et al., 2005]).

Hernandez-Sosa et al. [Hernandez-Sosa et al., 2005] propose the self-adaptation in a component-based robotic framework named CoolBOT. The dynamic adaptation is conducted at the component level for trading off between the computational resource consumption and the quality of results. This trade-off is then used to determine the system performance level. Each component is considered as execution units and executed as threads in the underlying operating system. Based on the component level observation such as period, elapsed time or CPU time, and on the system level measures such as computational load, battery level or load profile, the adaptation decisions can be to degrade a component to the lower level of performance, or to promote a component to the upper level of performance, or to bring a component to a specific level of performance. Two rule-based adaptation decisions have been utilized: *timeout control* at the component level and *CPU load control* at the system level. The control hierarchy is depicted in Figure 2.16. The methodology has been demonstrated in two different robotic missions: a tracking system and a mobile searching system. This research considers only computational context and computing resources, CPU in particular, but does not consider energy resources. The NFR defined as computing system balance is identified by the system.

2.6.2 Optimal Solution for Power Management, Zhang et al., 2009

Zhang, Lu and Hu [Zhang, Lu, and Hu, 2009] introduce the optimal solutions to a class of power management problems in mobile robots. The work deals with a navigation mission in which the navigating guideline is guaranteed by a recognition task that must be completed before the robot reaches next turning point in order to guarantee the safety (Figure 2.17). An optimal control problem of both robot velocity and processor frequency is proposed for accomplishing the mission with the least energy consumption. The optimal solution is resolved at the design time. At run-time, the current circumstance is defined by the recognition task computing workload in the form of CPU cycles, and the robot velocity and the processor frequency are calculated from the predetermined optimal solution adapting to this circumstance. This

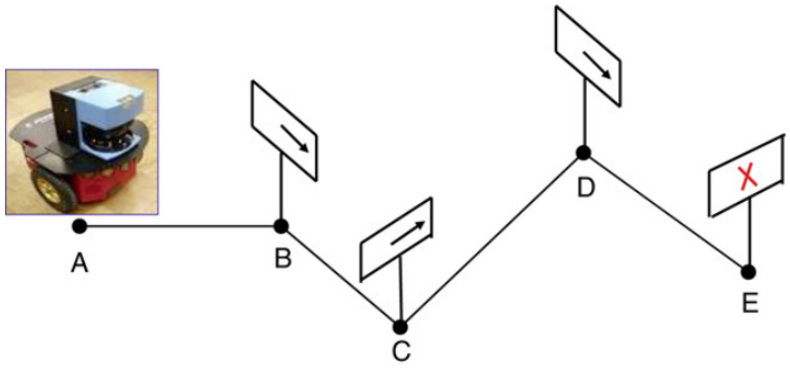


FIGURE 2.17: Motivational robotic mission example (reprinted from [Zhang, Lu, and Hu, 2009]). The robot must finish correctly some recognition tasks while navigating to determine the navigation directions at the waypoints B, C, D and E.

approach tackles both computing level and sensing and acting level with two power consumption models: computing power is a function of processor frequency, and sensing and acting power is a function of robot velocity. The analytical solutions are reserved for a class of power problems that ask the question of generalization. Moreover, the predetermined solutions at the design time cannot deal with the uncertainties and the unforeseen contexts at run-time.

2.6.3 Variability Modelling Language, Inglés-Romero et al., 2013

Inglés-Romero et al. [Inglés-Romero et al., 2013] present a Variability Modeling Language (VML) as depicted in Figure 2.18 to express run-time variability in service robotics with regard to the execution quality of the robot, or non-functional properties. The variability is modeled by variation points (or configuration knobs), current contexts, QoS properties and predetermined adaptation rules mapping between contexts and variation points. The adaptation logic is implemented in an external and centralized manner. The methodology is then applied for a coffee delivery mission in which the adaptation decisions are max robot velocity, type of coffee machine and other parameters while respecting the battery level, the safety and the user-defined QoS such as coffee temperature. In this approach, they do not consider the computing resources and do not model the power consumption of the system. The adaptation rules are determined at the design time and the constraint solver at run-time is also based on predetermined empirical equations. Thus, when dealing with the dynamics of the mobile robotic systems, this approach has its limitations.

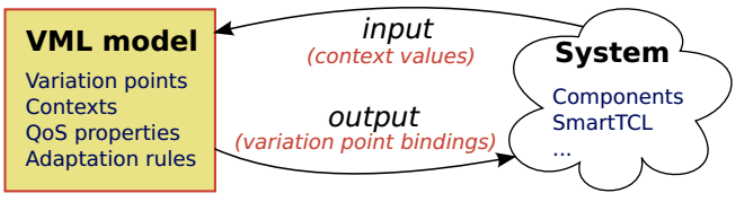


FIGURE 2.18: Modeling run-time variability: VML models and the interaction with the robotic system (reprinted from [Inglés-Romero et al., 2013]).

2.6.4 Model-Based Self-Adaptation, Gherardi and Hochgeschwender, 2015

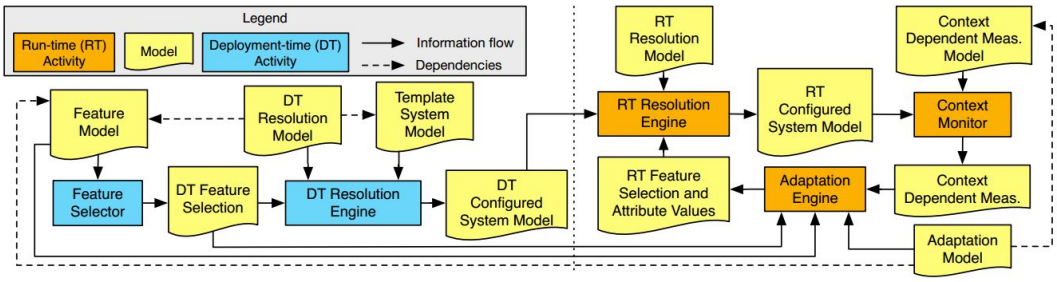


FIGURE 2.19: An overview of the model-based variability management approach (reprinted from [Gherardi and Hochgeschwender, 2015]). A set of models representing the system architecture, its variability, and the state of the context is defined at the design time and is used at run-time to implement the run-time adaptation.

Gherardi and Hochgeschwender [Gherardi and Hochgeschwender, 2015] develop a model-based approach for run-time adaptation of robotic systems as depicted in Figure 2.19. In this approach, a set of models representing the system architecture, its variability, and the state of the context is defined. Based on the current context, the framework will reason and adapt appropriate models for the system, so that the best QoS of the system can be obtained. The notion of QoS is still vague in this study. The run-time variability is resolved by a set of predetermined adaptation rules. This approach also does not consider the computing resources , nor is the power consumption model mentioned. It concentrates on the adaptation for completing the functional requirements, but the non-functional goals are not explicitly implemented.

2.6.5 Energy-Aware Perception Scheduling, Ondrúška et al., 2015

The work in [Ondrúška et al., 2015] explores the idea of reducing the energy consumption of the robotic system by scheduling the specific component, localization in this study, in the navigation mission. The scheduling decision is based on safety criteria known as "safe corridor" in order to switch the localization component on

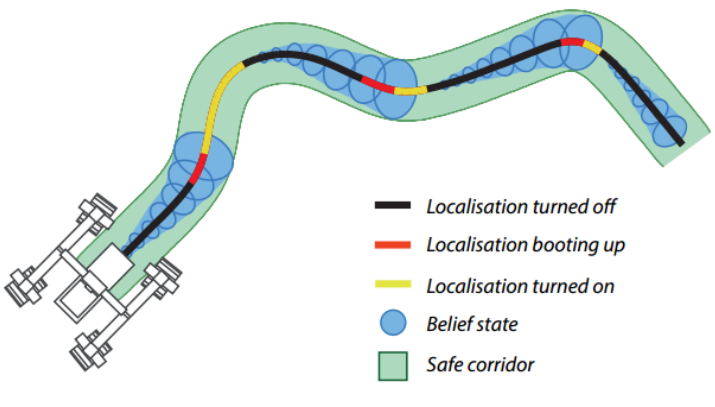
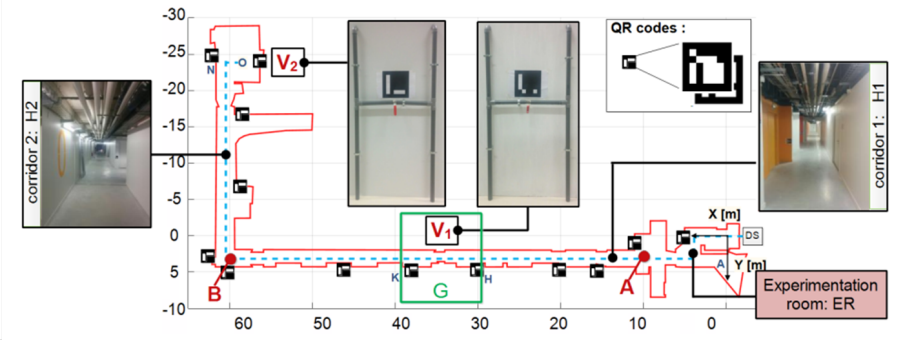


FIGURE 2.20: Scheduling perception for energy efficiency: scheduling decisions based on belief state in the safe corridor (reprinted from [Ondrúška et al., 2015]).

or off during some periods (Figure 2.20) for reducing the energy consumption of the localization component, and thus also of the robotic system. The problem of perception scheduling is framed as a belief Markov Decision Process and the dynamic programming is used to provide the optimal schedules. However, the application of this methodology to the other robotic components is not trivial and was not discussed. Thus, the generalization of the scheduling framework is an issue.

2.6.6 Performance-Aware Hardware and Software Adaptation, Jaiem et al., 2016



x_{i-1} - x_i (m)	0 - 34			37	37	37	37 - 93.5				93.5	93.5	93.5	93.5 - 187			
Objective O_i	DS → V1			↻	⋈	↻	V1 → V2				↻	⋈	↻	V2 → DS			
n_{alt_i}	21			2	1	2	21				2	1	2	21			
Task T_k	FM / L			R/L	VD	R/L	FM / L				R/L	VD	R/L	FM / L			
$A_k^{c_j}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
n_{alt_k}	21	9	3	2	1	2	9	21	21	7	2	1	2	21	21	9	21
$V_{max}(m/s)$																	

FIGURE 2.21: Mission decomposition (reprinted from [Jaiem et al., 2016b]). The mission is divided into several objectives. For each objective, there are a number of alternative implementations (n_{alt}) that can be chosen for deployment.

Jaiem et al. [Jaiem et al., 2016b] propose an approach to reconfigure the system architecture composed of algorithm implementations and sensors at run-time in order to guarantee the performance and energy goals while respecting constraints. The framework begins with an offline performance estimation and then it is used for online performance evaluation and for adaptation decisions. This approach proposes both power consumption models and explicitly indicates the non-functional requirements for mobile robots. However, the adaptation at the computing level is not yet implemented. Moreover, the mission decomposition as denoted in Figure 2.21 with many phases limits the generalization for other robotic missions.

2.6.7 Model-Based Reconfigurable Robotic Systems, Brugali et al., 2018

The recent work in [Brugali et al., 2018] also addresses model-based engineering with a set of models of the system architecture and a QoS model built offline in order to reconfigure the robotic system architecture at run-time. They did not consider the computing and energy resources, but only considered some navigation system-specific non-functional properties. The model-based adaptation is based on event triggers with a set of pre-determined scenarios (context model). The architecture of the reconfiguration system is shown in Figure 2.22 with three main components such as monitoring engine, adaptation manager and reconfiguration engine.

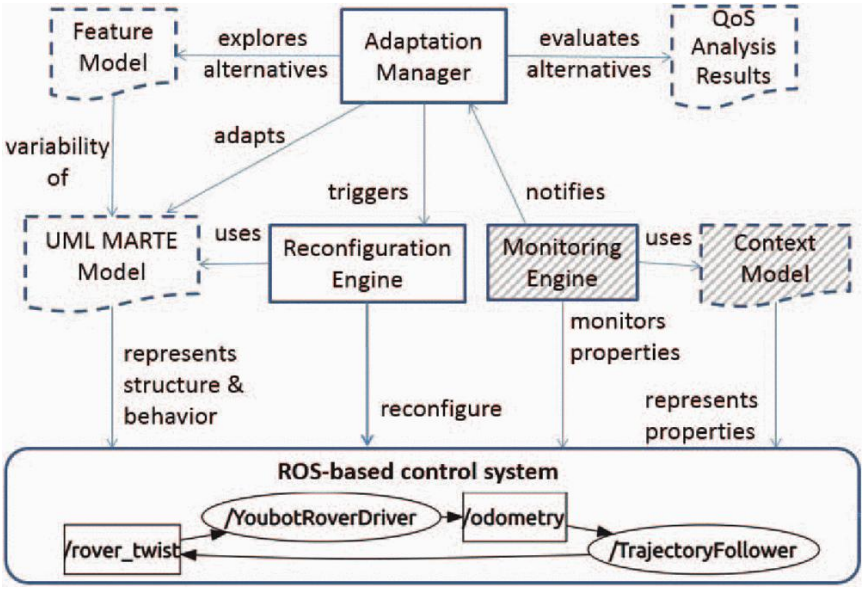
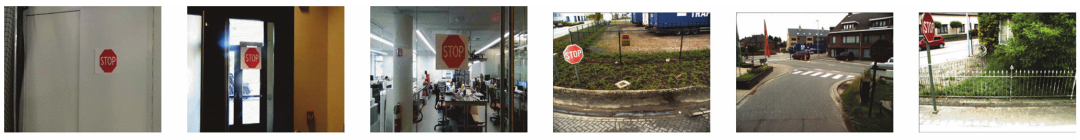


FIGURE 2.22: Model-based reconfigurable robotic systems: the architecture of the reconfiguration system (reprinted from [Brugali et al., 2018]).

2.6.8 Parameter Adaptation for Robotic Vision Algorithms, Pandey et al., 2018

This research [Pandey et al., 2018] addresses the problem of adapting the parameters of computer vision-based robotic missions to achieve real-time performance while respecting the accuracy in resource-constrained mobile robots. The motivation is illustrated in Figure 2.23 where the dynamics of the environment is analyzed and the relationship between input data, algorithmic parameters and execution time is also discussed. They then formulated the decision-making problem as a Markov Decision Process to adapt the algorithmic parameters depending on the input data in order to tradeoff detection accuracy and execution time. This approach is innovative and indicates that the dynamic adaptation of parameters according to incoming data is more effective than fixing them a priori in terms of resource consumption. However, the generalization of the methodology for different types of robotic missions has not yet been discussed.



(a) (b) (c) (d) (e) (f)

FIGURE 2.23: Motivation figures to illustrate that parameters of the object detection algorithms can be adapted to input data to achieve savings in time and energy. Figures (a, b, c) show variation in the amount of background clutter; (d and e) show variation in illumination, and (f) shows variation in camera viewpoint. These variations lead to differences in the minimum number of bounding boxes parameter (detection proposals) required for object detection. Lower detection proposals translate to lower execution time (reprinted from [Pandey et al., 2018]).

2.6.9 Remarks on Previous Work

The previous efforts that we mention above again highlight the important role of run-time adaptation or self-adaptation for enhancing the autonomy of mobile robotic systems because of shortage of onboard robot resources such as processing power and energy, and of the uncertain, dynamic circumstances that can occur at the robotic run-time. Although each approach has its advantages and provides the success in some case studies, most of the previous works does not meet our expectations for a robotic run-time adaptation framework that:

- **Explicitly takes into account user- and/or system-defined performance and energy goals.** This point means that these non-functional goals are defined as setpoints or thresholds for reconfiguring or not the structure and behaviour of the robotic system. Most of the previous work tend to reconfigure the robotic system and reassess these resulting non-functional properties;
- **Deals with the mobile robotic dynamics and uncertainties with run-time intelligent decision-making mechanisms.** The decision-making process is considered an important factor in deciding on the effectiveness of the adaptation framework. Decision-making algorithms based on rules and models are dominant in previous work that cannot cope with many unexpected dynamics and uncertainties in the context of mobile robotics;
- **Ensures the generalization of the management methodology.** This means that the methodology can be easily applied to the other robotic missions. Indeed, we see the difficulties of reusing the methodologies proposed by previous work. In addition, the context in which several robotic missions are deployed simultaneously in the same robotic system has not yet been considered.

Indeed, the three remarks above underline our motivation for a run-time adaptation framework in the context of mobile robotics and guide our research. The last line of Table 2.1 also reveals our expectation by highlighting the characteristics necessary for the targeted robotic run-time adaptation framework that is not yet fully achieved by the previous approaches.

2.7 Summary

In this chapter, we have provided the background for our research. The main concepts of two domains of mobile robotic systems and self-adaptive systems have been mentioned to understand the concept of the robotic run-time adaptation. The basis of decision-making techniques, reinforcement learning and case-based reasoning was provided to clarify the methodology that we will present in the next chapters. In addition, some related work was introduced and analyzed to position our research in the state of the art performance and energy-aware robotic run-time adaptation.

The characterization of the robotic mission is considered as the preliminary phase to the design of our run-time adaptation framework. In the next chapter, we will present unified concepts to characterize a robotic mission and then provide some case studies to prove these concepts and highlight the motivation for run-time adaptation.

Chapter 3

Mobile Robotic Mission Characterization

Contents

As we presented in the previous chapter, we propose the level of abstraction of a robotic system where the robotic mission is at the highest level and it hides the complex details of the robotic system. It uses the robot's necessary components and resources to meet its functional needs. The characterization of robotic missions is an important task in our thesis. This will provide the basic knowledge of the mission under consideration and will be the preliminary step in the framework for run-time adaptation of the mission. The characterization is considered under two aspects: **offline characterization** and **online monitoring**. While the offline characterization provides in-depth knowledge of the mission and helps to formulate the management

problem, the online monitoring provides real-time observation for the adaptation part of our framework.

In Section 3.1, we will present some related work of characterizing the robotic performance and energy consumption. Then, our concepts for the mobile robotic mission characterization and a systematic approach for characterization are proposed in Sections 3.2 and 3.3. Section 3.4 presents the concepts of self-aware mobile robotic mission. It is important to define the framework in simulation and in real robotic platform for implementation and validation of our proposed methodology as presented in Section 3.5. Some motivational examples of mobile robotic missions are also described in Section 3.6 and followed by some insightful discussions. Finally, Section 3.7 concludes this chapter.

3.1 Related Work

3.1.1 Energy Consumption Characterization

Energy is the survival problem of mobile robotic systems because of their limited energy capacity. Thus, in the literature, there is a great deal of effort to model and estimate the energy consumption of robotic systems in order to facilitate efficient energy management. The researches such as [Mei et al., 2006; Parasuraman et al., 2014; Jaiem et al., 2016a; Rappaport, 2016] break down the robotic systems into many components such as motion, sensing and embedded computers and estimate the energy consumption of these components. The consumption of the motion part is generally modelled according to the speed of the robot. The one of the sensing part is modelled by the sampling frequency of the sensors. The energy consumption of the embedded computer is reserved for high-level processing applications and many studies consider it to be constant or fairly constant depending on the application. The energy consumption model of the applications can be found in the work of [Noureddine, Rouvoy, and Seinturier, 2015] and it should be variable. In this work, the power consumption of an application or a software ($P_{software}$) is defined as:

$$P_{software} = P_{comp} + P_{comm} \tag{3.1}$$

Where P_{comp} is equal to the CPU power consumed by software, and P_{comm} is equal to the consumed power for transmitting software's data. Other efforts such as the work of [Dressler and Fuchs, 2005; Berenz, Tanaka, and Suzuki, 2011; Zhang et al., 2014; Hamza and Ayanian, 2017] directly estimate the remaining capacity of robotic batteries to effectively plan the mission.

Based on these principles, our characterization aims to propose unified concepts to understand the energy consumption characteristic of a robotic mission.

3.1.2 Performance Characterization

Although performance characterization is popular in many areas such as computer architecture, computer vision, machine learning, etc. [Clemons et al., 2011; Thomas et al., 2014], this type of characterization for mobile robotics systems is still vague, or only the characterization of individual components but not the overall performance of the robotics system, due to the lack of reference data and many dynamic factors in the operational context that limits repeatability and reproducibility. The work of [Holz, Iocchi, and Zant, 2013; Sprunk et al., 2014; Twigg, Gregory, and Fink, 2016]

aim to propose a methodology to characterize the performance of the entire navigation system, and the work of [Weisz, 2016] tends to propose a benchmark facilitating the sustainable robotic characterization. The characterization of offline performance of robotic systems is also mentioned in the work of [Cano et al., 2016] where the experts' experiences were used to estimate the performance of each component (ROS node) and the overall performance is the weighted sum of these components. In the work of [Jaiem et al., 2016b], they explicitly represent the concept of performance in robotic systems such as safety, localization, stability and duration. The performance model and the performance margin are determined at the design time and are used at the execution time to estimate the performance level. We then leverage these concepts to propose the robotic mission-specific performance or the mission's quality of service.

3.2 Mobile Robotic Mission Characterization

3.2.1 Mission Definition

Each robotic mission is deployed in order to satisfy specific objectives and requirements. The objectives can be functional or non-functional, and typically are defined by either the end user or the robot operator, who we will refer to as the robot users (see Figure 1.6). The functional objectives specify what the robotic system should do in the form of functional tasks and they are usually sequenced by the mission's task graph. For example, an autonomous navigation mission can be deployed for visiting a set of user-defined waypoints known as functional goals in the robot working environment. Each functional task can in this case be considered as a navigating task between two waypoints and a simple task sequencing algorithm can be depicted in Figure 3.1.

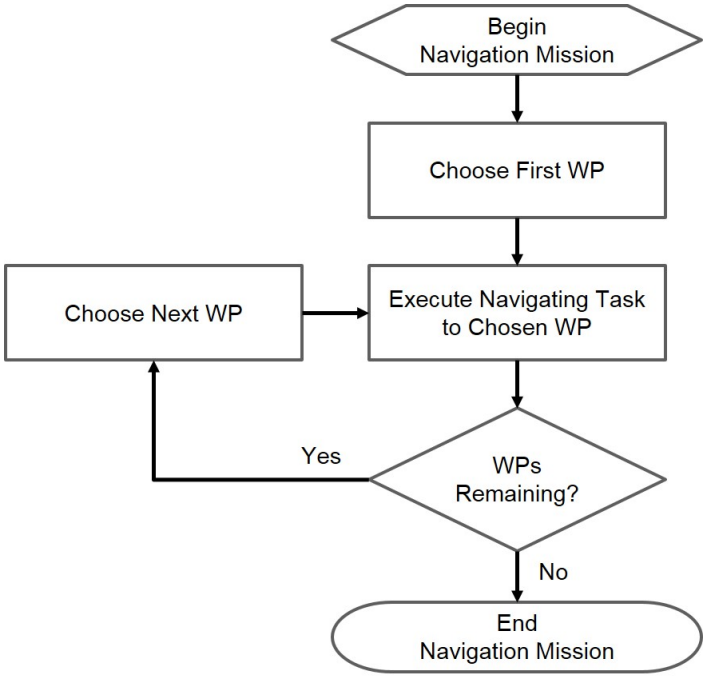


FIGURE 3.1: Example of an autonomous navigation mission: task sequencing algorithm (adapted from [Brutzman et al., 2018] for a navigation mission). *Note: WP - waypoint.*

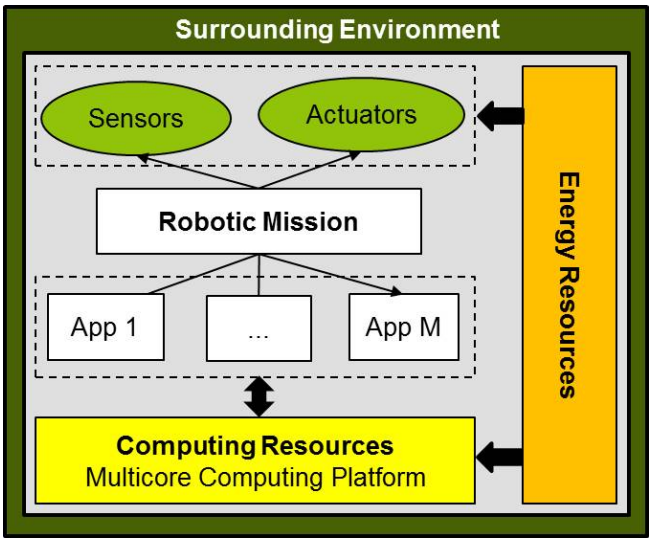


FIGURE 3.2: Mobile robotic mission model with components of sensors, actuators and computing applications, and connection with computing platform, power supply and surrounding environment.

The non-functional objectives specify how the system performs the mission, evaluation of performance, quality of service, etc. Each mission must interface with some necessary robot sensors and actuators and call some processing applications on the computing subsystem for executing functional tasks. All the robotic systems usually interact with their surrounding environment for accomplishing the mission as described in Figure 3.2.

At the highest level of abstraction in the autonomous mobile robotic system, a robotic mission *Mission* is formally defined as a 4-tuple:

$$Mission = \{Objs, Apps, Sens, Acts\} \tag{3.2}$$

Where,

- $Objs$ represents the mission's functional and non-functional objectives;
- $Apps = \{app_i\}_{i=1}^{N_{Apps}}$ is the set of processing applications that carry out the mission and the connection between these applications formulates the mission's computing graph;
- $Sens = \{sen_i\}_{i=1}^{N_{Sens}}$ is the set of sensors;
- $Acts = \{act_i\}_{i=1}^{N_{Acts}}$ is the set of actuators.

The definition we propose helps the robotic mission characterization to be generalized and applied for many different robotic systems. It indicates not only mission performance and constraints, but also the way that the mission uses the robotic resources and energy. Some previous works such as [Weisz, 2016; Jaiem et al., 2016b] also proposed the concepts for the robotic missions but mainly concentrates on the mission applicative tasks, and did not really indicate the mission characteristics in term of performance, constraints and resource utilization. Based on the *Level* concept in the taxonomy of self-adaptation (Figure 2.8), the adaptation at the mission level will involve the application level with an ensemble of applications and the technical resource level of sensors and actuators.

3.2.2 Mission from a Computing Platform Point of View

On the computing platform including the underlying operating system, we can consider a robotic mission as a multi-process computing workload. Each process will be responsible for some robotic functionalities. Many threads can be launched in a process, therefore each process is also a multi-thread computing workload (Figure 3.3). The mission computing workload is also dynamic because of the changing functional behavior of the mission, resource availability, as well as surrounding circumstances [Brugali et al., 2018], as characterized by the work in [Zhang, Lu, and Hu, 2009; Ho et al., 2018]. The robotic systems must also cope with this computing variation by a run-time adaptation identified at the computing platform level. In fact, the adaptation at this level has attracted one of the known research branches of SAS called autonomic computing systems [Kephart and Chess, 2003]. However, in order to apply the autonomic computing approaches to the mobile robotic systems, we need adequate modifications for coping with the robotic dynamics, as well as to make it coherent with different adaptation levels.

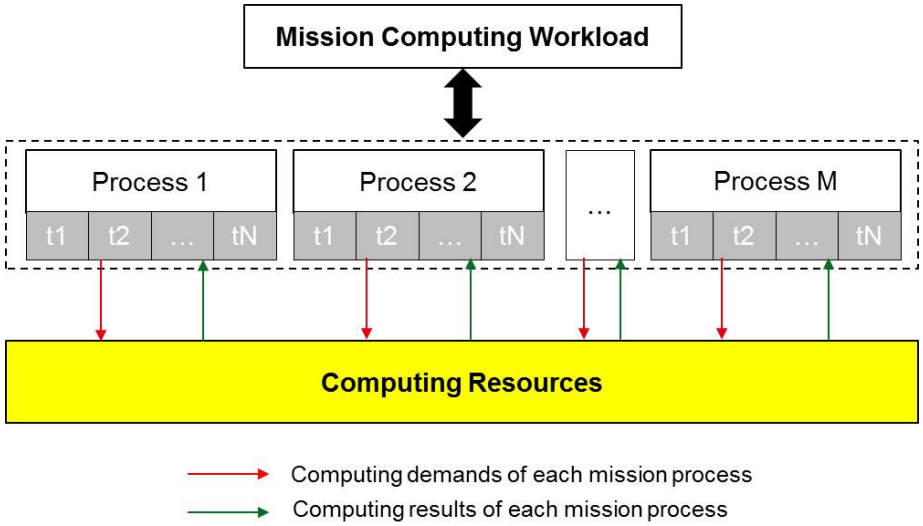


FIGURE 3.3: Mission computing workload. On the computing platform, a robotic mission is a multi-process workload. Each mission process is also a multi-thread workload. (Note: $t1$, ..., tN is thread)

3.2.3 Mission Power Consumption

Based on our mission definition and depending on its required components, two power consumption models of a robotic mission can be built: sensing and acting power consumption model, and computing power consumption model:

$$P_{Mission} = P_{sensing\ \&\ acting} + P_{computing} \tag{3.3}$$

$$P_{sensing\ \&\ acting} = P_{Sens} + P_{Acts} + P_{Controller} \tag{3.4}$$

$$P_{computing} = P_{Apps} + P_{Comm\_Sens\_Acts} \tag{3.5}$$

Where,

- $P_{Mission}$ is the power consumption of the mission;

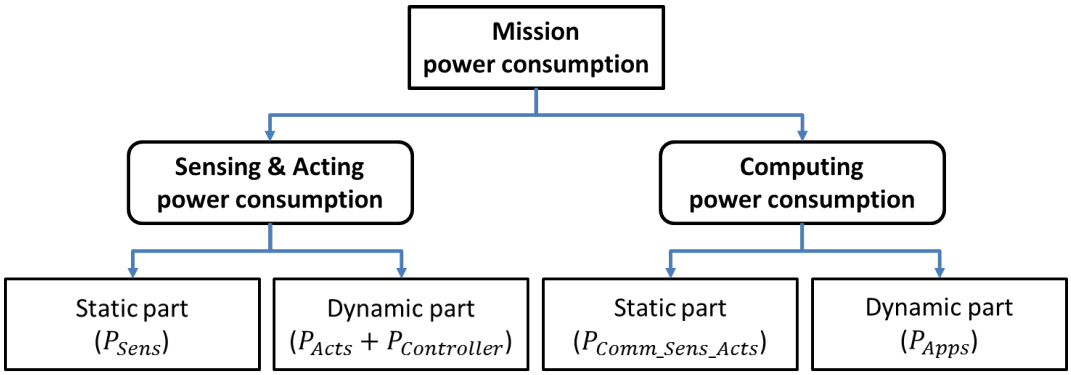


FIGURE 3.4: Mission power consumption model.

- $P_{sensing\ \&\ acting}$ is the power consumption of sensors P_{Sens}, of actuators P_{Acts} and occasionally of low-level controller $P_{Controller}$;
- $P_{computing}$ is the power consumption of computing processor reserved for applications P_{Apps} and $P_{Comm\_Sens\_Acts}$ for communication with sensors and actuators part.

In fact, for each power consumption model $P_{sensing\ \&\ acting}$ and $P_{computing}$, we can consider two parts such as **static** and **dynamic** parts. The static part means that its energy consumption is fairly constant during mission operation or for reasons of simplicity, we estimate a constant value for this part. And the dynamic part means that its energy consumption is variable and depends on the operating conditions. In the $P_{sensing\ \&\ acting}$ model, if we assume that the sensors operate at a constant frequency, their energy consumption P_{Sens} can be considered static. Otherwise, actuators and their controllers, such as wheel motors, depend on speed and environmental conditions, so their power consumption ($P_{Acts} + P_{Controller}$) should be dynamic. Similarly, in the model of $P_{computing}$, we can consider that the static part is $P_{Comm\_Sens\_Acts}$ and the dynamic one is P_{Apps}. The power consumption model of the mission is depicted in Figure 3.4. A complete power consumption model is useful if we consider allocating accurately the power budget for the robotic mission. For our run-time adaptation framework, our primary concern is the dynamic part of the power consumption and the adaptation mechanism strives to guarantee this dynamic part. The integration or not of the static part into the power consumption model of the robotic mission is an option.

The energy consumption of the mission $E_{Mission}$ is finally calculated by integrating the power consumption $P_{Mission}(t)$ over the mission duration dt as follows:

$$E_{Mission} = \int_{t=0}^{mission\_duration} P_{Mission}(t)dt \tag{3.6}$$

3.2.4 Mission Non-Functional Requirements

Based on the quality in use model defined by the standard ISO 25010 as presented in Section 1.1, in the mobile robotic context, we classify the non-functional requirements (NFRs) of a robotic mission into three subcategories (Figure 3.5):

- **Mission-specific performance** (M-Perf) is strongly related to the mission functionalities, and is usually user-defined. For example, the navigation time or the robot's mean speed can be considered as one of the autonomous navigation mission-specific performances [Jaiem et al., 2016b; Ceballos, Valencia, and

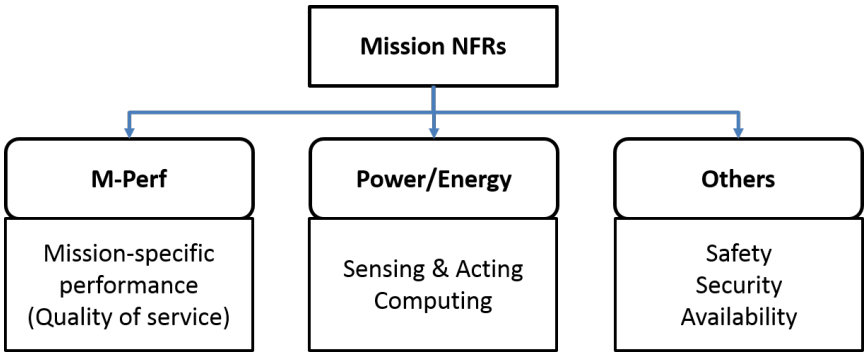


FIGURE 3.5: Three subcategories of the robotic mission non-functional requirements: mission-specific performance, power consumption, and others such as safety, security, etc.

J. Londoño Ospina, 2010; Cano et al., 2016]. The way in which the robotic system responds to these performance requirements will determine the **quality of service** (QoS) of the robotic mission;

- **Power/Energy consumption** (Energy) is concerned because of the limitation of energy resources on the mobile robots. As mentioned above, the energy consumption of the robotic mission is composed of sensing and acting, and computing consumption parts [Jaiem et al., 2016b; Weisz, 2016; Zhang, Lu, and Hu, 2009];

- **Other system requirements** (Others) such as safety, security, reliability or availability of robotic system are also non-functional requirements that can be considered while deploying the mission [Brugali et al., 2018; Jaiem et al., 2016b; Hernandez-Sosa et al., 2005; Zhang, Lu, and Hu, 2009; Ondrúška et al., 2015; Gherardi and Hochgeschwender, 2015]. These requirements are considered optional in our work and depend on the type of mission.

3.2.5 Robotic Configuration Knobs

A configuration knob is defined as the configuration that can be tuned to cope with the non-functional requirements. There are many configuration knobs in the different levels of mobile robotic systems. We divide them into two main levels: **mission-level** and **computing platform-level**. Figure 3.6 summarizes the possible configuration knobs of a mobile robotic system.

- At the **mission-level**, the mission is composed of a set of applications, sensors and actuators, therefore we can configure the compositional structure of these components [Jaiem et al., 2016b; Brugali et al., 2018; Gherardi and Hochgeschwender, 2015; Ondrúška et al., 2015] for not only accomplishing the functional requirements of the mission, but also contributing to the non-functional requirement guarantees. Furthermore, each component possesses a rich set of internal configurable parameters that can be tuned by the configuration knob [Cano et al., 2016; Hernandez-Sosa et al., 2005; Zhang, Lu, and Hu, 2009; Inglés-Romero et al., 2013]. Determining the key parameters is non-trivial, but it needs a rigorous characterization phase with a good understanding of the mission domain. In fact, changing the composition of robotic

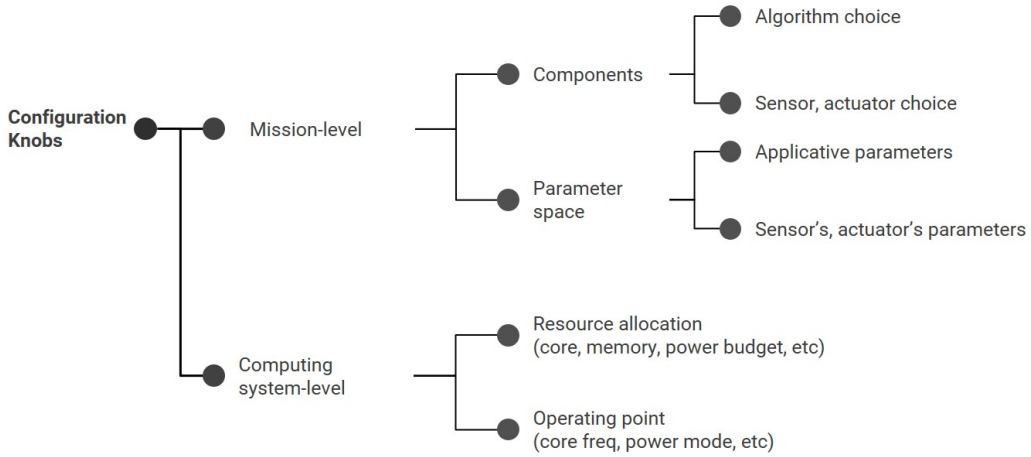


FIGURE 3.6: Robotic configuration knobs define the configurations that can be tuned to cope with the robotic non-functional requirements.

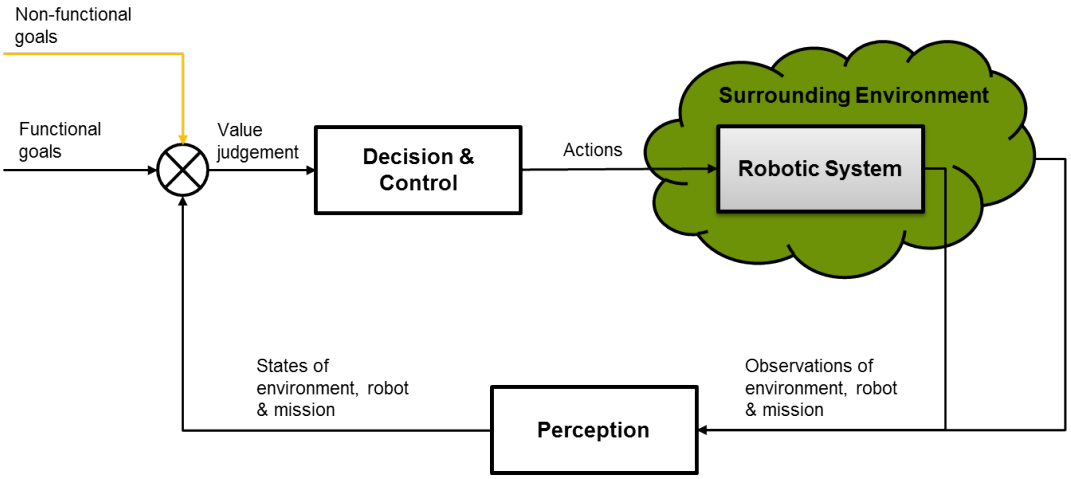
systems during execution may lead to a service interruption, while reconfiguring parameters in the parameter space of sensors, actuators and applications is easier and safer;

- At the **computing platform-level**, we can reallocate the number of computing resources such as core, memory or computing power budget [Cano et al., 2016]. Or we can reconfigure the operating point of the computing resources such as core voltage and frequency, power mode, etc [Zhang, Lu, and Hu, 2009]. For truly effective computing resource management, we need an approach to the granularity of application's tasks or threads. Thus, the adaptation at this level is not part of our thesis.

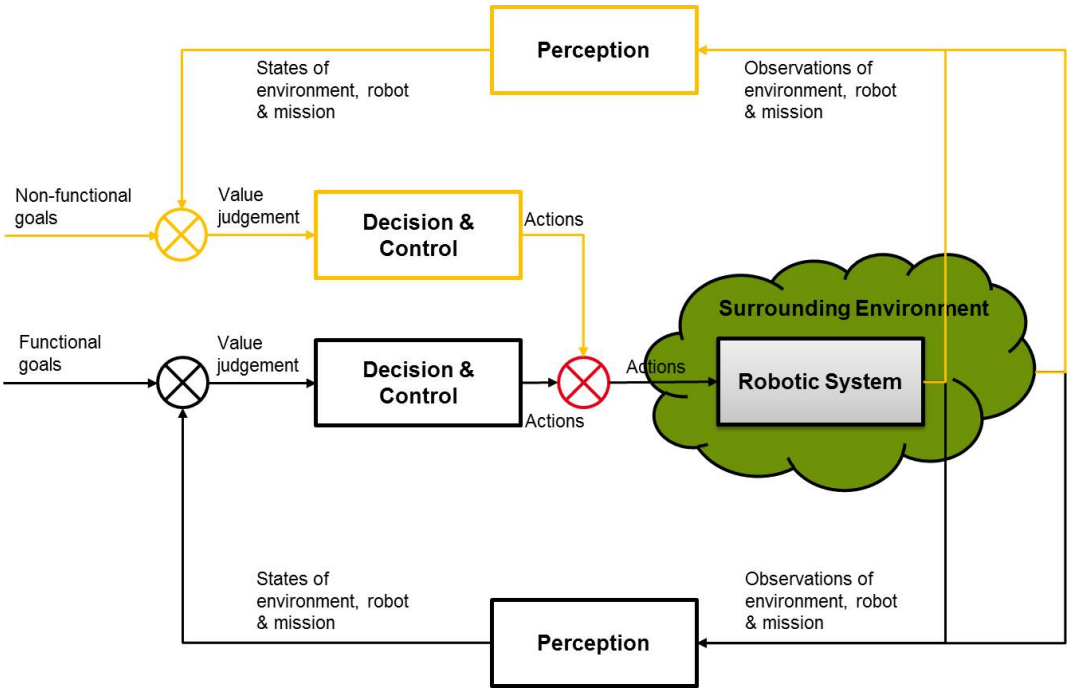
3.2.6 Autonomy Enhancement by Guaranteeing NFRs

In the paradigm denoted in Figure 1.6, the robot user deploys the robotic mission while identifying both functional and non-functional requirements. The functional requirements are dependent on the robot's functionalities and normally mandatory. The NFRs can be also identified by the robot's constraints. All these requirements will be applied to the robotic mission. The robot performs the mission by allocating all necessary components such as processing applications, sensors, actuators and energy to the mission. The full autonomy is obtained when the robotic mission can accomplish successfully all the functional requirements while guaranteeing the non-functional requirements. Figure 2.5 shows a generic control structure to meet the functional goals. With the addition of non-functional goals, two approaches are described in Figure 3.7:

- **The first approach in Figure 3.7a means that the same control and decision loop for robotic systems must take into account both functional and non-functional goals.** Thus, in the development phase of a robotic system, the robotic engineers and developers must model, design and program their software while regarding the non-functional goals. Guaranteeing these goals is considered as the intrinsic quality of the robotic software or system (see Section 1.1 for the product quality model);



(A) In the same control and decision loop.



(B) In two different control and decision loops: one for executing the functional goals and another for guaranteeing the non-functional goals.

FIGURE 3.7: Autonomy enhancement by guaranteeing both functional and non-functional goals.

- **The second approach in Figure 3.7b adds another closed loop of perception, decision and control to ensure non-functional goals.** The loop of the guarantee of non-functional goals can be considered to be on top of the loop of functional autonomy. The integration of this loop is generally independent of the robotic system development phase and the guarantee of non-functional goals is considered as guaranteeing the quality in use of the robotic system during the deployment phase (see Section 1.1 for the quality in use model). However, it is important to merge after these two control loops to avoid conflicts between actions.

Indeed, these two approaches are equivalent to the internal and external adaptation approaches of self-adaptive systems presented in Section 2.2. The work in [Salehie and Tahvildari, 2009] has indicated that the main advantage of the first approach is the ability to handle local adaptations. However, it has some notable drawbacks in terms of maintainability and scalability. While the second approach offers a significant advantage of flexibility, maintainability and scalability. Our thesis focuses on the quality in use of the robotic missions and the second approach is really our concern.

3.2.7 Multi-Mission Context

When the complexity of the modern mobile robotic systems increases in terms of number of components, such as in the case of autonomous cars [Kato et al., 2015], the robot user or operator can demand many robotic missions deployed in the same time. This leads to multi-mission context as defined in Figure 3.8, where many missions can share the same resources for executing the functional requirements of each mission. These shared resources can be considered to be mutex locked or superposed. Many missions can use the superposed resources in the same time such as computing resources, sensors and energy resources, but must exclusively use the mutex locked resources such as actuators like motors, because they cannot receive and execute two different commands simultaneously. As a consequence, the autonomy challenge becomes more demanding in order to cope with more complex and dynamic operational circumstances, as well as to guarantee the NFRs of each mission. In this case, the adaptation approach needs to coordinate between missions with different priorities, as well as to make these coherent between adaptation levels.

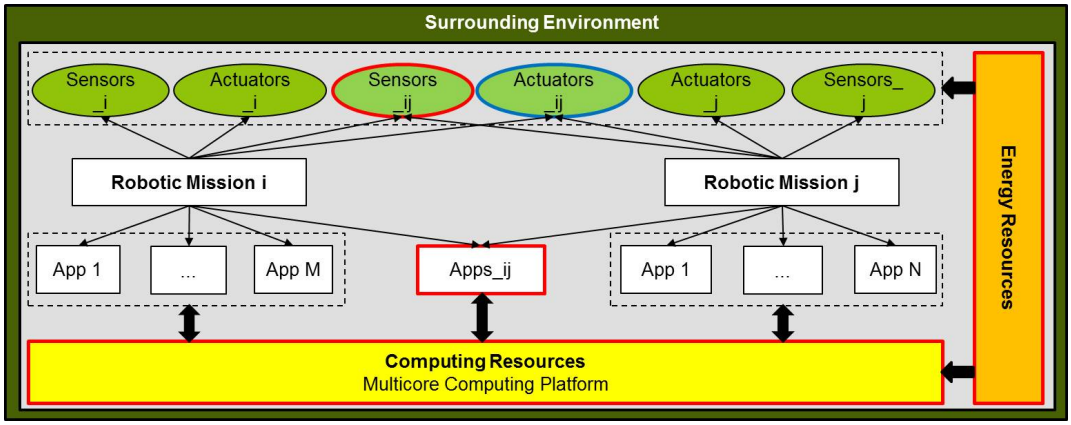


FIGURE 3.8: Multi-Mission context. Here, two example missions i and j can share resources (sensors, actuators, computing resources) and energy in a mutex locked (blue contour) or superposed manner (red contour).

3.3 Systematic Approach for Characterizing Robotic Missions

We intend to propose a systematic approach to find the key mission configuration knobs and to clarify how these knobs can influence mission non-functional requirements. The characterization process is described in 6 steps (Figure 3.9):

- **Step 1: Identify mission functional objectives**. As described above, the functional objectives represent the functionalities of the robotic mission. All robotic

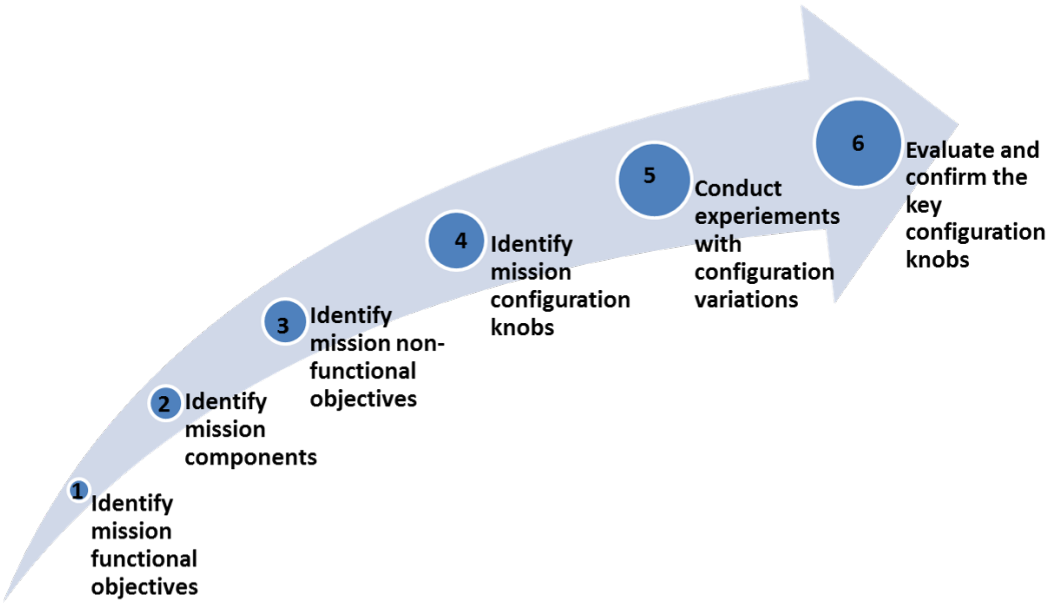


FIGURE 3.9: Systematic approach of 6 steps for characterizing robotic missions.

functionalities must be implemented at the design time and we do not focus on these objectives in our run-time adaptation framework. But it is important to identify these objectives in order to identify the components of the mission in the next step;

- **Step 2: Identify mission components**. In this step, we will identify all sensors, actuators and processing applications in the robotic mission. Then, the mission's energy consumption model can be formulated as in Equations (3.3), (3.4) and (3.5);

- **Step 3: Identify mission non-functional objectives**. We must then identify the non-functional objectives of the mission. While energy consumption constraints are general for all missions, performance and some other constraints may be mission-specific. It is therefore necessary to have a clear understanding of the mission. This step is important because our run-time adaptation framework will focus on these non-functional objectives. Of course, we do not need to identify all the non-functional requirements, but the requirements we would like to manage;

- **Step 4: Identify mission configuration knobs**. This step identifies the configuration knobs of the mission. Figure 3.6 summarizes the possible configuration knobs. We must have the ability to dynamically tune these knobs. We do not limit the number of configuration knobs, but mission-specific knowledge can help reduce the number of configurations considered, and then limit the number of experiments in the next step;

- **Step 5: Conduct experiments with configuration variations**. In this step, we can deploy the mission either in simulation or on the real robotics platform. In both cases, different environmental scenarios must be generated in order to take into account the impact of environmental dynamics in the context of mobile robotics. For each configuration change, we deploy the mission and record the measured results of the non-functional objectives. These results will be processed and analyzed in the next step;

- **Step 6: Evaluate and confirm the key configuration knobs**. In this last step, by analyzing the results of the experiments for each configuration, we can conclude on the qualitative relationship between the configuration knobs and the non-functional objectives. Quantitative models can also be formulated, for example, by regression analysis, but the accuracy of these models is a problem due to many dynamics during the mission that we cannot anticipate in the experiments performed. From these qualitative relationships, we can confirm the key configuration knobs that contribute significantly to non-functional objectives.

These steps are proposed to help characterize the robotic missions in a systematic and orderly manner. A generalized tool or automated characterization does not seem possible for this characterization at the mission level.

3.4 Self-Aware Mobile Robotic Mission

The characterization described above can be considered as an offline characterization that provides unified concepts of mobile robotic missions as well as a qualitative understanding of their dynamic characteristics. However, the run-time adaptation will be deployed during mission operation and requires the observation at run-time for the decisions to be taken. The self-awareness of mobile robotic mission will therefore be responsible for this task.

In the literature, there are some work that develop the framework for the data and event collection in robotic systems. These observations can be stored in memory to facilitate the offline analysis such as the work in [Shrewsbury et al., 2013] or the run-time management framework can use them directly as real-time observations such as the work in [Bihlmaier and Wörn, 2014; Forouher, Hartmann, and Maehle, 2014; Huang et al., 2014; Twigg, Gregory, and Fink, 2016].

In our framework, the self-aware mobile robotic mission is equipped with a monitoring service that provides online metrics that represent the internal and external situation of the robotic mission:

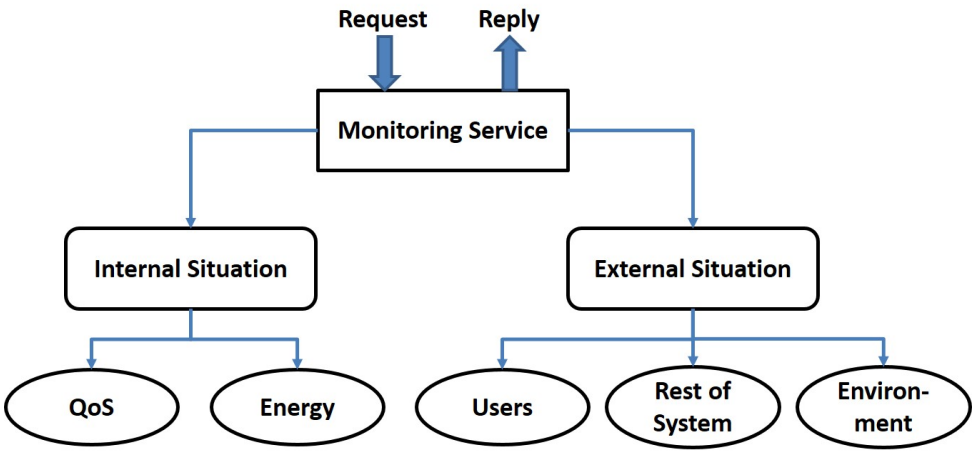


FIGURE 3.10: Self-aware mobile robotic mission: monitoring methodology. A monitoring service of the mission monitors both internal and external situation. The internal situation relates to the current state of quality of service and energy consumption. The external situation relates to the users, the rest of system and the environment.

- The **internal situation** indicates the current quality of service and energy efficiency of the robotic mission;
- The **external situation** represents the context surrounding the mission, such as the user's needs, the environment and the rest of the robotic system.

The characterization process can help to identify the necessary metrics in each monitoring group. All metrics will be grouped into a monitoring service with the role of a service provider. The monitoring diagram is depicted in Figure 3.10. By proposing this monitoring methodology, we also indicate that our management framework will be based on the data or monitored metrics. We will monitor the necessary metrics, but not the system's events. We can predict how an event will influence certain metrics, but it is difficult to deduce which events occurred with monitored data. Briefly, our management framework will be data-driven.

3.5 Mobile Robotic Framework for Implementation and Validation

The purpose of this section is to present the simulation and the real framework used in our thesis to test and validate the methodology we propose. The simulation framework means that the robotic mission is deployed in a simulator such as Gazebo and the real framework means that the mission is deployed in a real robotic platform. The simulation framework is used throughout our work to progressively evaluate our methodology while the real framework is used in the final step to validate the feasibility of the run-time management proposed in the real robotic platform.

3.5.1 Real Framework

The real framework means that the robotic mission is deployed in a real robotic platform and in the real environment. We do not intend to target any specific robotic platforms or environments. However, for sake of simplicity and availability in our laboratory, we have chosen the robotic platform based on the Pioneer-3DX[1] mobile base and powered by an NVIDIA Jetson AGX Xavier[2] embedded platform and this mobile robotic platform will operate in an indoor environment such as inside a building floor. The full system and its components are shown in Figure 3.11. The three main subsystems (see the generic structure of mobile robotic systems in Section 2.1.1) of the real framework are described as follows.

Computing Subsystem

Our mobile robotic platform is equipped with a NVIDIA Jetson Xavier embedded platform. This platform runs a Linux operating system and its robotic applications are developed and executed in ROS. The characteristics of the Xavier board are detailed in Appendix A. Small in size and low in power consumption, this embedded system harnesses the computing power of 8-core ARM processor and 512-core Volta GPU to execute efficiently and in real time many robotic applications such as object detection, people detection, scene segmentation, etc.

[1] See https://www.generationrobots.com/fr/402395-robot-mobile-pioneer-3-dx.html
[2] See https://developer.nvidia.com/embedded/buy/jetson-agx-xavier-devkit

Sensing and Acting Subsystem

The sensing and acting subsystem is based on a Pioneer-3DX mobile base and two external sensors such as a Kinect[3] sensor and a Hokuyo[4] laser scanner. The details of Pioneer-3DX can be found in Appendix B. Some sensors such as the odometer, sonar and bumper are integrated into the mobile base. The Kinect sensor provides depth and color information of the scene in front of the robot. The Hokuyo laser scanner provides the distance between the robot and surrounding obstacles.

Power Supply Subsystem

All the system is powered by a module of three batteries embedded in the Pioneer-3DX mobile base. Unfortunately, this module do not provide the information of power consumption and battery state of charge. Thus, we propose two measurement tools to obtain the measurements of power consumption. A software tool named PowerAPI [Noureddine, Rouvoy, and Seinturier, 2015] is in charge of estimating the power consumption of robotic applications on the computing hardware. And a hardware tool called Yocto-Watt[5] is used to measure and estimate the power consumption of the sensing and acting part. The details how to use these tools will be found in Appendix C and D.

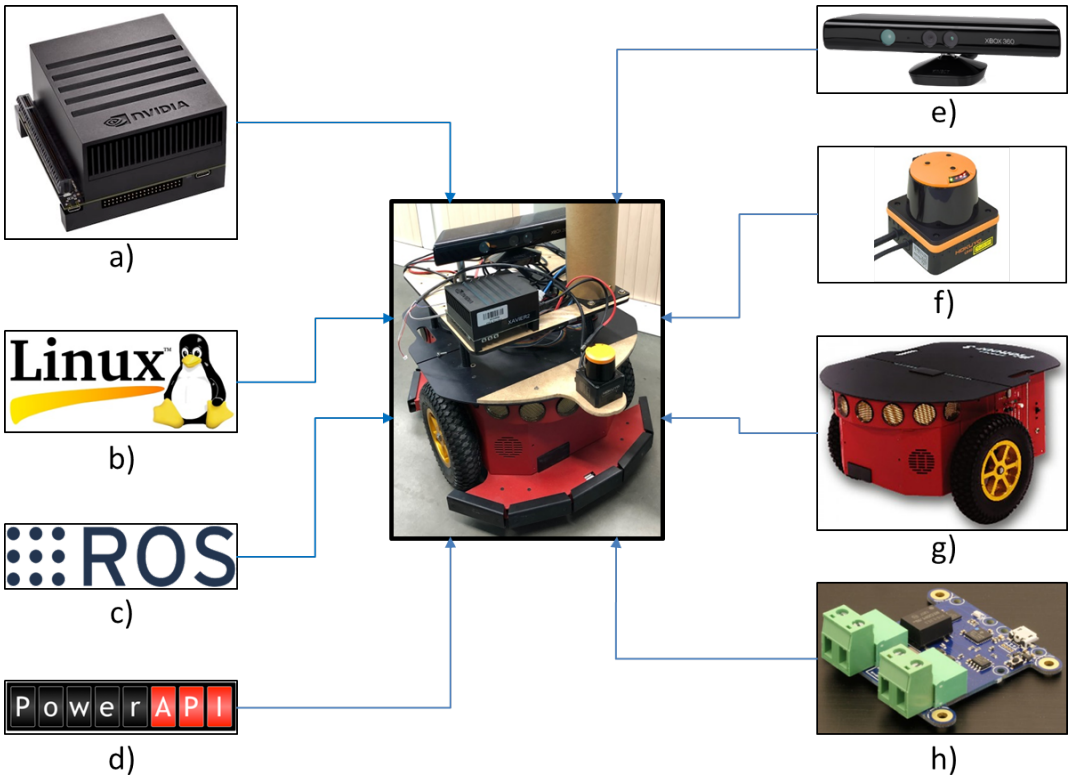


FIGURE 3.11: Our real robotic platform for experimentation: a) NVIDIA Jetson Xavier, b) Linux-based operating system, c) Robot Operating System, d) Process-level power estimation by PowerAPI, e) Kinect sensor, f) Hokuyo laser scanner, g) Pioneer-3DX mobile base and h) Power consumption measurement by Yocto-Watt.

[3]See https://www.generationrobots.com/en/401430-microsoft-kinect-sensor.html
[4]See https://www.generationrobots.com/en/401755-hokuyo-...
[5]See http://www.yoctopuce.com/FR/products/capteurs-electriques-usb/yocto-watt

3.5.2 Simulation Framework

The objective of the simulation framework is to reflect the real robotic framework and help us to have a quick prototype and validate our proposed methodology. The simulation framework is described in Figure 3.12. The three main subsystems are described as follows.

Computing Subsystem

The computing hardware is now an Intel core i7 desktop that launches the simulation framework including a Gazebo simulator[6] and robotic missions. It runs also a Linux operating system and ROS.

Sensing and Acting Subsystem

Gazebo is in charge of simulating the behavior of sensors and actuators necessary as described in the real framework. Figure 3.13 describes the data flow of simulated sensors and actuators subscribed and published by a Gazebo simulator.

Power Supply Subsystem

While the computing power consumption can be measured by PowerAPI as in the real framework, the sensing and acting power consumption needs an estimation model. The power estimation model proposed by the work of [Jaiem et al., 2016a]

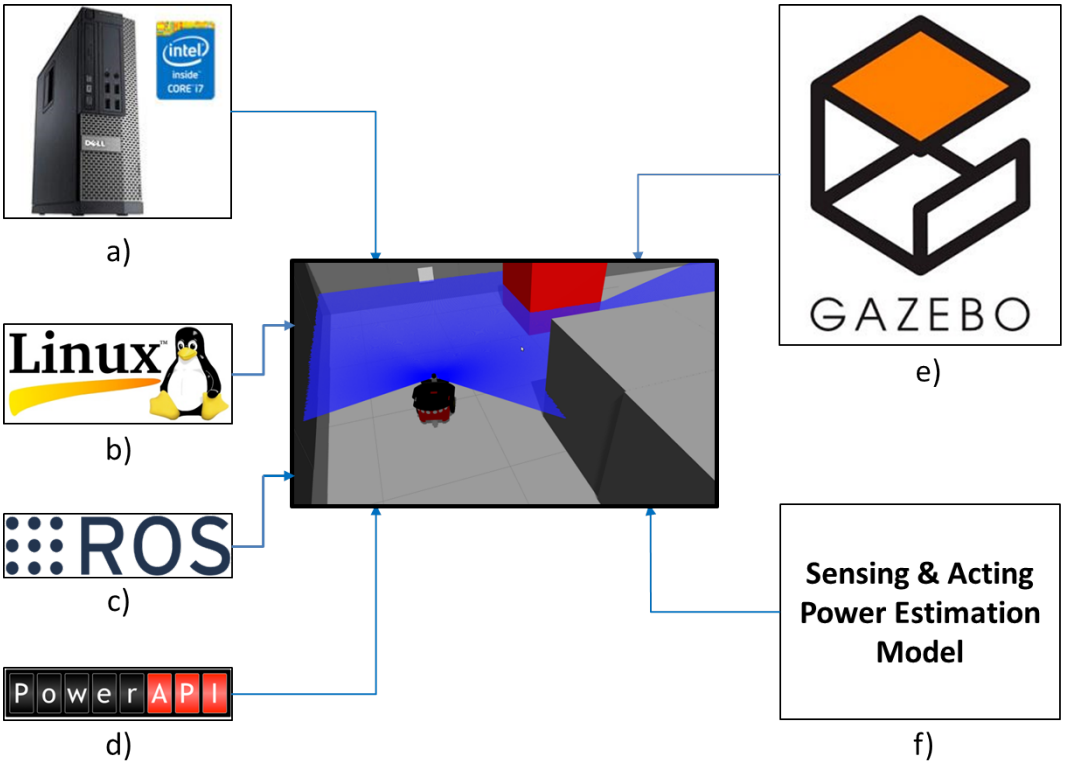


FIGURE 3.12: Simulation framework: a) Intel Core i7 desktop, b) Linux-based operating system, c) Robot Operating System, d) PowerAPI, e) Gazebo simulator and f) Sensing & acting power estimation model.

[6] See http://gazebosim.org/

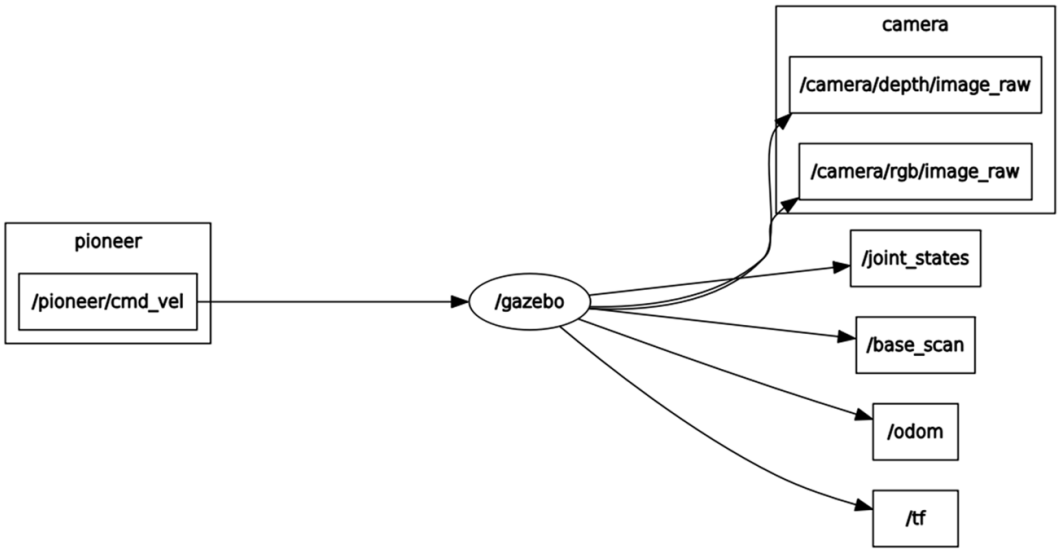


FIGURE 3.13: Simulating robot's sensors and actuators in Gazebo. The robot simulated in Gazebo receives the velocity command from the navigation's applications for controlling simulated actuators (motors), and publishes the data from simulated sensors such as depth and color images of RGBD camera, LIDAR scanner, odometry, etc.

TABLE 3.1: Energy estimation models used in the simulation framework [Jaiem et al., 2016a].

P_{motors}	$6.25v^2 + 9.79v + 3.66W$
P_{LIDAR}	$2.34W$
P_{Kinect}	$2.82W$
$P_{Controller}$	$2.67W$

is leveraged for our simulation framework. In this model, the power consumption of robot's motion is defined as a quadratic function of robot velocity and the power consumption of sensors such as Kinect and Hokuyo laser is considered static. Table 3.1 summarizes the energy estimation of the components used in our simulation framework.

Of course, the power estimation in the simulation framework cannot represent the real values of the power consumption in the real framework. These models only represent relatively the way how the power consumption will change depending on the changing configuration. Thus, the adaptation model that is found in the simulation framework needs to be tuned in order to apply into the real framework. This point will be mentioned in the next chapters.

3.6 Mobile Robotic Missions: Case Studies

In this section, we present three motivational mobile robotic missions with the concepts proposed above. We develop these missions mainly from the open-source ROS software packages and integrate them into our simulation and real frameworks. It is also important to note that the configuration of robotic missions is not always

reconfigurable. Thus, we need to extend the reconfiguration capability of these missions by using the *dynamic_reconfigure*[7] ROS package. These missions then serve as case studies to validate our Mission Manager and Multi-Mission Manager methodology proposal. The characteristics of these missions are finally resumed in Tables 3.6 and 3.7.

3.6.1 Autonomous Navigation Mission

The *autonomous navigation mission* takes advantage of a ROS Navigation Stack[8] which allows the robot to navigate autonomously between two predefined target points based on an existing environment map (usually a 2D laser-based map generated by a previous mapping process) and to receive information from certain range sensors such as LIDAR and/or Kinect (Figure 3.14). This ROS-based mission can be broken down into several ROS nodes, each node performs one or more functional tasks as detailed in Table 3.2. In the real robotic framework, three ROS nodes *sensor_sources*, *odometry_source*, *base_controller* are represented by real sensors and actuators such as LIDAR, Kinect and Pioneer-3DX mobile base. In the simulation framework, Gazebo with virtual sensors and actuators will play the roles of these three nodes.

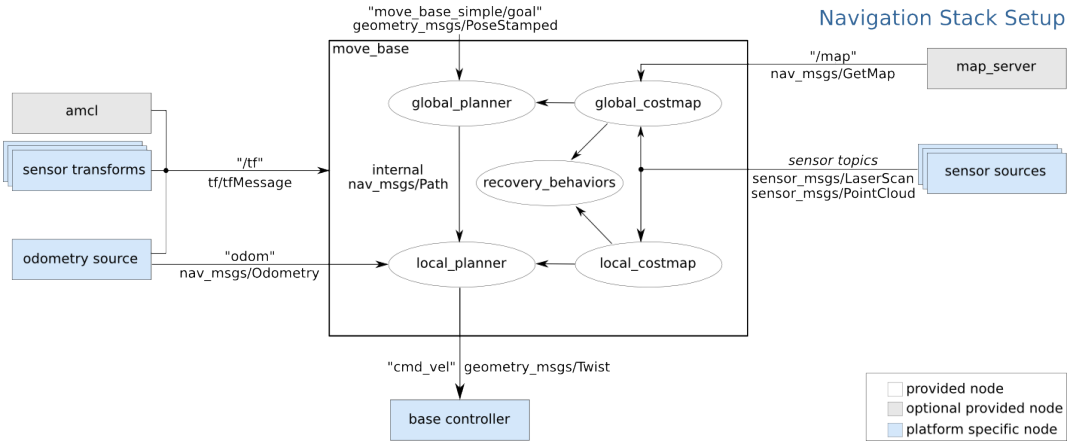


FIGURE 3.14: Autonomous Navigation Mission based on a ROS Navigation Stack. Its functional requirement is to visit a user-defined set of waypoints in the working environment based on a global map.

TABLE 3.2: Autonomous Navigation Mission: ROS nodes and their functional tasks.

ROS Node	Functional Tasks
map_server	providing an existing map for navigation
amcl	keeping robot localized in the map
move_base	updating the environment informations from sensors and controlling robot moving
sensor_sources	capturing the environment informations
odometry_source	capturing the robot odometry information
base_controller	receiving the velocity command from move_base node and controlling robot actuators

[7] See http://wiki.ros.org/dynamic_reconfigure
[8] See http://wiki.ros.org/navigation

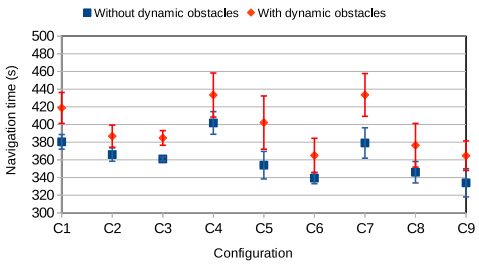
Motivation for Run-Time Adaptation

In this mission, its functional requirement is represented by the navigating task to visit a user-defined set of waypoints in the working environment. The non-functional requirements can be applied to this mission such as navigation time, energy consumption and safety to obstacles. A quantified characterization of the navigation mission has been realized and the results are shown in Figure 3.15. The objective is to clarify the impact of two key parameters of the navigation mission: max robot velocity (v_{max}) and control frequency ($f_{control}$) as chosen in Table 3.3 on the performance and the energy consumption of the navigation mission. The results are averaged over 10 deployment trials for each configuration and the standard deviation is also calculated. Two different environmental contexts are also simulated for characterization, one without dynamic obstacles and the other with dynamic obstacles. The dynamic obstacles mean some objects that do not appear in the static map of the environment but only while navigating.

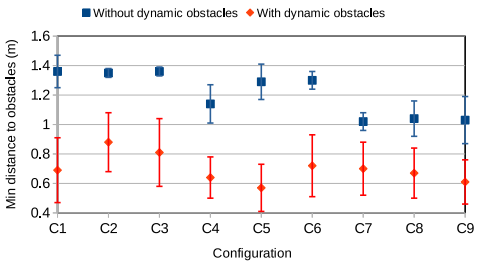
The characterization results give us two important qualifications. The first one relates to the fact that one configuration can give good results at some objectives but bad results for the others. For example, the *C9* configuration can give the best result for the navigation time objective, but increases the collision risk as evaluated by the

TABLE 3.3: Navigation configurations $\{v_{max} m/s, f_{control} Hz\}$ used for characterization.

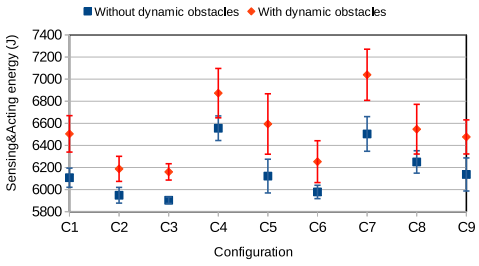
C1	C2	C3	C4	C5	C6	C7	C8	C9
0.65, 5	0.65, 10	0.65, 20	0.8, 5	0.8, 10	0.8, 20	1.0, 5	1.0, 10	1.0, 20



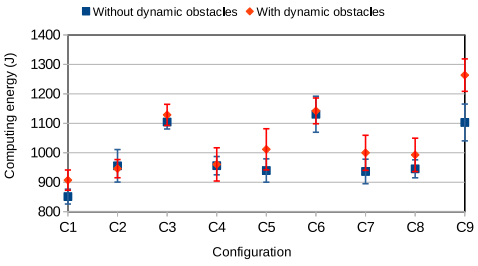
(A) Navigation time. The lower is the better.



(B) Minimum distance to obstacles. The higher is the better.



(C) Sensing&Acting energy consumption. The lower is the better.



(D) Computing energy consumption. The lower is the better.

FIGURE 3.15: Characterization results of a navigation mission in terms of performance and energy consumption. The results are also compared between two different environmental contexts: without and with dynamic obstacles.

minimal distance to obstacles and also consumes more computing energy. Hence, for a multi-objective or multi-criteria problem, choosing a configuration satisfying all objectives is non-trivial. The second qualification relates to the operational context changes such as the appearance of dynamic obstacles that will change the behavior of each configuration and give different results in term of performance and energy consumption. The larger variation is also realized in this case.

3.6.2 Video Server Mission

The second mission is a *video server mission* using some modules of Robot Web Tools[9] to provide the Kinect-based information in the form of an encoded depthcloud of the robot environment that can be viewed on a remote web browser or used for post-processing. Now, the robot can be considered as a server that provides the visual information surrounding the robot. This will be useful for security surveillance, for example. There are three nodes in this application as detailed in Figure 3.16 and Table 3.4. Similar to the navigation mission, this mission can be deployed both in the simulation and real framework.

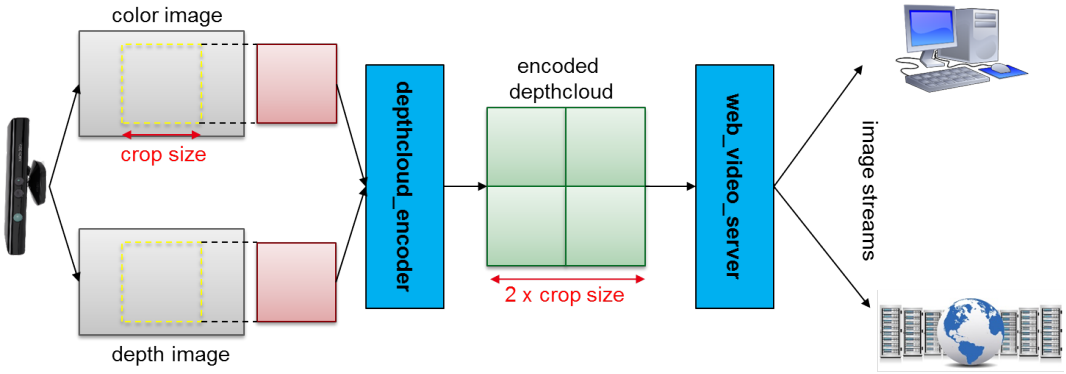


FIGURE 3.16: Video Server Mission. Its functional requirement is to provide the visual information of the working environment in the form of encoded depthclouds for the users.

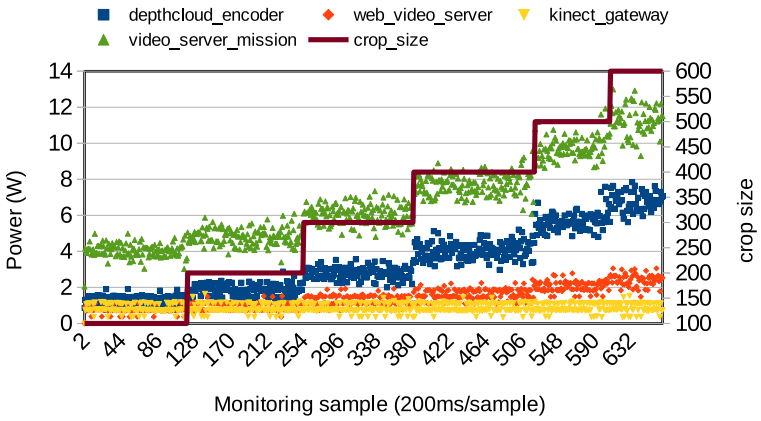
TABLE 3.4: Video Server Mission: ROS nodes and their functional tasks

ROS Node	Functional Tasks
kinect_source	capturing the depth and color images
depthcloud_encoder	encoding the depth and color image into a single image stream
web_video_server	opening a local port and waiting for http requests from user web browsers

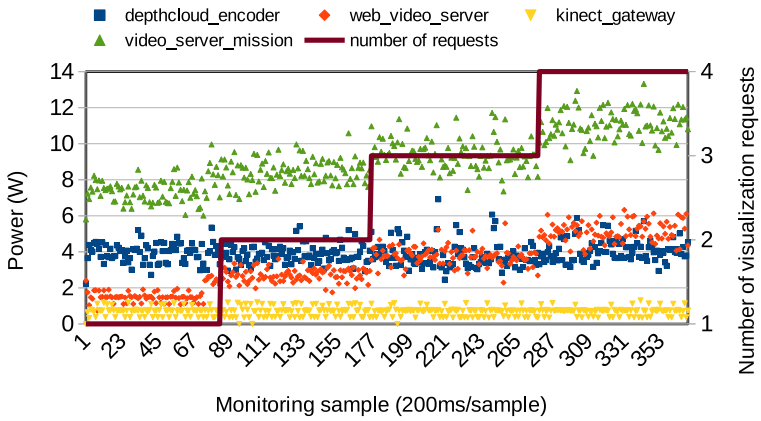
Motivation for Run-Time Adaptation

The functional requirement of this mission is to provide the visual information of the robot's surrounding environment in the form of encoded depthcloud. The non-functional requirements can be taken into account such as target resolution (equal to $2 \times crop\_size$), energy consumption and frame rate. Figure 3.17 denotes some

[9] See http://robotwebtools.org/



(A) crop_size vs. power consumption with 1 visualization request.



(B) visualization requests vs. power consumption with *crop_size=400*.

FIGURE 3.17: Characterization results of Video Server Mission in terms of performance and energy consumption.

characterization results of the video server mission. We can realize the dynamic behavior of power consumption when we change the *crop_size* parameter or the number of visualization requests. Figure 3.17a indicates that when the *crop_size* is increased, the target resolution and the power consumption of the mission are also increased because the mission must encode (by *depthcloud_encoder*) and transmit (by *web_video_server*) more information. When the number of visualization requests is increased, while the *crop_size* is kept static, the mission must transmit (by *web_video_server*) more information, the power consumption is therefore increased as depicted in Figure 3.17b. In addition to the number of visualization requests, the user can require different visualization parameters[10] such as visualization quality, resolution, etc. These variabilities then influence also the non-functional properties of the video server mission.

3.6.3 Semantic Environment Understanding Mission

To enrich our mission database, we have chosen to add a semantic environment understanding mission. The idea is to detect and recognize a certain amount of

[10] See http://wiki.ros.org/web_video_server

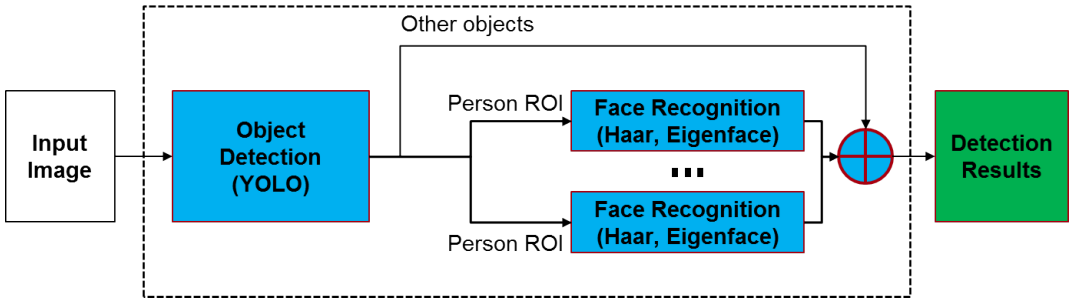


FIGURE 3.18: Semantic Environment Understanding Mission. Its functional requirement is to understand the scene by detecting and recognizing the objects and people in the robot's working environment.

TABLE 3.5: Semantic Environment Understanding Mission: ROS nodes and their functional tasks.

ROS Node	Functional Tasks
input image	capturing images
object detection	detecting and recognizing objects in input image
face recognition	detecting and recognizing faces in person ROIs

object classes and give the mission manager the ability to balance between the detection accuracy, the power consumption and the throughput (frame rate or number of processed frames per second) of the algorithm. To do that, we have considered a convolutional neural network (CNN) based algorithm implementing the object detection and recognition called YOLO [Redmon et al., 2016]. YOLO is a state-of-the-art real-time object detection system that is suitable for a resource-constrained environment such as mobile robotic systems. However, the most important reason we choose YOLO is that we can easily change the size of the inference model to make a compromise between speed and accuracy without the need for retraining. YOLO discriminates 80 object classes by taking as input the visual information (the 640x480 RGB image given by the Kinect sensor in our case) and returns a resulting image with bounding boxes around the detected objects as well as labels with their names.

To add some extra computation complexity depending on the scene, we have added a face recognition algorithm activated whenever a person was detected by YOLO. Indeed, we send the person's region of interest (ROI) to the facial recognition module named EigenFaceRecognizer based on OpenCV[11]. The result of the face recognition is displayed on the console output and may be used to verify the authenticity of the person. Thus, the final result of the mission is the resulting image with the bounding boxes and the names of the detected objects and the names of the detected persons in the scene. The mission is depicted in Figure 3.18. The ROS nodes and their functional tasks are shown in Table 3.5. Due to the complexity of simulating people and their faces in Gazebo, we only propose this mission for deployment in the real framework.

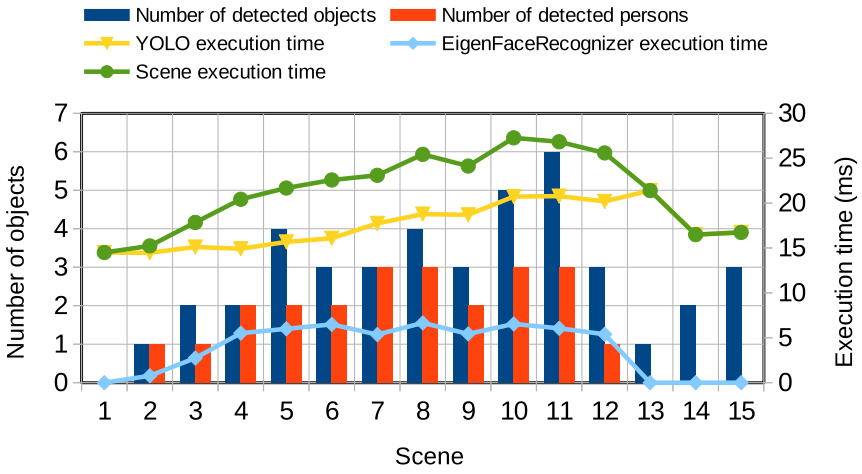


FIGURE 3.19: The non-functional properties of the semantic mission depend on the scene. In this case, 15 scenes with a different number of objects and people detected give different execution times (inverse of the frame rate) and therefore the power consumption of the semantic mission.

Motivation for Run-Time Adaptation

The efficiency of this semantic mission is very dependent on the scene. Indeed, Figure 3.19 shows the impact of scenes on non-functional properties such as the execution time (inverse of the frame rate) and therefore the power consumption of the semantic mission. The execution time of a scene or frame is divided into two parts: one for the YOLO algorithm to detect objects (including people), and another for the EigenFaceRecognizer algorithm to detect and recognize faces. We can easily see that the EigenFaceRecognizer algorithm only takes the execution time when there are people in the scene, and this processing time is relatively stable, for example from the scene 4 to 12. While the execution time of the YOLO algorithm depends on the number of objects and their size in the scene. In short, 15 different scenes in Figure 3.19 give different non-functional properties of the semantic mission. And the different YOLO inference models also have different impacts on this mission as explained below.

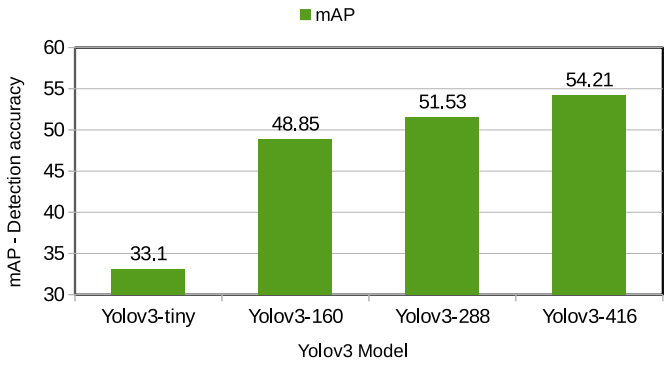
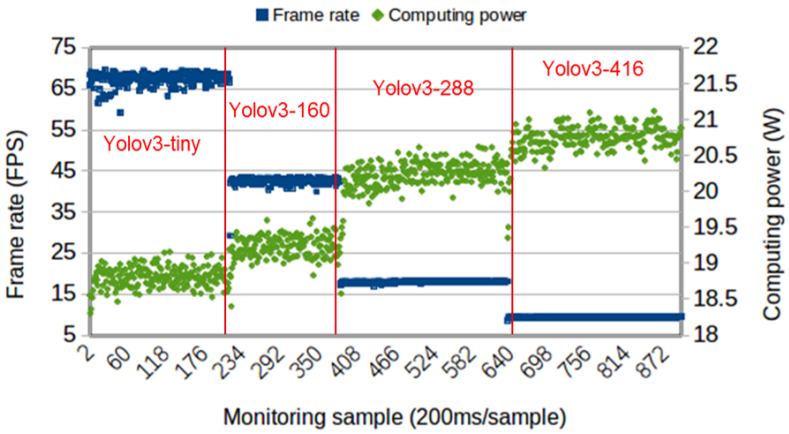
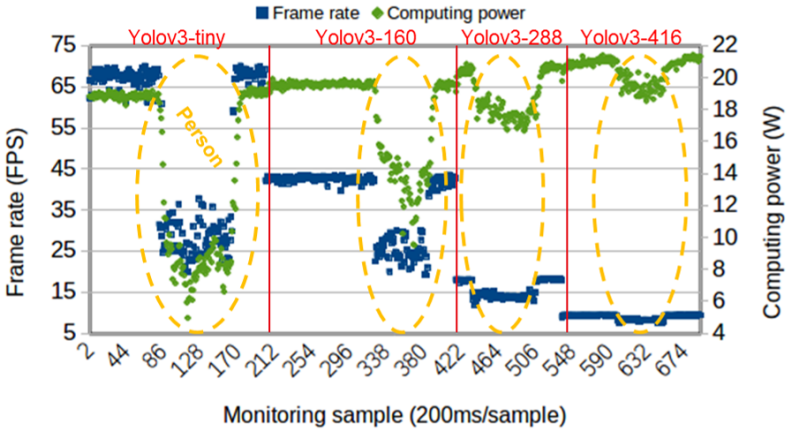


FIGURE 3.20: Detection accuracy of four Yolov3 models.

[11] See https://docs.opencv.org/3.4/dd/d7c/classcv_1_1face_1_1EigenFaceRecognizer.html



(A) Yolov3 model vs. frame rate and power consumption



(B) Yolov3 model vs. frame rate and power consumption in case of person appearance

FIGURE 3.21: Characterization results of Semantic Environment Understanding Mission in terms of performance and energy consumption.

Taking into account the memory and power constraints of the Xavier board, we deploy four small and medium inference models as shown in Figure 3.20. Indeed, the postfixes such as *-tiny*, *-160*, *-288*, *-416* determine the size of the inference model. The *-tiny* version is the smallest one. As above mentioned, we can change the size of the model without the need for retraining. To optimize the switching time from one YOLO model to another, we preload all the four models into memory and simply change the pointer's memory address when switching to another model. The detection accuracy of the YOLO model is calculated by a common metric for evaluating the object detection system called the mean average precision (mAP)[12]. In fact, the authors of YOLO give the mAP for some models like *Yolov3-tiny*, *Yolov3-320*, *Yolov3-416*, *Yolov3-608* and *Yolov3-spp*[13]. We make then a linear estimation to obtain the detection accuracy of the others. Figure 3.20 indicates that the detection accuracy increases when the size of the model increases. Meanwhile, the frame rate decreases significantly with the accuracy while the power consumption increases as depicted in Figure 3.21a.

The impact of the appearance of a person in the scene is shown in Figure 3.21b. When a person is detected by the Yolo algorithm, the facial recognition module will

[12] See http://cocodataset.org/#detection-eval

[13] See https://pjreddie.com/darknet/yolo/

be called and takes its time to detect and recognize the person's face. We can see clearly in Figure 3.21b the reduction of the frame rate when a person is detected as marked in the yellow eclipses. The variation is also realized with the power consumption. For different Yolo models, the variations are also different. Thus, the adaptation of inference models as well as number of processed persons is important to tradeoff between detection accuracy, frame rate and power consumption of the mission.

TABLE 3.6: Example of mobile robotic missions: components of sensors, actuators and applications.

Mission	**Sensors**	**Actuators**	**Applications**
Navigation	LIDAR, odometer	motors	mapping, localizing, navigating apps, and LIDAR, mobile base communicating apps
Server	Kinect		encoding, transmitting apps, and Kinect communicating apps
Semantic	Kinect / Camera		detecting and recognizing objects, detecting and recognizing faces

TABLE 3.7: Example of mobile robotic missions: requirements and key parameters.

Mission	**Requirements**		**Key Parameters**
	FRs	**NFRs**	
Navigation	visiting a user-defined set of 2D map-based waypoints	navigation time, energy consumption, safety to obstacles	robot velocity, control frequency
Server	providing encoded depthcloud of user-defined working area	target resolution, energy consumption, frame rate	crop size
Semantic	recognizing objects and person identity	detection accuracy, energy consumption, frame rate	type of network, number of person ROIs

3.6.4 Discussion

The results of the characterization of the three independent robotic missions above indicate that many dynamic operational conditions at run-time can have a significant impact on the mission's quality. A fixed configuration, a fixed set of mission parameters in our case, cannot adapt to all dynamic factors while meeting non-functional requirements. In some scenarios, a configuration that meets all non-functional requirements may exist, but the difficulty of finding this configuration is also a challenge, usually through an off-line optimization process such as the work in [Cano et al., 2016; Kabir et al., 2017; Berkenkamp, Krause, and Schoellig, 2016]. Therefore, the one-fit-all strategy seems ineffective in the context of mobile robotics.

Our vision targets the run-time adaptation at the mission level in the robotic system. This means that the mission parameters should be reconfigurable to adapt to the current operational conditions of the robotic mission. By observing the internal context such as the current level of quality of service, energy consumption and the external context that influences the mission behavior, a sequence of actions or

configurations can be chosen dynamically to meet the mission's non-functional requirements.

3.7 Summary

In this chapter, we have proposed concepts to characterize robotic missions. These concepts are unified in order to generalize the characterization of the different types of robotic missions. Many aspects of the mission are studied in order to understand its structure and in particular the quality of service and resource consumption. A systematic approach is also proposed to characterize the mission. This characterization phase is a premise for implementing our run-time management framework in future chapters.

For prototyping and validating our proposed methodology, two frameworks are presented. The real framework is based on Pioneer-3DX mobile robot and NVIDIA Jetson Xavier embedded platform and the simulation framework is deployed in Gazebo simulator. Moreover, three examples of mobile robotic missions are also described and analyzed to understand their dynamic characteristics. These missions will be then used as case studies to validate our run-time management framework.

By studying the three different mobile robotic missions, we see the dynamic characteristics of these missions and the preference for the run-time adaptation over the one-fit-all strategy is also highlighted. In the next chapter, we will present the methodology for designing a self-adaptive mission manager, aware of quality of service and energy, based on the reinforcement learning.

Chapter 4

QoS and Energy-Aware Self-Adaptive Mission Manager

Contents

Mobile robots can cover a large range of real life missions such as security surveillance, environmental monitoring, delivery, search and rescue missions. Such missions require different levels of autonomy in order to react to the dynamic factors during robot operation such as environmental condition change, computing demand fluctuation and system resource availability in respect to the mission performance requirements. These reactions should be taken at run-time and adaptively because the off-line configuration cannot anticipate all the dynamic, unforeseen and uncertain events.

The demand for the run-time adaptation and the fact that the mobile robot behavior is strongly dependent on the interaction between the robot and its surrounding environment make us think on the machine learning-based technique for decision-making, specifically reinforcement learning (RL). In this technique, an agent (a mobile robot here) observes its environment, makes some trial-and-error steps of decision-making and finally finds the policy mapping a suitable action at a given state. An agent can know or not the model of its environment. This is one of the advantages of RL for the autonomous context because the dynamic factors are stochastic and difficult to predict and model [Brugali, Capilla, and Hinchey, 2015].

In this chapter, we propose a reinforcement learning based approach for the design of a quality of service (QoS) and energy-aware robotic mission manager. The manager takes into account the metrics related to the performance and the resource consumption of the mission and generates the adaptive mission-specific parameter configuration. To the best of our knowledge, our study is the first one that applies the machine learning based technique, specifically reinforcement learning technique, for managing dynamically the mission's QoS and energy consumption on an autonomous mobile robot.

The chapter is organized as follows. Firstly, some related work are mentioned in Section 4.1. The problem of Mission Manager is then formally stated in Section 4.2. Sections 4.3, 4.4 and 4.5 present our methodology and two approaches based on Q-Learning and Deep Q-Learning. The methodology is then detailed in Section 4.6. We then prototype and validate our proposed methodology with some case studies and their results are presented in Sections 4.7 and 4.8. Finally, Section 4.9 concludes the chapter.

4.1 Related Work

Our methodology is based on the reinforcement learning technique for managing at run-time the non-functional goals such as performance and energy consumption for the mobile robotic missions. In the literature, the (Deep) Q-Learning technique has been applied to the run-time resource, performance and energy management from small scale such as embedded systems like the work in [Shen, 2013; Islam, 2015; Biswas et al., 2017; Zhao et al., 2018] to large scale such as data centers like the work in [Farahnakian, 2014; Chasparis, 2015; Mao et al., 2016; Liu et al., 2017; Cheng, Li, and Nazarian, 2018]. For the mobile robotic domain, RL techniques have been employed for reactive navigation or optimal low-level control such as the work in [Papierok, 2008; Tai, 2017; Chen et al., 2017]. Bringing the RL-based methodology

from other domains to mobile robotic domain is non-trivial, but needs a rigorous consideration of the robotic context with many dynamic, uncertain and unforeseen operational circumstances.

Most of mobile robotic missions are deployed without taking into account the run-time circumstances and the configuration is usually chosen by an offline optimization process. The work of [Cano et al., 2016] can be considered as a significant contribution to the offline optimization of a set of mobile robotics system parameters. With the same research objective, the previous efforts in the works presented in Chapter 2 [Jaiem et al., 2016b; Brugali et al., 2018; Zhang, Lu, and Hu, 2009; Inglés-Romero et al., 2013] tend to reconfigure the system in order to cope with run-time variations while respecting some non-functional requirements. Nevertheless, most of them are model-based approaches, where a set of analytical models of the robotic system, the environment, as well as the non-functional goal models needs to be built at the design time. The modeling process is non-trivial and the model accuracy is critical for guaranteeing the efficiency of run-time adaptation. Our learning-based methodology reduces the burden and the complexity of the modeling process at the design time by a data-driven approach. The important metrics are monitored actively and the learning-based decision-making phase is based on these metrics in order to make adaptation decisions. Moreover, the online learning phase can deal with uncertain and unforeseen operational circumstances at run-time.

4.2 Problem Statement

We focus on the non-functional requirements/goals of the robotic missions with the three main categories as presented in Section 3.2.4:

- **Performance goals** are represented by a set of n goals:

$$G_{perf} = \{gp_1, gp_2, ..., gp_n\} \tag{4.1}$$

- **Energy goals** are divided into sensing and acting energy consumption goal, and computing energy consumption goal:

$$G_{energy} = \{ge_{sens\&acts}, ge_{comp}\} \tag{4.2}$$

- **Other non-functional goals** are represented by a set of m goals:

$$G_{others} = \{go_1, go_2, ..., go_m\} \tag{4.3}$$

Finally, the non-functional goals applied to the robotic mission are modeled by

$$G_{Mission} = \{G_{perf}, G_{energy}, G_{others}\} \tag{4.4}$$

Moreover, the robotic mission possesses also a set of parameters that can be tuned dynamically in order to change the performance level, the energy consumption as well as the other non-functional properties. The set of l **mission-specific parameters** is represented by

$$K_{Mission} = \{k_1, k_2, ..., k_l\} \tag{4.5}$$

The problem is now stated by how to configure dynamically the set of parameters $K_{Mission}$ to take into account the dynamic operational circumstances of mobile robotic

systems at run-time for guaranteeing the set of non-functional requirements $G_{Mission}$. Indeed, the considered requirements are often contradictory (such as performance and energy consumption for example), our objective is to propose an approximately good solution with the best compromise between these requirements in all possible contexts, in other words, a solution that minimizes the level of violation of these requirements.

4.3 Proposed Methodology

4.3.1 Overview of Self-Adaptive Mission Manager

Figure 4.1 aims to describe our general methodology for the design of a self-adaptive mission manager. Our methodology is based on the **MAPE-K** (Monitor, Analyze, Plan, Execute - Knowledge) paradigm known in the autonomic computing and self-adaptive system domain, where the system under consideration is abstracted by the robotic mission. The objective of a self-adaptive mission manager is to guarantee the performance and energy goals applied to the robotic mission by reconfiguring the mission-specific parameters during mission operation. The *Knowledge* element possesses the information and the characteristics of the robotic mission and will be shared with the other elements of the MAPE process:

- Knowledge for *Monitoring*: defines a set of important metrics that need to be monitored in order to represent the current observation of performance level and energy consumption of the robotic mission, as well as of the overall robotic system;
- Knowledge for *Analyzing & Planning*: possesses the required performance and energy goals of the robotic mission, the trade-off weights of these goals, as well as the configuration knobs of the mission;

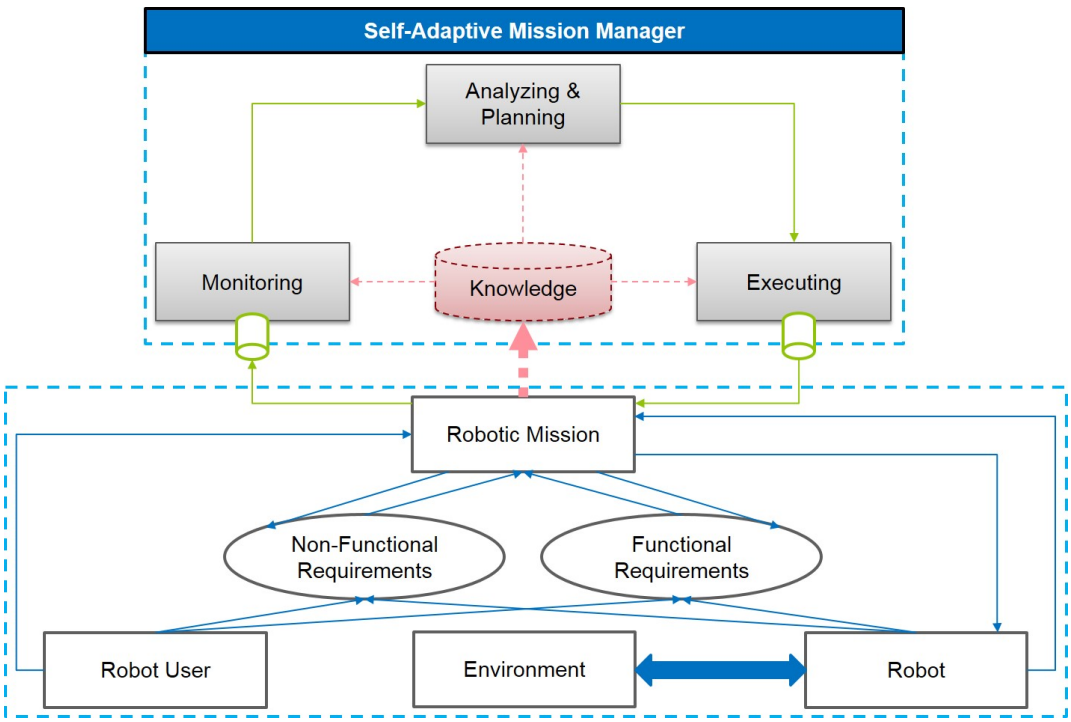


FIGURE 4.1: Overview of the structure of our Self-Adaptive Mission Manager.

- Knowledge for *Executing*: provides the configuration knobs (key mission-specific parameters) needed to be reconfigured in order to react to the current operational circumstances.

Normally, the knowledge is acquired through an off-line characterization phase of the robotic mission for figuring out the concepts mentioned in Chapter 3.

The closed loop of adaptation is then deployed at run-time with a MAPE adaptivity process. The *Monitoring* element collects at run-time the important metrics that represent the current observation of performance level and energy consumption of the mission, as well as of the overall robotic system. Then, the *Analyzing & Planning* elements, also considered as decision-making phase, analyze the monitored metrics, evaluate the current state of performance level and energy consumption, and make the suitable adaptation decisions. Finally, the *Executing* element receives the adaptation decisions from the decision-making phase and reconfigures correspondingly the parameters of the robotic mission.

Indeed, the structure of the self-adaptive mission manager reflects the external approach while addressing the non-functional requirements of robotic systems (see Section 3.2.6). The mission manager's control loop is considered independent and does not influence the control loop of the mission's functionalities. To do this, the mission's configuration knobs found in the characterization phase must not influence the normal functionality or stability of the mission. For example, for an autonomous navigation mission, the choice of the maximum speed of the robot v_{max} and the control frequency $f_{control}$ must be correlated to guarantee the reactive behaviour of the navigation, we cannot operate the robot at a high speed of $1.0m/s$ with a low control frequency of $1.0Hz$. In brief, the actions of the mission manager are constrained in such a way that there is no interference with the functional control loop.

In this paradigm, the three elements of *Knowledge*, *Monitoring* and *Executing* reflect directly the mission and are normally implemented in a mission-specific manner. We propose a generalization of the decision-making phase (*Analyzing & Planning*) to cover many types of robotic missions. The following section will present the motivation for a decision-making approach based on reinforcement learning.

4.3.2 Motivation for Applying Reinforcement Learning

In the context of autonomous and intelligent robotics, systems can act in real world environments for a long time without any form of external control or human intervention, and must have capacity to **perceive, think and act**. The performance of these systems depends heavily on their interaction with the working environment. In these systems, many levels of intelligence coexist as well as many perception, decision and action loops. For example, in an autonomous mobile robot, the lowest level of intelligence is the reactive navigation. It means that the navigation takes place in a sensory-motor loop to control the motion while detecting and avoiding the dynamic obstacles. Some higher levels can be deployed such as path planning, task allocation in multi-robot collaboration, etc. Our robotic mission manager is at such high level of intelligence where the mission manager is considered as an **autonomous agent**, and the rest of the robotic system and its surrounding environment as **working environment** (see Section 3.2 for the mission definition and granularity).

The objective of our robotic mission manager is to fulfill the mission performance requirements taking into account the resource constraints. For that, the manager must monitor actively the actual state of mission performance and resource utilization, then give an adaptive decision about the internal parameter configuration via

a **run-time decision-making** mechanism, and finally reconfigure the concerned parameters. In fact, during the mission operation, there are many dynamic, unforeseen factors such as environmental conditions, computing demand fluctuation and resource availability that contribute to the difficulty and the challenge to anticipate or model the working environment. This model-free decision-making problem can be well formulated and resolved by a reinforcement learning approach, more specifically by Q-Learning and Deep Q-Learning that we will present in the following paragraphs.

4.3.3 Generic Pattern for Formulating RL-Based Mission Managers

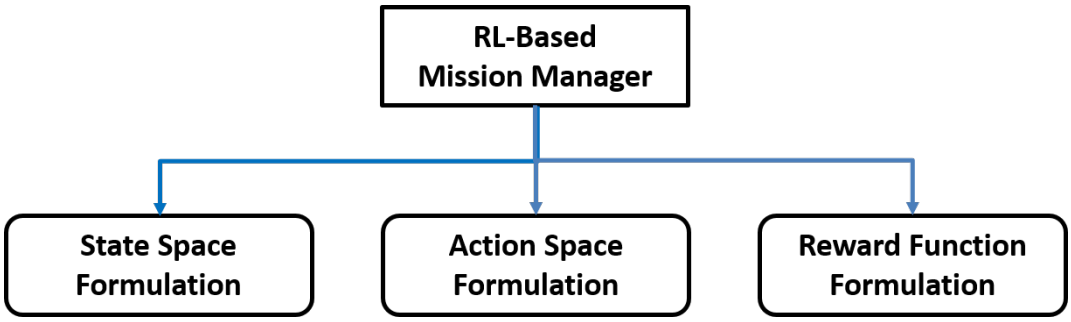


FIGURE 4.2: Generic pattern for formulating a RL-based Mission Manager.

As presented in Section 2.5, formulating a sequential decision-making problem based on the reinforcement learning approach is normally realized by determining three mandatory points (Figure 4.2):

- **State space formulation**: The state space is a set of possible states of the mission's internal and external contexts observed by the mission manager. The state is normally deduced from the monitored metrics provided by the mission's monitoring service (see Section 3.4);
- **Action space formulation**: The action space is a set of possible actions that the mission manager can take to reconfigure the mission (see Section 3.2.5);
- **Reward function formulation**: The reward can be considered as an important indicator to find the policy mapping between states and actions. For example, at a given state, an action that gives a better reward should be reinforced. For the problem of goal management, this reward can evaluate the level of goal guarantee.

Based on this generic pattern, we will develop the methodology based on the Q-Learning and Deep Q-Learning for the self-adaptive mission manager.

4.4 Q-Learning based Approach

4.4.1 State Space Formulation

The state identified by the mission manager reflects the current level of performance, energy consumption as well as other non-functional properties of the robotic mission represented by the corresponding set of run-time metrics:

- n_1 monitored metrics related to the performance goals:

$$M_{perf} = \{mp_1, mp_2, ..., mp_{n_1}\} \tag{4.6}$$

- n_2 monitored metrics related to the energy goals:

$$M_{energy} = \{me_1, me_2, ..., me_{n_2}\} \tag{4.7}$$

- n_3 monitored metrics related to the other non-functional properties:

$$M_{others} = \{mo_1, mo_2, ..., mo_{n_3}\} \tag{4.8}$$

In total, we have a set of monitored metrics:

$$M_{monitored} = \{M_{perf}, M_{energy}, M_{others}\} = \{m_i\}_{i=1}^{n_1+n_2+n_3} \tag{4.9}$$

These metrics are continuous, so we need to discretize them for a finite state space of Q-Learning. However, the discretization of a large number of metrics can lead to a very large number of states that increases the complexity of the Q-Learning algorithm. We choose to binarize these metrics into 0 and 1 by proposing the expected metrics m_i^* that are available in the knowledge for analyzing and planning (see Section 4.3.1):

$$M^*_{monitored} = \{M^*_{perf}, M^*_{energy}, M^*_{others}\} = \{m_i^*\}_{i=1}^{n_1+n_2+n_3} \tag{4.10}$$

and correspondingly comparing with the monitored metrics m_i. For each monitored metric m_i in the set of metrics and its corresponding expected metrics m_i^*, a binary indicator I_i is defined as follows

$$I_i = \begin{cases} 1 & \text{if } m_i \leq m_i^*, \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

Where,

- $I_i = 1$ means the non-functional goal is currently satisfied;
- $I_i = 0$ means the non-functional goal is currently violated.

A state s is therefore defined by $s = [I_1, I_2, ..., I_{n_1+n_2+n_3}]$ as illustrated in Figure 4.3. Thus, we have in total $2^{n_1+n_2+n_3}$ states in the state space.

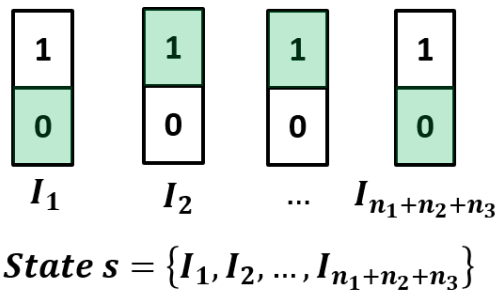


FIGURE 4.3: State formulation for the Q-Learning approach. A state space of $2^{n_1+n_2+n_3}$ states is formulated.

4.4.2 Action Space Formulation

The action given by the mission manager represents a configuration of mission-specific parameters $K_{Mission}$. We need also a finite action space for Q-Learning, so each element k_i in $K_{Mission}$ should be discretized into n_{k_i} values. Thus, we have $\Pi_{i=1}^{l} n_{k_i}$ actions. Figure 4.4 illustrates the action space of the Q-Learning approach.

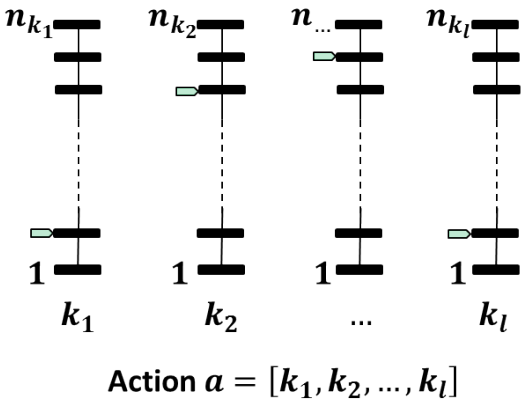


FIGURE 4.4: Action formulation for Q-Learning approach. An action space of $\Pi_{i=1}^{l} n_{k_i}$ actions is formulated.

Finally, with a finite state of $2^{n_1+n_2+n_3}$ states and a finite action space of $\Pi_{i=1}^{l} n_{k_i}$ actions, our Q-Table has $(2^{n_1+n_2+n_3} \times \Pi_{i=1}^{l} n_{k_i})$ Q-values in total.

4.4.3 Reward Function Formulation

The reward that the mission manager receives in this problem is the quantifiable evaluation of the current level of non-functional goal guarantees. This current goal guarantee level L can be calculated as a function (f_1, f_2, f_3) of the monitored metrics M_{perf}, M_{energy} and M_{others} and their respecting expected metrics M^*_{perf}, M^*_{energy} and M^*_{others} also known as reference values:

$$L_{perf} = f_1(M_{perf}, M^*_{perf}) \tag{4.12}$$

$$L_{energy} = f_2(M_{energy}, M^*_{energy}) \tag{4.13}$$

$$L_{others} = f_3(M_{others}, M^*_{others}) \tag{4.14}$$

The non-functional goal of metric m_i is satisfied if $L(m_i) \geq 0$. In fact, the reward formulation is specific to each mission and depends on the optimization requirement and the relative importance of the different goals. For example, if the optimization requirement is to trade-off all non-functional goals in order to minimize the violation level of each goal, the reward can be defined by the following weighted sum:

$$R = W_p \times L_{perf} + W_e \times L_{energy} + W_o \times L_{others} \tag{4.15}$$

Where W_p, W_e, W_o are the corresponding trade-off weights of performance, energy and other non-functional goals.

The work in [Edwards and Bencomo, 2018] proposes many operators or functions to evaluate the extent of non-functional goal satisfaction in self-adaptive systems. In our work and as described above, if the monitored metric m_i is less than or equal to its expected metric m^*_i, the corresponding non-functional goal is currently

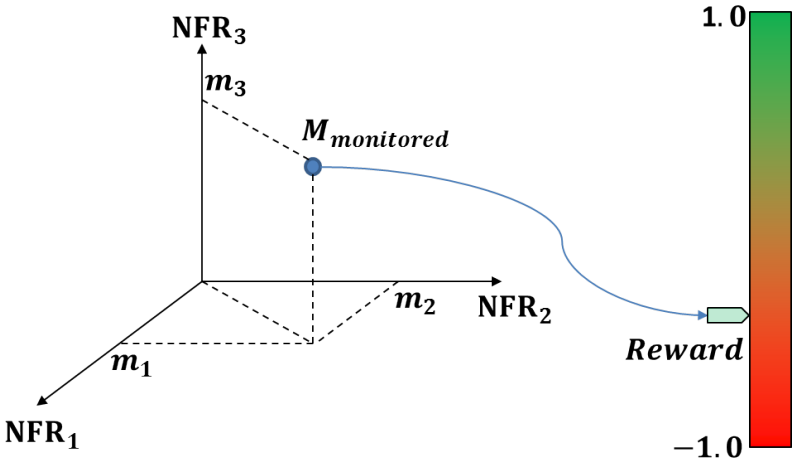


FIGURE 4.5: An example of a mapping between three monitored metrics (m_1, m_2, m_3) of three respective NFRs (NFR_1, NFR_2, NFR_3) and the obtained reward. The reward is clipped in a range of $[-1.0, 1.0]$, where the best reward is 1.0 and the worst reward is -1.0.

satisfied, and we choose to minimize the level of violation of all non-functional goals by defining the following reward function:

$$L(m_i) = \begin{cases} 0 & \text{if } m_i \leq m_i^*, \\ 1 - \frac{m_i}{m_i^*} & \text{otherwise} \end{cases} \tag{4.16}$$

$$R = \begin{cases} 1.0 & \text{if } \forall i, m_i \leq m_i^*, \\ \sum_{i=1}^{n_1+n_2+n_3} w_i \times L(m_i) & \text{otherwise,} \end{cases} \tag{4.17}$$

Where $L(m_i)$ is the level of violation of non-functional goal m_i and w_i is the corresponding importance weight. The reward can be interpreted as follows:

- $R = 1.0$ means that all non-functional goals are currently satisfied;
- $R < 0.0$ means at least one of non-functional goals is violated.

In our work, R is also limited at the negative part to -1.0 in order to ensure the stability of the obtained reward. Figure 4.5 illustrates an example of a mapping between three monitored metrics of three non-functional requirements and the obtained reward.

4.5 Deep Q-Learning based Approach

Similarly to the problem formulation based on Q-Learning, we also need to define a state space, an action space and a reward function for the Deep Q-Learning approach. However, the only difference in the formulation of the problem between these two approaches is on the formulation of the state space. The action space and the reward function are the same as those described above (see Sections 4.4.2 and 4.4.3).

4.5.1 State Space Formulation

The state is also defined by all the measured metrics $\{M_{perf}, M_{energy}, M_{other}\}$ and their expected metrics $\{M^*_{perf}, M^*_{energy}, M^*_{others}\}$. However, we do not binarize these metrics because the input of Q-Network can be continuous. In this case, we propose to integrate into the state space n_4 metrics representing the external context denoted as M_{ext} and their corresponding expected metrics M^*_{ext}. This integration is complex in case of a Q-Learning formulation because of the discretization. It is also important to note that the reward formulation is the same as in the case of Q-Learning, M_{ext} is not considered in this formulation. The expected metrics are used to normalize the measured metrics. The indicateur I_i of the metric m_i is now continuous and defined as $I_i = \frac{m_i}{m_i^*}$. Thus, a state s is an 1-D vector of $(n_1 + n_2 + n_3 + n_4)$ continuous elements: $s = [I_1, I_2, ..., I_{n_1+n_2+n_3+n_4}]$ and is the input of the Q-Network.

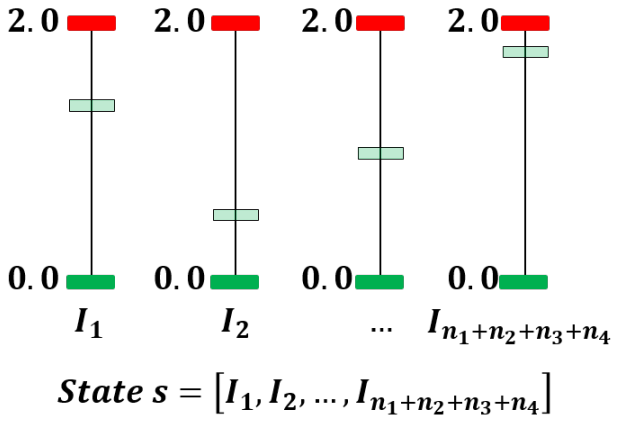


FIGURE 4.6: State formulation for the Deep Q-Learning approach. A state is an 1D vector of $(n_1 + n_2 + n_3 + n_4)$ continuous elements. The state space is therefore infinite.

The state space is therefore infinite. These continuous values give more insightful information about the variation than a simple binary indicator as defined above. As a good practice of clipping the input of neural networks, we also limit the range of I_i into $[0.0, 2.0]$:

- $I_i \in [0.0, 1.0]$ means that the non-functional goal is currently satisfied;
- $I_i \in (1.0, 2.0]$ means that the non-functional goal is violated.

Figure 4.6 illustrates the state formulation for the Deep Q-Learning approach.

4.6 Methodology Details

4.6.1 Learning Phase of Mission Manager based on Q-Learning

Figure 4.7 shows the details of the implementation at each step of the Q-Learning training. The robotic mission can be deployed in the simulation framework or in the real framework. At the time step t, the monitoring service (see Section 3.4) provides a set of monitored measurements $M_{monitored}$. The current state S_t is then calculated from these metrics by the block defining the state. The Q-Table takes the current state S_t as input and outputs the Q values from this state. The ϵ-greedy exploration and exploitation block (see Section 2.5.4) chooses the suitable action A_t. Then, the

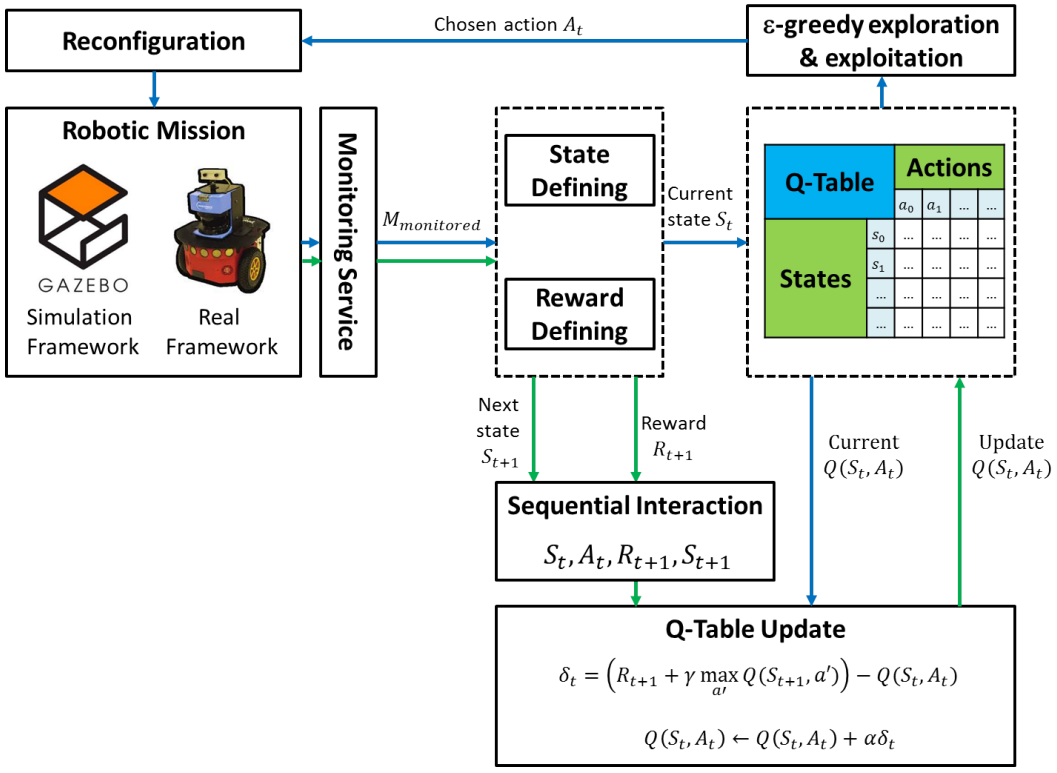


FIGURE 4.7: Q-Learning implementation details.

robotic mission is reconfigured by the chosen action A_t. Through this reconfiguration, at the next step $(t+1)$, the mission manager receives the reward R_{t+1} and the next state S_{t+1}. A complete sequential interaction $(S_t, A_t, R_{t+1}, S_{t+1})$ is known at this time. The Q-Table update block then calculates the temporal difference δ_t and updates the Q value $Q(S_t, A_t)$ in the Q-Table. Similarly, the Q-Table update process occurs at each time step until a terminal state is reached. The time between the beginning of adaptation and reaching a terminal state is called an episode. The terminal state can be real or virtual and defines the end of an episode. For example, the navigation mission consists on visiting in order a set of user-defined waypoints in the environment, so the real terminal state can be attained when reaching the final waypoint. Whereas, a virtual terminal state can be defined as the end of a fixed navigation time. The Q-Learning training process continues until the convergence of Q values is reached or until the end of a desired number of episodes. Indeed, the convergence of a Q-Learning training process is achieved when there is a fairly constant trend in the Q-values and reward.

4.6.2 Learning Phase of Mission Manager based on Deep Q-Learning

The training process of the Mission Manager based on Deep Q-Learning is described in Figure 4.8. The iterating process is similar to the one in the Q-Learning training. The difference is in the blocks of experience replay memory, target Q-Network and method updating the weight matrix θ of Q-Network.

Q-Network

As formulated above, the input of Q-Network is an 1-D vector of $(n_1 + n_2 + n_3 + n_4)$ continuous elements and the output is Q values of $\Pi_{i=1}^{l} n_{k_i}$ actions. The Multi-Layer

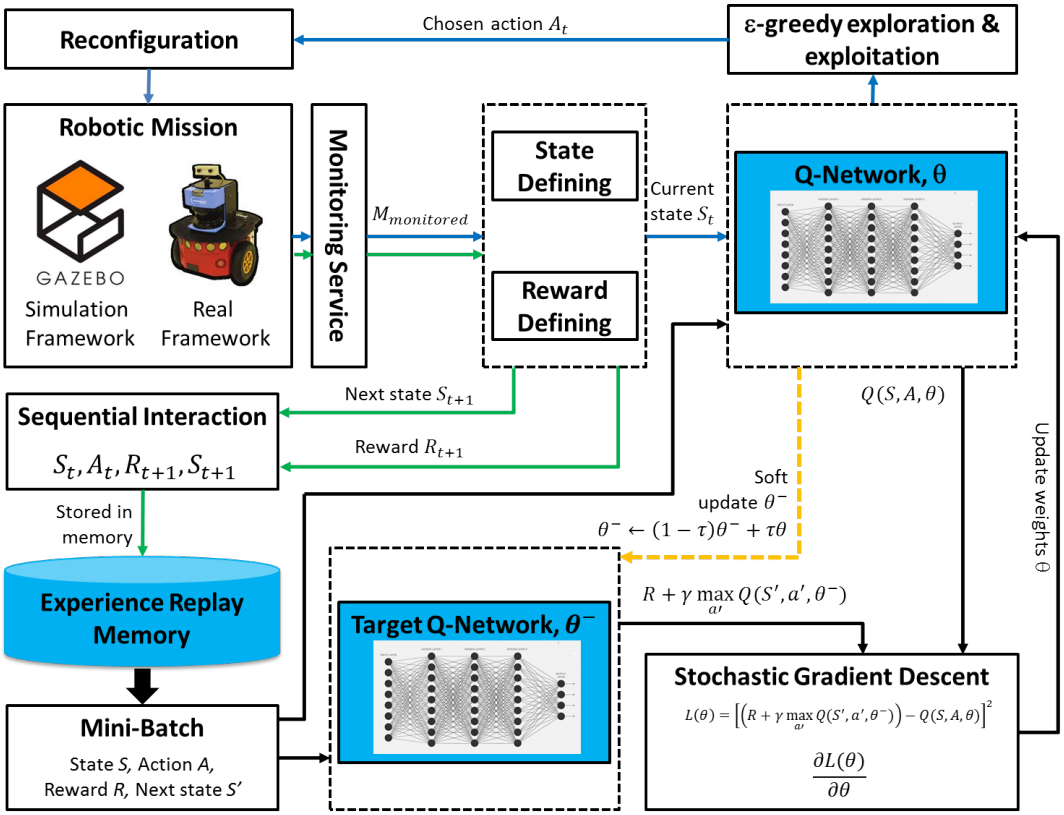


FIGURE 4.8: Deep Q-Learning implementation details.

Perceptron (MLP) with an input layer of $(n_1 + n_2 + n_3 + n_4)$ neurons, a number of hidden layers and an output layer of $\Pi_{i=1}^{l} n_{k_i}$ neurons is therefore chosen for representing the Q-Network. The structure of the Q-Network is shown in Figure 4.9. In this structure, the number of hidden layers and the number of neurons in each hidden layer are empirical and manually tuned parameters. In fact, the higher the number of hidden layers, the deeper the neural network is and the better the representation and extraction of information is. However, the increase in the number

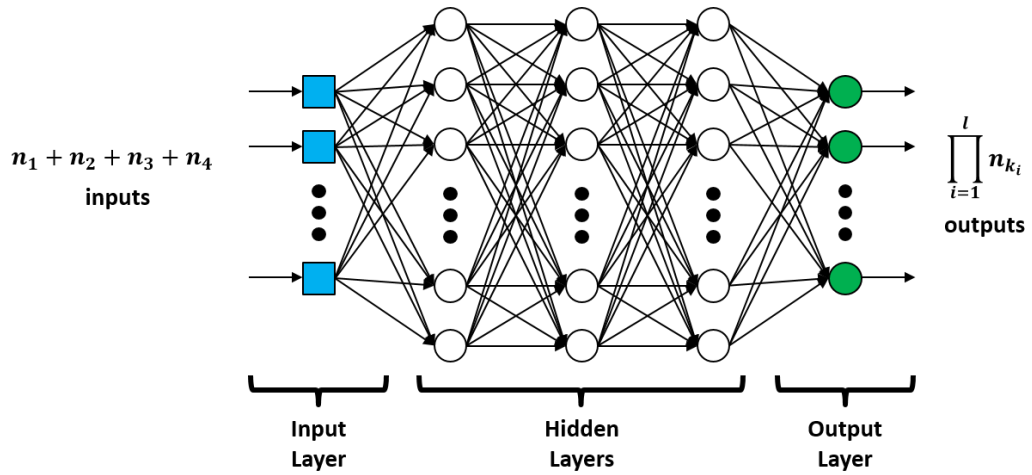


FIGURE 4.9: Q-Network structure: a Multi-Layer Perceptron (MLP) with an input layer of $(n_1 + n_2 + n_3 + n_4)$ neurons, a number of hidden layers and an output layer of $\Pi_{i=1}^{l} n_{k_i}$ neurons.

of hidden layers will lead to a greater consumption of computing resources such as CPU, memory and energy. This is very important if we consider deploying the management framework in an embedded platform.

Experience Replay Memory

Unlike Q-Learning where the state transition between time steps t and $(t+1)$ is used directly to update Q-Table, the state transition (current state, chosen action, received reward, next state) is stored in a finite memory called experience replay memory as illustrated in Figure 4.10. At time step $(t+1)$, a mini batch B over T experiences is extracted from this memory to train the Q-Network. Similarity to the number of hidden layers and the number of neurons in the Q network, the choice of the batch size B and the number of stored experiments T must be appropriate to be deployed on an embedded platform.

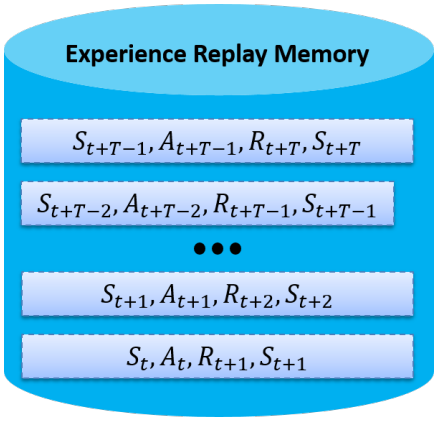


FIGURE 4.10: Experience replay memory of T transitions.

Target Q-Network

Target Q-Network is another idea proposed by [Mnih et al., 2015] to keep the Q-Network training stable. The weight matrix θ^- of target Q-Network is updated gradually from Q-Network. In the litterature of Deep Q-Learning, there are two methods to update target Q-Network from Q-Network:

- **Hard update** as originally proposed by [Mnih et al., 2015] means after a number of learning steps, θ^- is assigned to θ, or $\theta^- \leftarrow \theta$;
- **Soft update** means θ^- is gradually updated from θ with a very small coefficient τ (as a moving average of θ) [Lillicrap et al., 2015]:

$$\theta^- \leftarrow (1-\tau)\theta^- + \tau\theta \tag{4.18}$$

The work in [Lillicrap et al., 2015] indicates that with soft update, the target network is constrained to change slowly and therefore greatly improves the stability of learning. Our work implements also the soft update for the target Q-Network.

Stochastic Gradient Descent

The Q-Network is now updated by an optimization technique named Stochastic Gradient Descent (SGD). The loss function $\mathbb{L}(\theta)$ is calculated as Mean Squared Error

(MSE) between target Q-value and current Q-value of all transitions in mini-batch. The gradient of loss is then calculated and the Q-Network is updated by an optimizer known in deep learning such as Adam, AdaGrad, RMSProp, etc. Among these optimizers, Adam is the most commonly used in the state of the art deep learning deployment.

4.6.3 Planning Phase of Mission Manager

After the training process is finished, a Q-Table or Q-Network mapping between state space and action space is found depending on whether it is a Q-Learning or a Deep Q-Learning implementation.. Then, the learned model is used for the planning or control phase of the Mission Manager in different operational scenarios (Figure 4.11). This process is also described in Figures 4.7 and 4.8 with $\epsilon = 0$ in the ϵ-greedy exploration and exploitation block and without update block. We distinguish three cases:

- **Learning and planning in the simulation framework**. In this case, both learning and planning phases are deployed in the simulation framework. The objective is therefore to validate our proposed methodology in simulation before launching the methodology in the real framework;
- **Learning and planning in the real framework**. After the methodology is successfully validated in simulation, we deploy both the learning and planning phases in the real framework. It means that the learned model in simulation is not reused for the real framework;
- **Learning in the simulation framework and planning in the real framework**. This case is our target where the methodology is successfully validated in simulation and the learned model is reused for planning in the real framework. However, the shift between simulation and real framework is an inherent fact. The online learning in the planning phase is therefore important in this case.

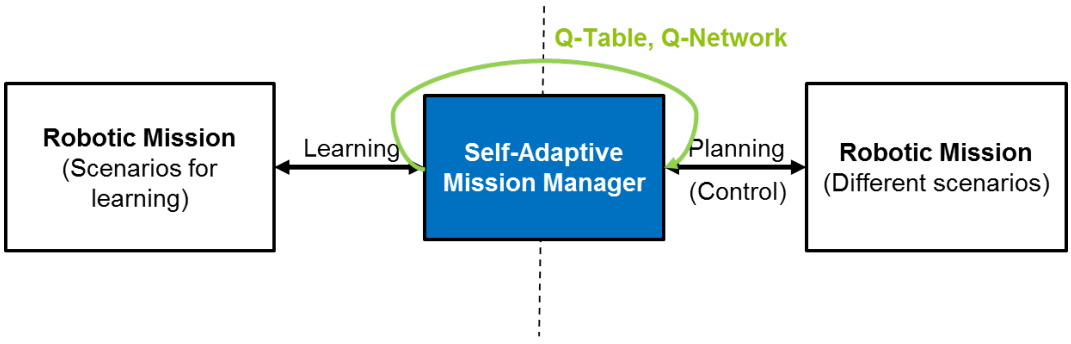


FIGURE 4.11: Learning and planning phase of a self-adaptive mission manager.

4.6.4 Online Learning

If in the planning phase, we also want to update the adaptation model (update the Q values in Q-Table or update the weight matrix θ in Q-Network), we call it an **online learning**. Online learning is necessary because in the learning phase, we cannot

anticipate all dynamic operational conditions. The adaptation model is therefore updated to deal with more unexpected and unforeseen scenarios. Indeed, the deployment of online learning in the planning phase is known as **hybrid planning** which we discussed in the planning strategy of self-adaptive systems in Section 2.2.3.

4.6.5 Transfer Learning

Transfer learning is a popular technique in deep learning, where the model learned for one task is reused as a starting point for learning another similar task. A learning task in our problem is to learn an adaptation policy for a desired set of non-functional requirements. The learning process of a new set of NFRs can use the adaptation policy of an old set of NFRs as a starting point.

4.7 Q-Learning based Mission Manager: A Case Study

In this section, we use a navigation mission presented in Chapter 3 as a case study in order to validate our proposed methodology of a Q-Learning based Self-Adaptive Mission Manager. The implementation and validation are realized in the simulation framework.

4.7.1 Problem Statement

The mission's functional objective is to visit in order a set of user-defined waypoints in the robot working environment whose global static map is given a priori. The non-functional goals applied to this mission $G_{Mission}$ are defined as follows:

- $G_{perf} = \{V_{mean\_min}\}$, where V_{mean_min} is the minimum mean velocity that the robot should achieve;
- $G_{energy} = \{P_{s\&a\_mean\_max}, P_{c\_mean\_max}\}$, where $P_{s\&a\_mean\_max}$ and $P_{c\_mean\_max}$ are the respecting maximum mean power consumption for sensing&acting and computing parts;
- $G_{others} = \{D_{obs\_min}\}$, where D_{obs_min} defines the minimum distance to obstacles during navigation.

The configuration knobs of the navigation mission are defined by a set of two key parameters: $K_{Mission} = \{v_{max}, f_{control}\}$, where v_{max} is the max robot velocity and $f_{control}$ is the control loop frequency of the path following process. The impact of these two parameters on the non-functional properties of the navigation mission has been also analyzed in Section 3.6.

4.7.2 Problem Formulation

State Space Formulation

In this problem, there are four non-functional objectives applied to the mission. We monitor directly these four objectives: *mean robot velocity*, *mean sensing and acting power consumption*, *mean computing power consumption* over the passed trajectory and *minimum distance to obstacles in front of robot*. As presented in Section 4.4.1, for representing the current guarantee level of non-functional goals, we define four run-time metrics:

- mp_1 = *inverse of mean robot velocity*;
- me_1 = *mean sensing and acting power consumption*;
- me_2 = *mean computing power consumption*;
- mo_1 = *obstacle collision risk (inverse of minimum distance to obstacles)*.

And their expected metrics $\{mp_1^*, me_1^*, me_2^*, mo_1^*\}$ are directly derived from $G_{Mission}$. The smaller these metrics are, the better their non-functional goals are guaranteed. Thus, the navigation mission manager has $2^{1+2+1} = 16$ normal states in its state space. Furthermore, for defining the end of a navigation mission, we add an extended binary indicator (1 - end of mission, 0 - during mission) to the 16 normal states. This indicator is then used to define a terminal state in the Q-Learning problem (see the definition of decision epoch and episode in Section 2.5.3). Finally, the state space of the navigation mission manager possesses $2^{1+2+1+1} = 32$ states.

Action Space Formulation

We choose in this study an action space including 11 configuration knobs of $K_{Mission} = \{v_{max} m/s, f_{control} Hz\}$ as detailed in Table 4.1. This choice leverages the knowledge that when the robot moves at high speed, the frequency of detecting and avoiding obstacles should be higher [Cano et al., 2016; Ho et al., 2018].

TABLE 4.1: Action space: configuration knobs $K_{Mission} = \{v_{max}, f_{control}\}$.

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
0.5, 5	0.5, 10	0.65, 5	0.65, 10	0.65, 15	0.8, 10	0.8, 15	0.8, 20	1.0, 20	1.0, 25	1.0, 30

Reward Function Formulation

In this case study, we consider all non-functional goals as constraints and the reward function is defined as a trade-off between the mission's functional progress and the constraints penalty. In fact, this formulation is only taken into account for the particular case study of the navigation mission where the functional progress can be defined as the number of waypoints visited over the number of waypoints required. This formulation is then replaced by the reward function formulation presented in Section 4.4.3 to guarantee the methodology generalization.

The mission's functional progress $P \geq 0$ is calculated as the ratio of number of visited waypoints and total required waypoints:

$$P = \frac{number\ of\ visited\ waypoints}{total\ required\ waypoints} \tag{4.19}$$

The constraints penalty $C < 0$ evaluates the level of non-functional goal guarantees in case of violation: L_{perf_1}, L_{energy_1}, L_{energy_2} and L_{others_1}. Each level L is calculated based on the corresponding metrics m and m^* as follows:

$$L = \begin{cases} 0 & \text{if goal satisfied } (m \leq m^*), \\ 1.0 - \frac{m}{m^*} & \text{otherwise} \end{cases} \tag{4.20}$$

And the global constraint penalty C is then computed as:

$$C = W_{p\_1} \times L_{perf\_1} + W_{e\_1} \times L_{energy\_1} + W_{e\_2} \times L_{energy\_2} + W_{o\_1} \times L_{others\_1} \tag{4.21}$$

Where W_{p_1}, W_{e_1}, W_{e_2} and W_{o_1} are corresponding trade-off weights of four non-functional goals.

Finally, the reward R is positive as mission's functional progress if all non-functional goals are satisfied, and negative as constraints penalty if one of non-functional goals is violated:

$$R = \begin{cases} W_P \times P & \text{if all constraints satisfied,} \\ W_C \times C & \text{otherwise} \end{cases} \tag{4.22}$$

Where W_P and W_C are the trade-off weights of mission progress and constraints penalty. Moreover, for the stability of the reward feedback, the reward is clipped in the range of $[-1.0, 1.0]$. An action that guarantees all non-functional goals will be reinforced (positive reward).

4.7.3 Implementation

Our objective is to demonstrate the results obtained by the offline learning phase in Gazebo-based simulations. In this study, we implement a classic Q-Learning algorithm that balances the exploration and exploitation phase by using the *ϵ-greedy* algorithm [Sutton and Barto, 2014] in order to find a Q-Table, and then the mission manager uses this look-up table to map an adaptive action corresponding to the actual state. The exploration probability ϵ is determined by max probability ϵ_{max}, min probability ϵ_{min} and decay factor ϵ_{decay} as follows:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \times e^{-\epsilon_{decay} * learning\_steps} \tag{4.23}$$

The parameters that we consider are:

- **Q-Learning parameters** with discount rate (γ), learning rate (α), exploration-exploitation parameters ($\epsilon_{max}, \epsilon_{min}, \epsilon_{decay}$) and time step ($\tau$);

TABLE 4.2: Q-Learning parameters

τ	γ	α	ϵ_{max}	ϵ_{min}	ϵ_{decay}
5s	0.9	$\frac{1}{N(s,a)}$	1.0	0.01	0.0001

- **Reward function parameters** with trade-off weights between non-functional goals ($W_{p\_1}, W_{e\_1}, W_{e\_2}, W_{o\_1}$), trade-off weights between mission progress and constraint penalty (W_P, W_C).

TABLE 4.3: Reward function parameters

W_{p_1}	W_{e_1}	W_{e_2}	W_{o_1}	W_P	W_C
2.5	2.5	2.5	2.5	1	1

The parameter values that we have chosen empirically for this implementation are detailed in Table 4.2 and 4.3. The learning rate α is specially chosen for ensuring the convergence of the learning algorithm as mentioned in the work of [Watkins, 1992] with $N(s,a)$ the number of times the pair state-action (s,a) has been visited. The time step τ is chosen as $5s$ in this case study and this choice is guided by considering the latency that a new configuration is taken into account and modifies the mission's behavior. Other Q-Learning parameters are chosen from a commonly used range that gives the relatively optimal performance in the state of the art Q-Learning

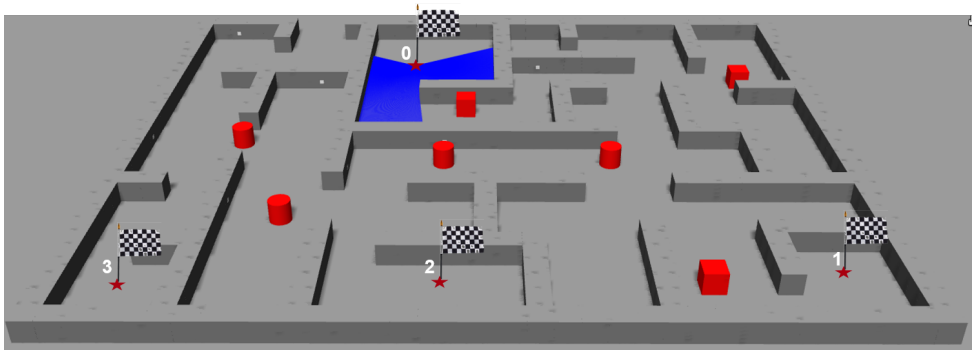


FIGURE 4.12: Example of a simulation environment in Gazebo with 7 dynamic obstacles: single box or cylinder in red models dynamic obstacles, blue area represents laser scan. The flags represent the examples of navigation waypoints.

problem. These parameters have some impact on the convergence rate as well as the time of the learning process, but our experiments indicate that they do not affect considerably the final mapping policy.

4.7.4 Learning Phase Deployment

The simulation environment in Gazebo is shown in Figure 4.12. The dynamic obstacles can be randomly generated in the environment (red boxes and cylinders in Figure 4.12). A learning episode of Q-Learning is a completed mission. The desired non-functional goals of the navigation mission given in Table 4.4 are the averaged results of mission deployments with a set of static configurations in a simulated environment without any dynamic obstacles.

TABLE 4.4: Non-functional goals for navigation missions

V_{mean_min}	$P_{s\&a\_mean\_max}$	$P_{c\_mean\_max}$	D_{obs_min}
0.56m/s	17.05W	2.75W	0.7m

We deployed 400 learning episodes in simulation. The first 200 episodes are to visit a set of waypoints with the total optimal trajectory length of 204m and the last 200 ones for a different set of waypoints with the total path length of 166m. Moreover, a different set of dynamic obstacles (max 8 obstacles) is randomly generated at each 10 episodes. Hence, 40 different environment scenarios are randomly generated during the learning phase. This enhances the generalization of the learned policy.

Figure 4.13 depicts the total reward obtained over 400 learning episodes. Despite of many fluctuations due to the variable complexity of environments and the efficiency of existing navigation algorithms, the trend line also indicates the growth of the total reward during the learning phase. This figure shows that the policy mapping between states and actions has evolved aiming to guarantee the non-functional goals of the navigation mission. The changing of 4 desired non-functional goals during learning phase is described in Figure 4.14. The increasing trend of velocity and distance to obstacles and the decreasing trend of computing power can be easily realized, while the trend of sensing&acting power is correlated to the velocity. In fact, the wheel motor power consumption estimation model that we use is a quadratic equation of velocity and if the velocity is increased, this power consumption is also

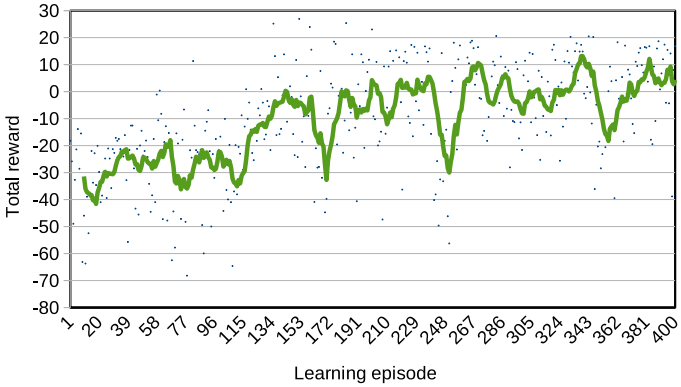
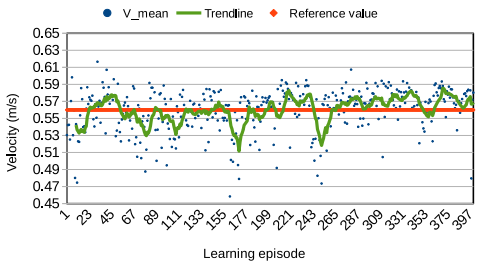
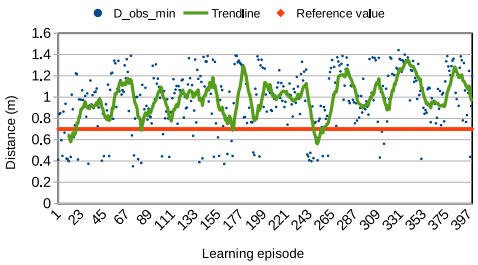


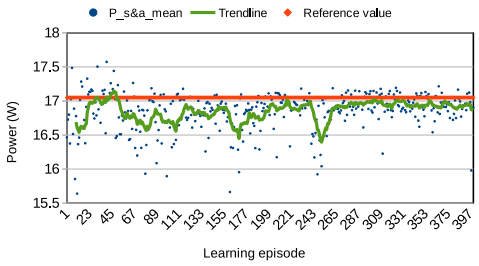
FIGURE 4.13: Total reward of the learning phase over 400 episodes of Q-Learning based Navigation Mission Manager with the trend line in green. The growth of total reward during learning indicates the evolution of the adaptation policy.



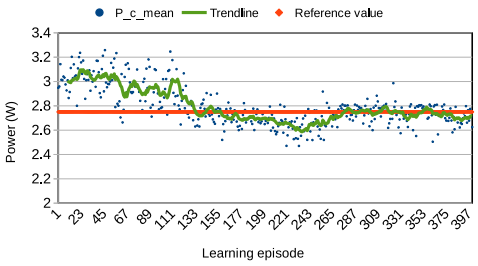
(A) Mean robot velocity.



(B) Minimum distance to obstacles.



(C) Mean sensing&acting power consumption.



(D) Mean computing power consumption.

FIGURE 4.14: Non-functional requirements over 400 learning episodes of Q-Learning based Navigation Mission Manager.

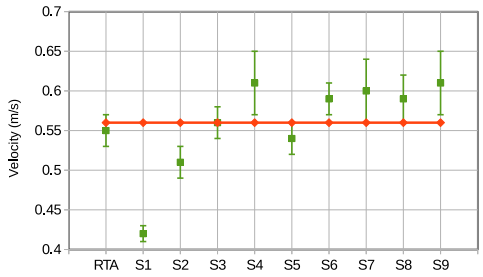
increased. So, even if the objective of increasing velocity is conflicting with the objective of decreasing sensing&acting power consumption, our Q-Learning based decision-making has managed to balance both objectives.

4.7.5 Validation of Resulting Adaptation Policy

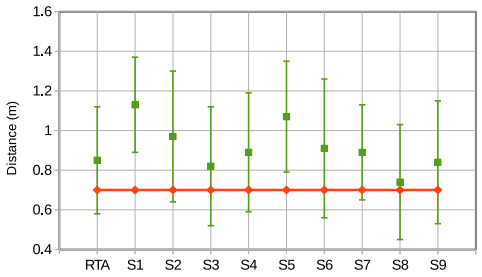
We intend to compare our run-time adaptation (RTA) based on the mission manager with 9 static configuration cases (chosen before the mission deployment and fixed during mission execution as indicated in Table 4.5) in terms of non-functional goal guarantees. In fact, static configurations are usually used in state of the art mobile robotic mission deployment. We generated randomly 24 simulated environments

TABLE 4.5: 9 static configurations $K_{Mission} = \{v_{max}, f_{control}\}$ chosen for validation.

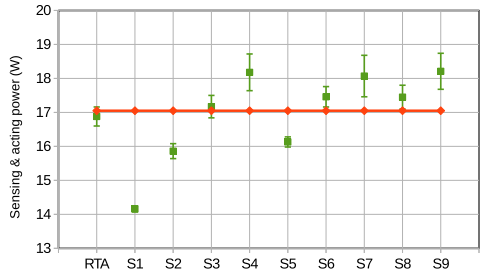
S1	S2	S3	S4	S5	S6	S7	S8	S9
0.5, 5	0.65, 5	0.8, 10	1.0, 20	0.65, 10	0.8, 15	1.0, 25	0.8, 20	1.0, 30



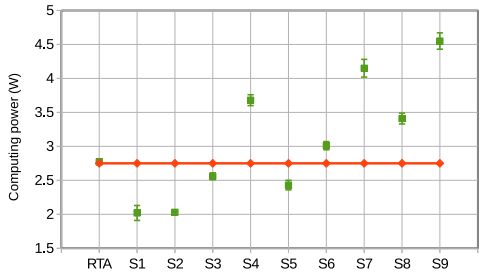
(A) Mean robot velocity.



(B) Mean minimum distance to obstacles.



(C) Mean sensing&acting power consumption.



(D) Mean computing power consumption.

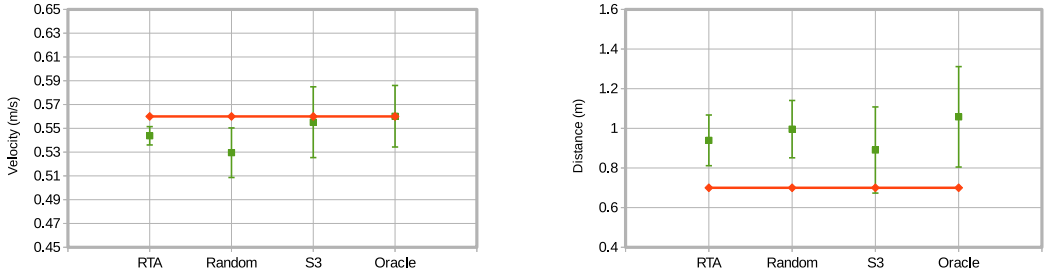
FIGURE 4.15: First validation of resulting adaptation policy. The red points indicate the same goals for both dynamic and static cases. Our run-time adaptation (RTA) trades off all non-functional goals and give the most feasible solution for guaranteeing the goals.

with max 8 dynamic obstacles. We run our RTA and the 9 static configurations of the navigation mission on a map with a total path length of $194m$ on these 24 environment scenarios. The same non-functional goals in Table 4.4 are also applied to these missions. For each dynamic or static case, the results, averaged over 24 episodes, and their standard deviation are depicted in Figure 4.15.

We can see that the run-time navigation mission manager achieves a good balance between the four non-functional goals, while the static configurations can just guarantee some among these goals but violate significantly the others. Furthermore, our validation results also indicate that with RTA in 2 over 24 validation episodes we perfectly guarantee all the non-functional goals ($V_{mean} \geq 0.56m/s$, $P_{s\&a\_mean} \leq 17.05W$, $P_{c\_mean} \leq 2.75W$, $D_{obs} \geq 0.7m$), while none of the validation episodes of the 9 static configurations gives the same performance.

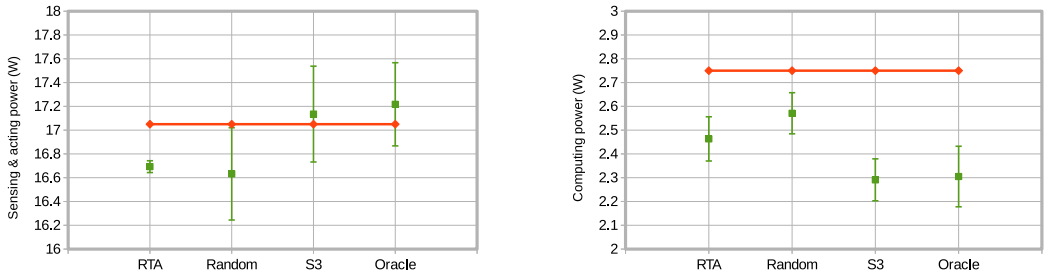
After the comparison with static configurations, we deploy a second validation for comparing our RTA with three other policies:

- A **random policy** where the configuration is randomly chosen in the action space (Table 4.1) during mission with the same adaptation period of $5s$ as RTA;
- The **best static case** $S3 = \{0.8m/s, 10Hz\}$;



(A) Mean robot velocity.

(B) Mean minimum distance to obstacles.



(C) Mean sensing&acting power consumption.

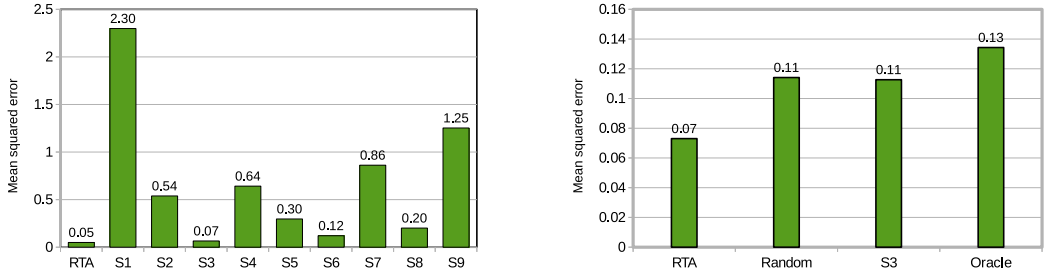
(D) Mean computing power consumption.

FIGURE 4.16: Second validation of resulting adaptation policy. The red points indicate the same goals for all the policies.

- An **oracle** where all dynamic obstacles are known a priori (the path planners have taken into account these obstacles) and the mission is configured by the best static case *S*3.

The results are averaged over 10 deployment episodes in the same environmental context with dynamic obstacles as shown in Figure 4.16.

We can realize that the best static case and the oracle can give good results for the goals of velocity and computing power, but they cannot deal with the natural conflict between the velocity and the sensing&acting power objectives as analyzed above. For the random policy, the adaptation behavior is not controlled and the objective function is not defined, so the results are also random. Our RTA is based on an objective function determined by a trade-off between all non-functional goals. It may not give the best results for all objectives, but it can handle multi-objective



(A) First validation.

(B) Second validation.

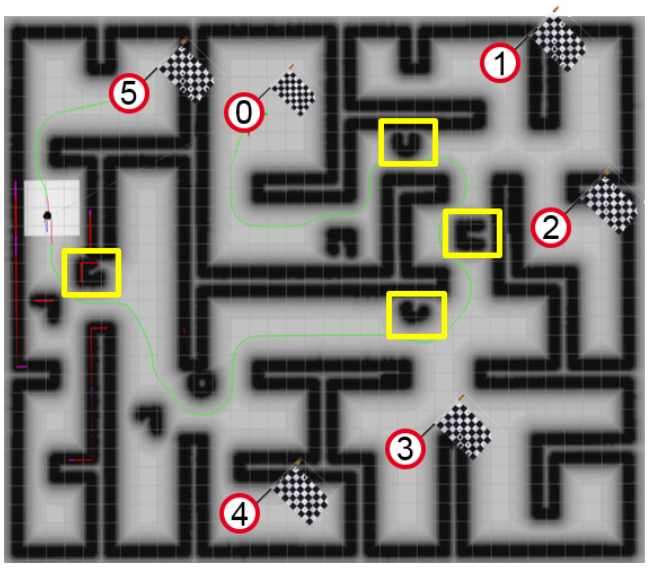
FIGURE 4.17: Mean squared error (Note: the environmental context between two validations is different).

problem where the conflicts between objectives can occur, and its adaptation behavior evolves in a controlled manner.

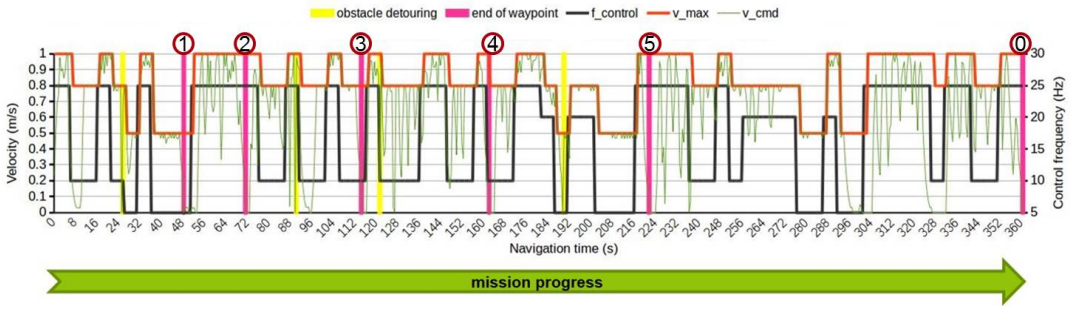
The mean squared error (MSE) is also calculated for evaluating the ability of meeting closely the non-functional goals in each dynamic or static case as depicted in Figure 4.17a:

$$MSE = \frac{1}{4}\sum_{j=1}^{4}(\frac{1}{24}\sum_{i=1}^{24}(g_i^j - g^j)^2), \tag{4.24}$$

where g^j is the required goal (V_{mean_mean}, $P_{s\&a\_mean\_max}$, $P_{c\_mean\_max}$, D_{obs_min}) and g_i^j is the respecting result of the validation episode i over 24. The results indicate that our RTA gives the lowest error compared to the static configurations. Similar to the first validation, the MSE results for the second one have also been calculated as shown in Figure 4.17b and the lowest error for our RTA is also found.



(A) Navigation mission of visiting 6 waypoints.



(B) Mission progress and run-time adaptation.

FIGURE 4.18: Example of run-time adaptation during a navigation mission of visiting a set of 6 waypoints. The max robot velocity v_{max} and control frequency $f_{control}$ are dynamically reconfigured by our proposed mission manager whilst targeting the four required non-functional goals. The v_{cmd} is the actual velocity sent by the navigation mission in order to command directly the mobile base.

4.7.6 Adaptation Interpretation

Figure 4.18 describes an example of applying the resulting adaptation policy to a navigation mission. The max robot velocity (v_{max}) and the control frequency ($f_{control}$) are dynamically reconfigured by our mission manager in order to reach the four required non-functional goals. The real velocity (v_{cmd}) sent to the mobile base is given by the navigation mission. There are many dynamic behaviors during the mission progress. The run-time adaptation (RTA) takes into account the consequences of these behaviors and gives the adaptive decisions in order to meet the non-functional goals in a feasible situation.

Figure 4.19 depicts the run-time adaptation policy decisions and environmental changes in a navigation portion between two waypoints (0 and 1). On the robot's trajectory, the state of performance and energy consumption is actively monitored and the corresponding configuration is chosen based on the found Q-Table. For example, the state 7 corresponds to the violation of the velocity goal, so the RTA takes the decision at time 17s to increase the robot max velocity from 0.8 to 1.0m/s and the control loop frequency is also increased to 25Hz for guaranteeing the safety criteria at high speed. At the transition to state 9 corresponding to the violation of the sensing&acting and computing power goal, the RTA takes the decision at time 27s to decrease the robot max velocity and the control loop frequency so that the energy consumption is guaranteed.

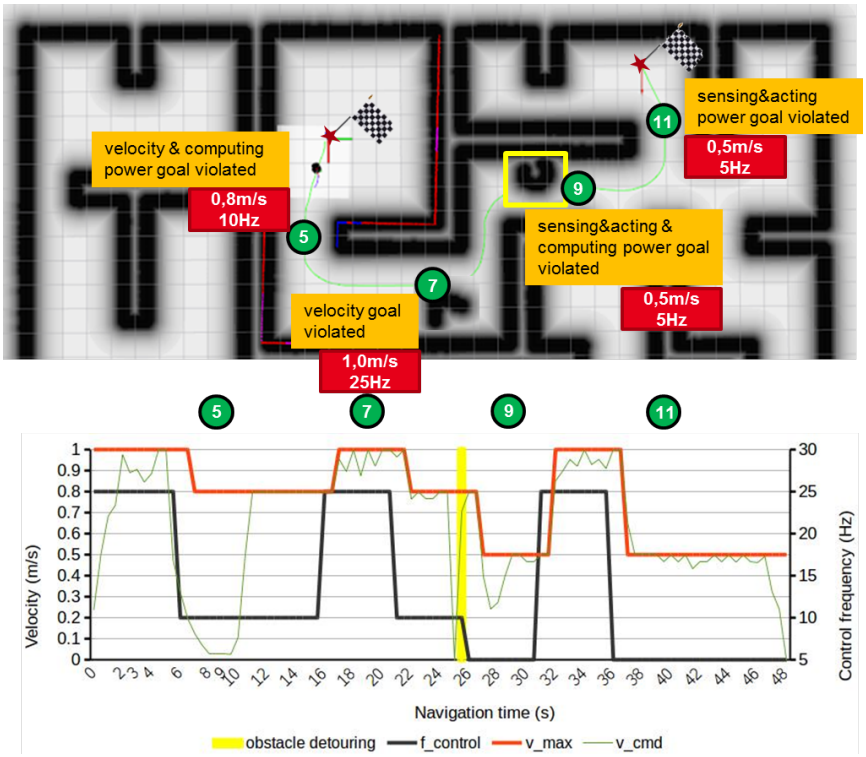


FIGURE 4.19: Adaptation interpretation while the robot visits the first waypoint. The state of performance and energy consumption is actively monitored on the robot's trajectory, states 5, 7, 9 and 11 in this case, and the corresponding configuration is chosen based on the found Q-Table.

4.7.7 Discussion

The proposed method shows that the mission manager is able to be aware of changes in operational circumstances such as dynamic obstacles, and then to choose the suitable mission-specific parameter configuration. All decisions are made dynamically and sequentially, so that a sequence of actions chosen during operation will contribute to meet the final mission non-functional goals.

The Q-Learning approach is our first study of the application of reinforcement learning into our run-time adaptation framework. However, we complain about the possibility of extending the state space as well as the possibility of generalization. For example, if we have 10 input metrics and with a simple binarization, we would have $2^{10} = 1024$ states. If we want to discretize each metric into n bins ($n > 2$) to keep more information, the state space would be now composed of n^{10} states (ex. $n = 3, 3^{10} = 59049$ states). Addressing this problem is really a challenge for Q-Learning and we do not want to limit the generalization of our methodology because of this problem. Thus, for the rest of our thesis, we focus only on the Deep Q-Learning approach.

4.8 Validation of Deep Q-Learning based Mission Manager

In this section, we implement and validate the Deep Q-Learning based run-time adaptation for two robotic missions such as a navigation mission and a video server mission in the simulation framework.

4.8.1 Problem Statement

Navigation Mission

The problem statement of the navigation mission is the same as the case of Q-Learning described above. The functional objective is to visit a set of user-defined waypoints in the robot working environment whose global static map is given a priori. A set of four non-functional goals $G_{Nav\_Mission}$ is also applied to this mission:

$$G_{Nav\_Mission} = \{V_{mean\_min}, P_{s\&a\_mean\_max}, P_{c\_mean\_max}, D_{obs\_min}\} \tag{4.25}$$

Where,

- V_{mean_min} is the minimum mean velocity;
- $P_{s\&a\_mean\_max}$ and $P_{c\_mean\_max}$ are the respective maximum mean power consumption for sensing&acting and computing parts;
- D_{obs_min} defines the minimum distance to obstacles during navigation.

The configuration knobs of the navigation mission are also defined by the set of two key parameters: $K_{Nav\_Mission} = \{v_{max}, f_{control}\}$, where v_{max} is the max robot velocity and $f_{control}$ is the control loop frequency of the path following process during navigation.

Video Server Mission

The functional objective of the video server mission is to provide the visual information in the form of encoded depthcloud of the robot working environment that can

be visualized on a remote web browser or used for post-processing. A set of three non-functional goals $G_{Ser\_Mission}$ can be applied to this mission:

$$G_{Ser\_Mission} = \{RES_{mean\_min}, P_{c\_mean\_max}, FPS_{mean\_min}\} \tag{4.26}$$

Where,

- RES_{mean_min} is the minimum mean target resolution ($2 \times crop\_size$);
- $P_{c\_mean\_max}$ is the maximum mean computing power consumption;
- FPS_{mean_min} is the minimum mean frame rate of the image stream.

The configuration knobs of the video server mission are defined by a key parameter known as $crop\_size$ (see Section 3.6.2): $K_{Ser\_Mission} = \{crop\_size\}$.

4.8.2 Problem Formulation

State Space Formulation

The state space formulation methodology presented in Section 4.5.1 is applied to both missions. We present in details the set of metrics for each robotic mission.

Navigation Mission. In this case, we define 12 run-time metrics representing the state space. Three metrics mp_1, me_1 and me_2 are the same as defined in the Q-Learning approach (see Section 4.7.2), mp_1 = *inverse of mean robot velocity*, me_1 = *mean sensing and acting power consumption*, and me_2 = *mean computing power consumption*. And we add 9 metrics as *inverse of distances measured by the laser sensor* for having a wider view of obstacles in front of the robot. They represent also the external context of the navigation mission. With these 12 metrics, if we use Q-Learning, there are $2^{12} = 4096$ states in the state space. Hence, we see once again the infeasibility of Q-Learning in this case.

Video Server Mission. In this mission, there are three non-functional goals and we observe directly these goals: mean target resolution, mean computing power consumption and mean frame rate. The external context such as visualization requests is not monitored because of its complexity. Thus, the state space of this mission is defined by three run-time metrics representing the internal context of the mission: mp_1 = *inverse of mean target resolution*, me_2 = *mean computing power consumption*, and mo_1 = *inverse of mean frame rate*.

Action Space Formulation

The action space of the Deep Q-Learning based Navigation Mission Manager and Server Mission Manager is shown in Tables 4.6 and 4.7. For the video server mission, it gets the color and depth images from a Kinect sensor with an image resolution of 640×480, we propose therefore a range for $crop\_size$ from 100 to 650 with an interval of 50. The choice of a larger range of $crop\_size$ or a finer interval makes the action space of the mission manager bigger, but does not have any impact on the mission's functionality.

TABLE 4.6: Action space of Deep Q-Learning based Navigation Mission Manager: configuration knobs $K_{Nav\_Mission} = \{v_{max}, f_{control}\}$.

K1	K2	K3	K4	K5	K6	K7	K8	K9
0.5, 5	0.65, 5	0.65, 10	0.8, 10	0.8, 15	0.8, 20	1.0, 20	1.0, 25	1.0, 30

TABLE 4.7: Action space of Deep Q-Learning based Server Mission Manager: configuration knobs $K_{Ser\_Mission} = \{crop\_size\}$.

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12
100	150	200	250	300	350	400	450	500	550	600	650

Reward Function Formulation

In the reward function formulation, we use the equations 4.16 and 4.17 for all missions. In the case of the navigation mission, we use 9 distance metrics to define the state space, the minimum value is used to calculate the reward by comparing with the corresponding goal D_{obs_min}. We aim to trade-off all non-functional goals with the same importance weights. Thus, these weights are set to 1.0.

4.8.3 Implementation

Q-Network

The Q-Network based on Multi-Layer Perceptron structure is shown in Figure 4.9. In this network, the input and output layers are different for each mission and depend on the number of input metrics and the number of configuration knobs, and we implement the same hidden layers for all missions. We implement 3 hidden layers as fully connected layers, each hidden layer has 64 neurons.

Navigation Mission. Q-Network of the navigation mission has 12 inputs and 9 outputs. So, the weight matrix θ_{nav} has

$$(12 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 9 + 9) = 9737$$

weights.

Video Server Mission: Q-Network of the video server mission has 3 inputs and 12 outputs. So, the weight matrix θ_{ser} has

$$(3 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 12 + 12) = 9356$$

weights.

In fact, these weights are floating-point numbers and typically occupy 8 *bytes* in terms of memory footprint. Thus, a Q-Network of 9737 weights of the navigation mission will occupy approximately 9737×8 *bytes* $= 76$ *kBytes*, and a Q-Network of the video server mission will occupy approximately 9356×8 *bytes* $= 73$ *kBytes*.

Target Q-Network

The structure of target Q-Networks is the same as the one of Q-Networks. We implement the soft update as indicated in Equation 4.18 with $\tau = 0.001$.

Experience Replay Memory

We implement an experience replay memory for each mission manager with maximum size $max\_size = 4096$ and batch size $batch\_size = 32$.

Stochastic Gradient Descent

In the literature, the Adam optimizer can be considered as the most used optimizer for the gradient descent based optimization. So, we implement also this optimizer for our Q-Networks. We choose also a learning rate of 0.0001 for this optimizer.

TABLE 4.8: Parameters of Q-Network for each mission manager.

Structure of Q-Network	Hidden layers: 3 Neurons in each hidden layer: 64
Soft update of Target Q-Network	$\tau = 0.001$
Experience Replay Memory	Max size: 4096 Batch size: 32
Stochastic Gradient Descent	Optimizer: Adam Learning rate: 0.0001

Table 4.8 summarizes the aforementioned parameters. It is important to note that the choice of these parameters is orthogonal to the effectiveness of our methodology. We consider to choose the values in the range that have been proposed in the state of the art training Deep Q-Networks and provided the best results [Mnih et al., 2015; Lillicrap et al., 2015] to prove the performance of our run-time adaptation proposal. Tuning these parameters and choosing the best may be reserved for the future work.

4.8.4 Learning Phase Deployment

For a generalization purpose, we define an episode of any mission as a function of time. This means that each episode occurs during a specific period of time. In an episode, the adaptation is performed at each time step. The choice of the adaptation time step is dependent on the latency of reconfiguration and the fact that a new configuration will be taken into account to generate the new behavior of the robotic mission. We choose an episode of 1 minute for both missions. For the navigation mission, the adaptation time step is set to 2 seconds, so there are $60/2 = 30$ time steps in an episode. For the video server mission, the adaptation time step is set to 1 second, so there are $60/1 = 60$ time steps in an episode.

We deploy these missions in the same time and the learning phase is executed for each mission. The learning phase is deployed in 1500 episodes, equivalently 25 hours. During the simulation, we randomly generate obstacles in the robot simulated environment. The desired non-functional goals of the navigation mission and the video server mission are given in Table 4.9. The ϵ-greedy algorithm (described in Equation 4.23) is also implemented for balancing between exploration and exploitation. Its parameters are given in Table 4.10.

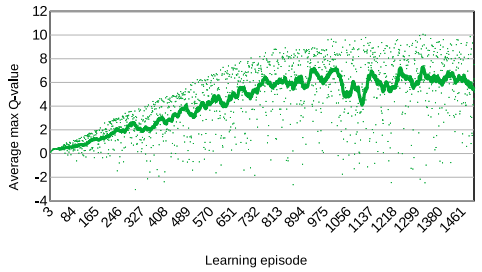
TABLE 4.9: Non-functional goals for Navigation Mission and Server Mission.

NFRs	Navigation Mission	Server Mission
G_{perf_1}	$V_{mean\_min} = 0.60m/s$	$RES_{mean\_min} = 800.0pixels$
G_{energy_1}	$P_{s\&a\_mean\_max} = 18.5W$	
G_{energy_2}	$P_{c\_mean\_max} = 4.0W$	$P_{c\_mean\_max} = 8.0W$
G_{others_1}	$D_{obs\_min} = 0.7m$	$FPS_{mean\_min} = 25.0Hz$

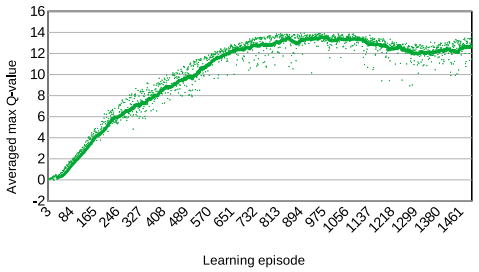
The learning results of the two missions are shown in Figure 4.20. The average max Q-value and the average reward per time step are calculated for each learning

TABLE 4.10: Parameters for exploration and exploitation in the learning phase of DQN-based Mission Managers.

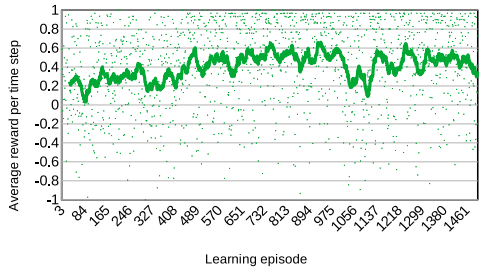
ϵ_{max}	ϵ_{min}	ϵ_{decay}
1.0	0.01	0.0001



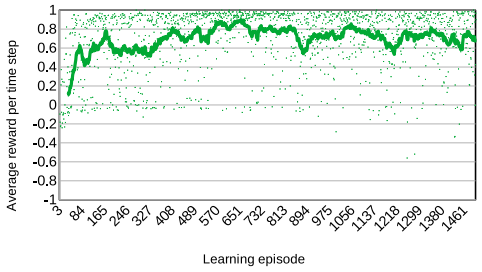
(A) Average max Q-value of Navigation Mission.



(B) Average max Q-value of Server Mission.



(C) Average reward per time step of Navigation Mission.



(D) Average reward per time step of Server Mission.

FIGURE 4.20: Learning phase results of DQN-based Mission Managers.

episode by the following equations:

$$Average\ max\ Q-value = \frac{\sum_{t=1}^{T_{episode}} \max_a Q(S_t, a)}{T_{episode}} \tag{4.27}$$

$$Average\ reward\ per\ time\ step = \frac{\sum_{t=1}^{T_{episode}} R_t}{T_{episode}} \tag{4.28}$$

Where $T_{episode}$ is the number of time steps in each episode. The increasing trend of the average max Q-value and the average reward per time step indicates the evolution of the adaptation policy and the constant trend then indicates the convergence of the learning algorithm.

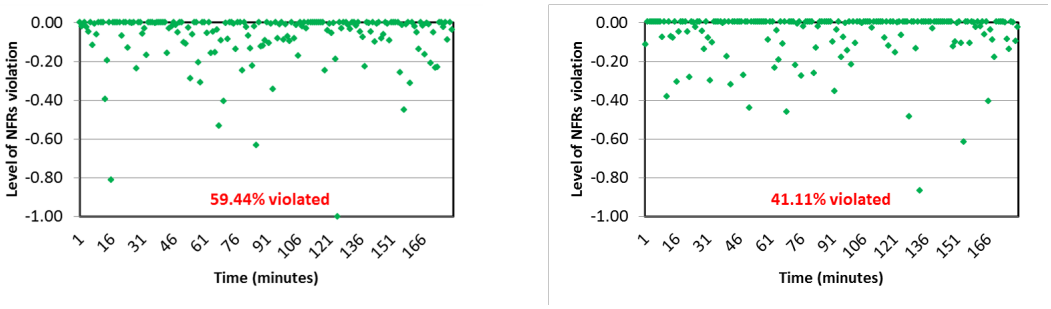
We can also realize the difference of average max Q-value and average reward per time step between two missions. The video server mission has the Q-value and the reward higher than the ones of the navigation mission. This indicates that the capability of guaranteeing non-functional requirements of the video server mission is more feasible than the navigation mission. In fact, the navigation mission is heavily impacted by the external environment such as dynamic obstacles, while the video server mission is less impacted by the external environment, but influenced significantly by its own internal dynamics and the computing environment. The adaptation policies therefore tend to automatically search in the action space to give the

best trade-off between non-functional goals.

4.8.5 Validation of Resulting Adaptation Policy

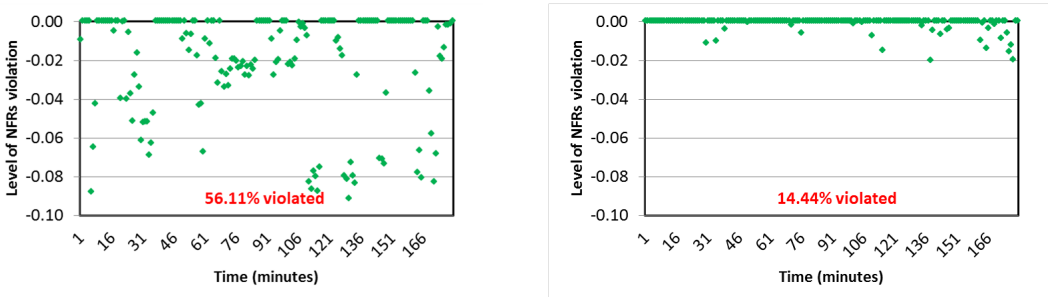
Figures 4.21 and 4.22 present the comparison between the best static case and the run-time adaptation controlled by the mission managers of the two considered robotic missions. The best static case for the navigation mission is the knob $K4 = \{0.8, 10\}$ as chosen in the section of the Q-Learning validation and the one for the video server mission is the knob $K7 = \{400\}$ as the desired performance $RES_{mean\_min} = 800.0 pixels$ (Table 4.9). In fact, it is not evident to identify the best static case for each robotic mission because of dynamic non-functional goals and dynamic operational conditions. Thus, our proposed mission manager will make the run-time adaptation in a self-adaptive manner to take into account these difficulties.

To compare our RTA policy with the best static configuration we propose to use a quantizition of the degree of NFRs violation (corresponding to the negative part of our reward function). The validation phase is deployed within three hours, equivalently 180 episodes for each case. We can see that for the best static case of the navigation mission, in 59.44% of the validation period, the non-functional requirements are violated, while the violation ratio is reduced to 41.11% by the run-time adaptation. For the video server mission, the violation ratio is reduced from 56.11% for the best static case to 14.44% for the run-time adaptation. These results show that our RTA policy can efficiently balance between several non-functional objectives to keep the violation rates as low as possible. The evaluation has been carried on two very different missions in terms of complexity and sensitivity to external events. In



(A) Best Static Case. (B) Run-Time Adaptation.

FIGURE 4.21: Validation of the navigation mission.



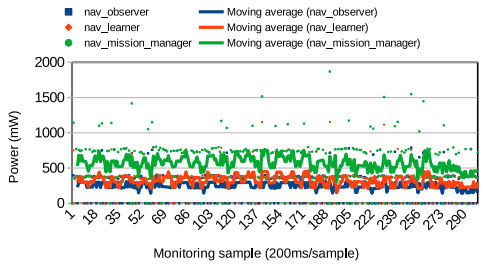
(A) Best Static Case. (B) Run-Time Adaptation.

FIGURE 4.22: Validation of the video server mission.

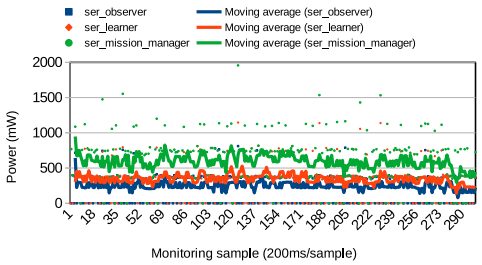
the next paragraph, we intend to integrate the intrinsic dynamic adaptation cost in the final evaluation.

4.8.6 Efficiency of Deep Q-Learning based Mission Manager

In this paragraph, we also characterize the efficiency of the DQN-based Self-Adaptive Mission Manager in terms of computing power consumption in the learning phase (Figure 4.23) and in the planning phase (Figure 4.24). The navigation mission manager consumes an average power of $566mW$ in the learning phase and of $505mW$ in the planning phase, and for the video server mission, an average consumption of $597mW$ in the learning phase and of $510mW$ in the planning phase is found. We can also point out that the monitoring part of each mission manager takes approximately $250mW$. The rest of the total power consumption is dedicated to the learning or planning part.

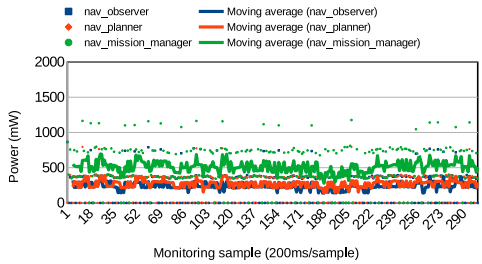


(A) Power consumption of Navigation Mission Manager: $566mW$ in average.

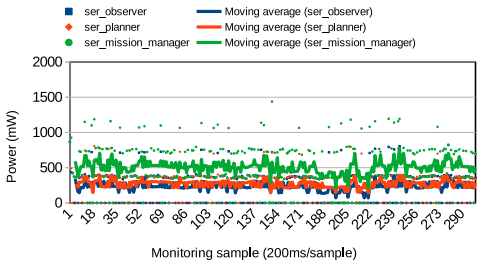


(B) Power consumption of Server Mission Manager: $597mW$ in average.

FIGURE 4.23: Power consumption of DQN-based Mission Managers in the learning phase.



(A) Power consumption of Navigation Mission Manager: $505mW$ in average.



(B) Power consumption of Server Mission Manager: $510mW$ in average.

FIGURE 4.24: Power consumption of DQN-based Mission Managers in the planning phase.

4.8.7 Validation of Transfer Learning

In the context of a continuously learning system, we want to evaluate the online adaptation feasibility to handle changes on the non-functional objectives (tighter power budgets for example). Now we are considering another learning task for two missions. The set of non-functional objectives for each mission is presented in Table 4.11. In fact, the computing power requirement (G_{energy_2}) for each mission has changed and is tighter. We have implemented and compared two types of learning:

- **Learning from scratch**. In this case, the learning process is implemented from scratch as the above section without any initialization of the adaptation model and the parameters for exploration and exploitation are shown in Table 4.10;
- **Transfer learning**. The learned adaptation models in the previous section have been reused as a starting point for these learning tasks. The parameters for exploration and exploitation are also modified as indicated in Table 4.12 with a reduction in the maximum probability of the exploration.

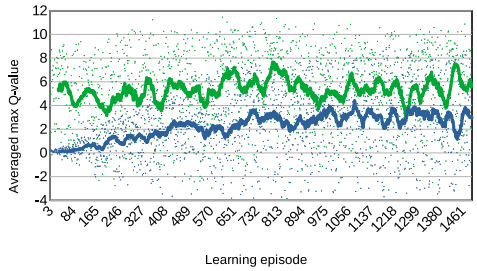
TABLE 4.11: Non-functional goals for Navigation Mission and Server Mission in the transfer learning. In this case, the constraints on the computing power G_{energy_2} are tighter than the ones in Table 4.9

NFRs	Navigation Mission	Server Mission
G_{perf_1}	$V_{mean\_min} = 0.60m/s$	$RES_{mean\_min} = 800.0pixels$
G_{energy_1}	$P_{s\&a\_mean\_max} = 18.5W$	
G_{energy_2}	$P_{c\_mean\_max} = 3.0W$	$P_{c\_mean\_max} = 7.0W$
G_{others_1}	$D_{obs\_min} = 0.7m$	$FPS_{mean\_min} = 25.0Hz$

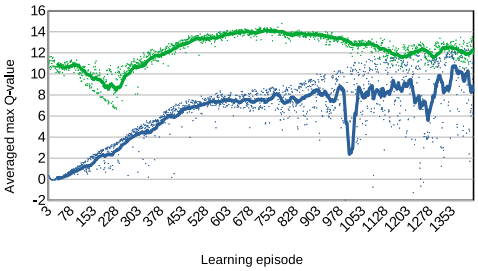
TABLE 4.12: Parameters for exploration and exploitation in the transfer learning of DQN-based Mission Managers.

ϵ_{max}	ϵ_{min}	ϵ_{decay}
0.5	0.01	0.0001

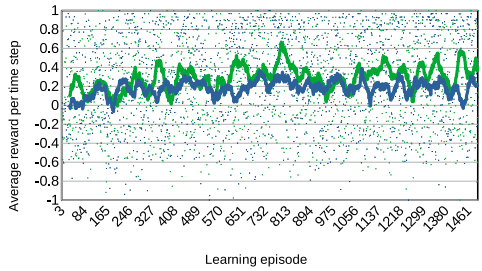
The performances of the transfer learning and the learning from scratch are compared in Figure 4.25. We can realize a significant amelioration between the transfer learning indicated by the green curves and the learning from scratch indicated by the blue curves. The transfer learning is initialized with a high jumping start. For example, for the navigation mission, the average max Q-value of the learning from scratch is initialized from 0.0 while the transfer learning is started from a value around 5.0 and similar for the average max Q-value of the video server mission. In terms of average reward per time step, the transfer learning also gives a higher performance. Thus, the transfer learning accelerates the learning process of the adaptation policy and it can play an important role in the online learning (see Section 4.6.4) when considering to deploy the learning during the planning phase.



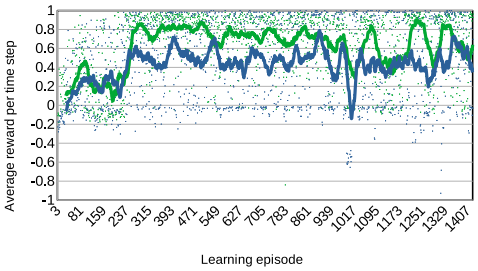
(A) Average max Q-value of Navigation Mission.



(B) Average max Q-value of Server Mission.



(C) Average reward per time step of Navigation Mission.



(D) Average reward per time step of Server Mission.

FIGURE 4.25: Transfer learning results: lines in blue indicate learning from scratch and lines in green indicate transfer learning.

4.9 Summary

This chapter introduces the self-adaptive mission manager. The objective of a self-adaptive mission manager is to minimize the level of violation of a desired set of non-functional requirements during mission operation by automatically reconfiguring mission parameters. Two approaches based on Q-Learning and Deep Q-Learning were presented and validated with two different examples of robotic missions. While the Q-Learning approach is our first effort to formulate and solve the problem of sequential decision-making as a Markov Decision Process, Deep Q-Learning is the core technology we would like to target to improve the generalization of the self-adaptive mission manager. Good opportunities for online learning were explored by considering the intrinsic power and timing costs of the learning phase through the power caracterization and the use of transfer learning.

The mission manager presented in this chapter is responsible for a local management at the robotic mission level. In the next chapter, the multi-mission context will be solved by an adaptive hierarchical multi-mission manager.

Chapter 5

Adaptive and Hierarchical Multi-Mission Manager

Contents

In Chapter 4, we have presented the methodology based on the reinforcement learning for a QoS and energy-aware Mission Manager. This manager defined at the mission level takes into account a desired set of non-functional requirements and reconfigures the considered robotic mission under dynamic operational conditions.

Nowadays, a mobile robotic system can be equipped with many types of sensors or actuators and an advanced computing subsystem. There are therefore many missions that can be deployed at the same time on the same platform. These missions have their own characteristics and priorities, and share the resources of the robotic system. The management in a multi-mission context is a challenge to guarantee a set of requirements of each mission while respecting the whole system constraints. We propose in this chapter a rule-based decision-making methodology and a case-based reasoning technique addressing the problem. While the Mission Manager performs a local adaptation at the mission level, our proposed Multi-Mission Manager operates at the system level with a global adaptation. The Multi-Mission Manager is implemented and validated in the simulation framework and the results are discussed.

This chapter is organized as follows. The context of adaptive multi-mission management as well as some related work are presented in Section 5.1. Section 5.2 then defines formally the problem of multi-mission management. The overview of the methodology that we propose is described in Section 5.3 and the implementation details of the decision-making mechanism are given in Section 5.4. The validation results are presented and discussed in Section 5.5. Finally, this chapter ends with a summary in Section 5.6.

5.1 Related Work

As presented in Section 3.2.7, mobile robotic systems now face the context of multiple missions being deployed simultaneously and sharing robotic resources such as sensors, actuators, applications, computing hardware (CPU, memory, etc) and energy resources. These resources can be shared in a mutex locked or superposed manner. In our thesis, we consider these robotic missions as independent entities that can operate in parallel and share robotic resources in a superposed way. Indeed, three missions presented in Section 3.6 can operate independently and use the robotic resources in their own way. Thanks to the publish-subscribe communication mechanism of a ROS-based robotic system (see Section 2.1.1), sharing sensors and applications does not really cause conflict, a sensor or an application can broadcast the information for all missions. Therefore, the robotic resources such as computing hardware and energy are really our concern because of their limited capacity and of conflicts that can occur between missions. And an adaptive management of this context is crucial to effectively use robotic resources and ensure the quality of service of robotic missions. We review the researches that address the problem of multiple independent entities operating at the same time and sharing the same resources and propose a hierarchical resource management for these systems. These can be multiple applications sharing the same embedded architecture, multiple servers sharing the same data center's energy budget, or multiple services requiring the computing resources of the same cloud system, etc. We note the lack of research in the field of mobile robotics for this problem.

In embedded systems, we focus on the work enabling multiples applications on the same multicore platform with their own performance requirements. The work in [Abeni and Buttazzo, 2001] proposes a hierarchical QoS management for time sensitive applications on the same computing system. This hierarchical control scheme integrates an adaptive reservation mechanism at the system level to allocate the computational resources for multiple applications, and a QoS adaptation at the application level to manage the allocated resources and the QoS of the applications. With the same context of multiple applications on the same computing platform, the work in [Cucinotta et al., 2010; Chasparis et al., 2013] proposes also the integration of application level and resource level QoS control for guaranteeing the QoS of each application while respecting the system-level constraints. Hoffmann et al. [Hoffmann et al., 2011] propose SEEC, a framework for self-aware management of multicore resources. By managing both application and system level, SEEC can resolve shared resource conflicts and meet applications' performance goals.

In large-scale systems such as cloud computing systems or data centers, the work in [Lim, Kansal, and Liu, 2011] presents a hierarchical management for power budgeting for virtualized data centers that enable multiple distributed applications to share the same servers but operate with their individual quality of service guarantees. In this approach, a data center level controller receives the total power capacity

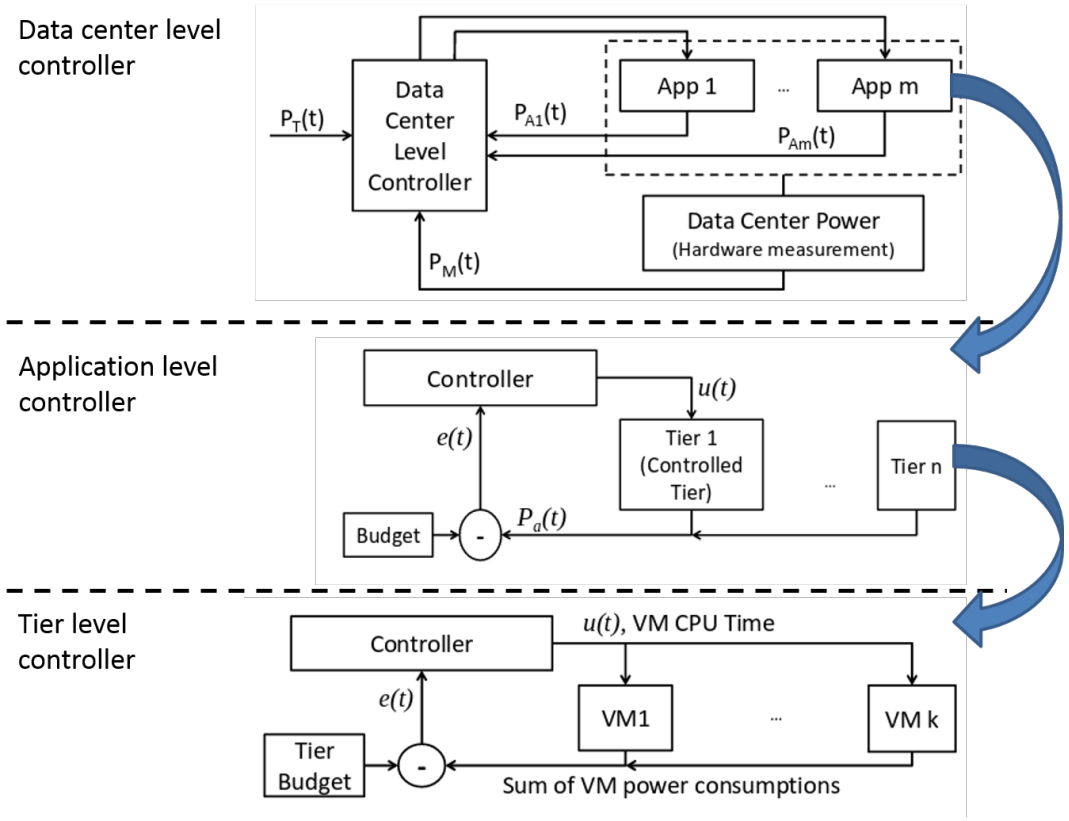


FIGURE 5.1: Hierarchical management in data center (adapted from [Lim, Kansal, and Liu, 2011]). The adaptive management is defined at three levels: data center, application and tier level.

as input and decides the power budget distribution among the application level controllers. The application level controller then monitors and controls the tier level controllers within the application. Thus, there are three hierarchy levels of management in order to guarantee total power capacity and the quality of service of applications. The management structure is depicted in Figure 5.1.

In the mobile robotic systems, the work in [Hernandez-Sosa et al., 2005] proposes a hierarchical management for the component-based robotic systems (see Section 2.6.1). The local management at the component level decides to trade off between the computational resource consumption and the quality of results of this component. The global management at the system level monitors the execution of components and the system level state such as CPU availability and decides to promote or degrade the quality of these components. We consider this study as the first and the only one in the domain of mobile robotic systems that proposes the hierarchical management at many system levels. In the following sections, we will formally present the problem of the multi-mission context and propose our adaptive management methodology.

5.2 Problem Statement

As mentioned above, the sharing of computing hardware and energy resources can cause conflicts between robotic missions due to their limited capacity and dynamic availability. In fact, the problem of sharing computing hardware is defined at the level of the computing platform (Figure 3.6) and the adaptation is usually achieved

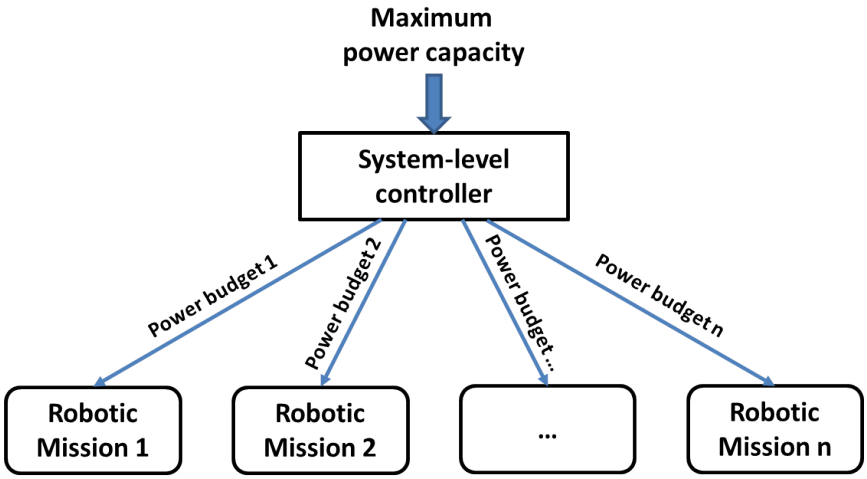


FIGURE 5.2: Problem of determining the power budgets for the robotic missions in a multi-mission context.

by reassigning computing resources (CPU, memory, etc.) or by reconfiguring hardware operating points such as dynamic voltage and frequency scaling (DVFS) of the processors [Kephart and Chess, 2003]. We do not intend to propose an adaptation at this level, but we focus on the total power budget reserved for the robotic missions in a multi-mission context. Indeed, the availability of this total power budget varies according to the operating time of the robotic system due to a decrease in battery capacity or temperature constraints. We aim to guarantee the total power budget by assigning adaptively the power budget for each robotic mission (Figure 5.2). This allocation of the power budget must also take into account the criticality level of the robotic mission among others. In addition, the allocation of a new power budget to the mission will also have an impact on the quality of service of this mission. For example, with a lower power budget, the mission's desired level of quality of service should be reduced. To propose an adaptive management in this multi-mission context, our problem is based on the assumptions presented below.

5.2.1 Assumptions

A1: Computing Power Budget as Dynamic System-Level Constraint

For each robotic mission, the power consumption is divided into sensing and acting, and computing consumption (see Section 3.2.3). In these models, the sensing power consumption of each mission is considered static, while the acting power consumption is not apparent for all robotic missions. For example, a video server mission and a semantic environment understanding mission presented in Section 3.6 do not use any actuators, so the acting power consumption is zero. We focus only on the dynamic part of the power consumption of the robotic mission to provide runtime adaptation. Thus, we limit our problem to cope with the maximum computing power budget reserved to the robotic missions. This budget is dynamic during robot operation, for example, because of battery state of charge or temperature constraint as mentioned above. At the time t, the maximum computing power budget reserved to all the robotic missions is denoted by $P^{\Sigma}_{c\_max}(t)$. We assume that we have n robotic missions denoted respectively by $M1$, $M2$, ..., Mn and the computing power budgets allocated to these missions are represented by $P^{M1}_{c\_ref}(t)$, $P^{M2}_{c\_ref}(t)$, ..., $P^{Mn}_{c\_ref}(t)$. Thus,

at the time t, we have the following system-level constraint:

$$P^{\Sigma}_{c\_ref}(t) = \sum_{i=1}^{n} P^{Mi}_{c\_ref}(t) = P^{M1}_{c\_ref}(t) + P^{M2}_{c\_ref}(t) + ... + P^{Mn}_{c\_ref}(t) \leq P^{\Sigma}_{c\_max}(t) \quad (5.1)$$

A2: Power Mode and Priority of Robotic Missions

We assume that each robotic mission has some computing power consumption modes. These modes are in fact the constraints of computing power consumption of each mission. We propose three active modes denoted by *High*, *Medium* and *Low*, and one non-active mode denoted by *Off*:

- In the *High* mode, the computing power consumption constraint is light and easy to guarantee, so the run-time adaptation strives to optimize other non-functional requirements, such as promoting the mission's quality of service;
- The power constraint in the *Medium* mode is tighter than the *High* mode, but the desired quality of service is still easy to achieve;
- The *Low* mode is the strictest mode and the desired quality of service should be degraded to guarantee the constraint of computing power consumption;
- The *Off* mode means that the mission is not actually active.

We propose the number of power modes in a subjective way. Our objective is to demonstrate that with different power budgets or limits, the quality of service of the robotic mission would be different and regulating the desired quality of service of the mission can contribute to the guarantee of the computing power consumption constraint. Table 5.1 describes the definition of computing power modes of n robotic missions. Hence, at the time t, we have the following constraint:

$$\forall Mi, P^{Mi}_{c\_ref}(t) \in \{P^{Mi}_{c\_ref\_h}, P^{Mi}_{c\_ref\_m}, P^{Mi}_{c\_ref\_l}, 0\} \quad (5.2)$$

TABLE 5.1: Definition of the computing power consumption modes of the robotic missions.

Mode	Mission M1	...	Mission Mi	...	Mission Mn
High	$P^{M1}_{c\_ref\_h}$	...	$P^{Mi}_{c\_ref\_h}$	...	$P^{Mn}_{c\_ref\_h}$
Medium	$P^{M1}_{c\_ref\_m}$	...	$P^{Mi}_{c\_ref\_m}$	...	$P^{Mn}_{c\_ref\_m}$
Low	$P^{M1}_{c\_ref\_l}$	...	$P^{Mi}_{c\_ref\_l}$	...	$P^{Mn}_{c\_ref\_l}$
Off	0	...	0	...	0

The priority is also assigned to each robotic mission denoted as $Pr^{Mi}(t)$. In a multi-mission context, the mission with a higher priority should have a higher computing power consumption mode. These priorities can be dynamic due to changes in the users requests or to internal changes. Each mission should run at the highest power mode as long as possible in order to facilitate the guarantee of the non-functional requirements, but the decision can be made to degrade the power mode or even stop the mission (change to power mode *Off*). The objective remains to keep the maximum number of missions at active modes.

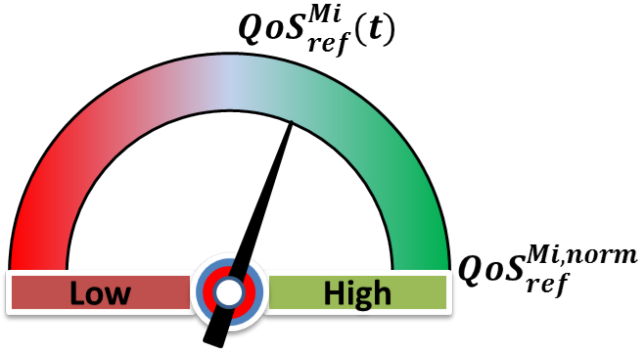


FIGURE 5.3: Quality of service knob of the mission Mi. A normal level of quality of service $QoS_{ref}^{Mi,norm}$ is defined for the mission. The current level of QoS $QoS_{ref}^{Mi}(t)$ then can be regulated.

A3: Quality of Service of Robotic Missions as a Configuration Knob

As defined in Section 3.2.4, the mission-specific performance (M-Perf) can be considered as a quality of service. We assume in this chapter that this desired quality of service of the mission Mi at the time t denoted by $QoS_{ref}^{Mi}(t)$ can be regulated. It means that the quality of service of each robotic mission can be degraded or promoted. We define also a normal level of quality of service of the mission Mi represented by $QoS_{ref}^{Mi,norm}$ (Figure 5.3). This normal QoS can be obtained via an offline characterization as presented in Chapter 3 and it is usually included in the knowledge of the robotic mission. The quality of service requirement less constrained should facilitate the guarantee of other non-functional requirements. For example, in the context of a tight computing power budget, a lower level of quality of service would be appreciated so that this power budget is not exceeded.

5.2.2 Problem Definition

Based on the above assumptions, the problem is now stated as how to configure the appropriate computing power consumption mode and the desired quality of service for each mission within the context of dynamic system-level constraints and dynamic mission priorities. This global run-time management leads to the modification of non-functional requirements for each robotic mission. Thus, the Mission Manager at the mission level must adapt to these changing non-functional goals. We propose in the next section an adaptive hierarchical management of the multi-mission context and a reasoning technique based on case-based reasoning to cope with the problem of changing NFRs.

5.3 Methodology Overview

5.3.1 Hierarchical Management

Our management methodology is depicted in Figure 5.4. The adaptive management is divided into two hierarchical levels as follows:

- **At the local management** of each robotic mission, a Mission Manager observes the current state related to the mission in order to adapt dynamically some

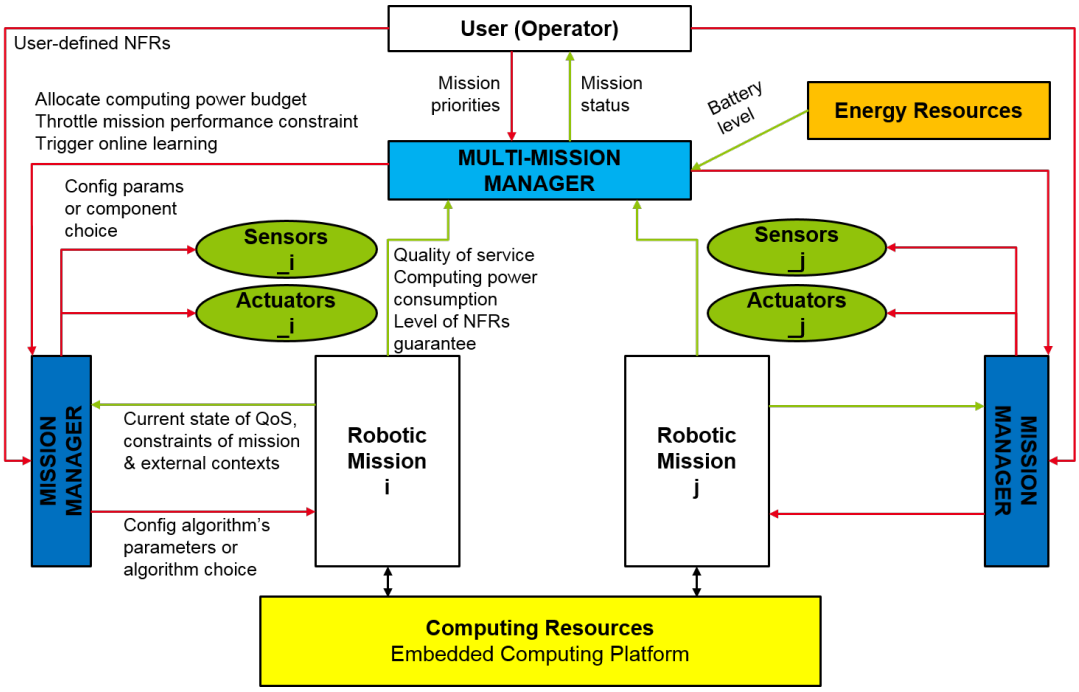


FIGURE 5.4: Overview of the adaptive management methodology in a multi-mission context. The management is divided into two hierarchies: the Mission Managers at the mission levels and a Multi-Mission Manager at the system level.

interesting configuration while respecting the desired non-functional requirements;

- **At the global management**, a Multi-Mission Manager observes the current state of system constraints, the current priorities of robotic missions, and the current level of guarantee of non-functional requirements of the different missions in order to choose the appropriate power mode, regulate the quality of service and trigger the online learning (see the definition of the online learning in Section 4.6.4) for each robotic mission.

We can say that each Mission Manager receives the **adaptation request** from the users and the Multi-Mission Manager and gives the **adaptation response** for the Multi-Mission Manager:

- **Adaptation request**: is a set of non-functional requirements applied to the robotic mission. The Mission Manager is responsible for guaranteeing these requirements;

- **Adaptation response**: can be considered as the adaptation results of the Mission Manager. The information that will be provided for the Multi-Mission Manager includes the obtained quality of service, the computing power consumption and the level of NFRs violation.

The adaptation of the Multi-Mission Manager and Mission Managers is executed in a time-synchronized manner as depicted in Figure 5.5. At a larger period denoted as T_{global}, the Multi-Mission Manager makes its decisions. Within this global period, each Mission Manager makes its decisions with a finer period denoted as T_{local_i} for the mission Mi and T_{local_j} for Mj. The episode concept of the missions is therefore

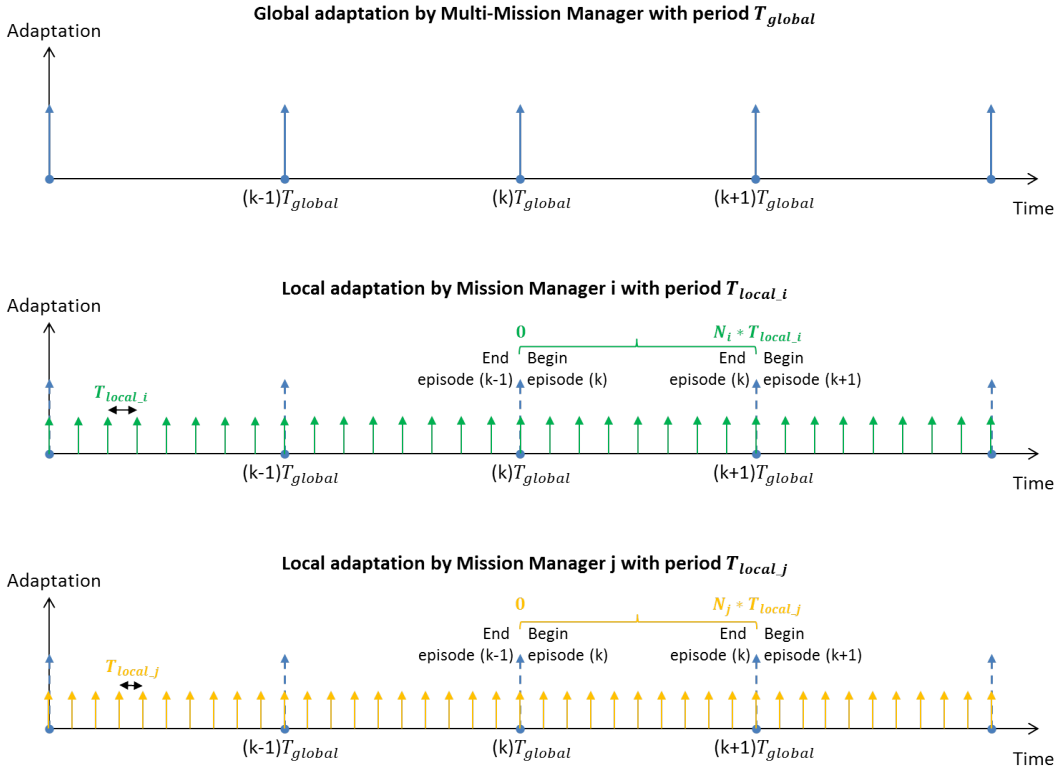


FIGURE 5.5: Timing synchronization between Multi-Mission Manager and Mission Managers. Each Mission Manager has its local adaptation period. The episode of each Mission Manager is then synchronized with the global adaptation period of the Multi-Mission Manager.

defined by the global period and the episode of all missions has the same duration T_{global}. Thus, the adaptation at the mission level is independent from each other and is coordinated at the global level. The conflict of the adaptation decisions between local levels, as well as between the local and global level is therefore avoided.

The algorithm of the hierarchical management in a multi-mission context is described in Algorithm 1. For each decision epoch identified by T_{global}, the Multi-Mission Manager checks whether the maximum computing power budget $P^{\Sigma}_{c\_max}(t)$ or mission priorities change to reallocate the computing power budget for each active mission. Then, for each mission *Mi*, if a new active power mode (*High*, *Medium*, *Low*) is determined, the Multi-Mission Manager will decide to deactivate the mission's online learning if it is in progress and initialize the normal QoS level $QoS^{Mi,norm}_{ref}$ for the mission. Otherwise, if the new power mode is *Off*, the mission is completely closed. If the mission power mode is not changed, the Multi-Mission Manager will check the adaptation response of the mission to decide to trigger the online learning and/or adjust the quality of service $QoS^{Mi}_{ref}(t)$ for the mission. Finally, a new adaptation request will be applied to the mission *Mi*. The details of these three decisions will be presented in the next section.

Algorithm 1: Hierarchical management in a multi-mission context.

```
for each decision epoch of Multi-Mission Manager do
  if maximum computing power budget or mission's priorities change then
    reallocate computing power budget for each active mission (Section 5.4.1)
  end if
  for each active mission Mi do
    if power mode changes then
      if still in active mode (High, Medium, Low) then
        deactivate online learning of mission if in progress
        initialize normal mission's quality of service
      else
        shutdown completely mission
        continue
      end if
    else
      check condition to throttle quality of service of mission (Section 5.4.2)
      check condition to trigger online learning for mission (Section 5.4.3)
    end if
    apply new adaptation request (NFRs) to mission Mi
  end for
end for
```

5.3.2 Adaptation of Changing NFRs with Case-Based Reasoning

As explained above, the adaptation of the Multi-Mission Manager leads to the modification of the non-functional requirements (a new adaptation request) applied to each robotic mission. Thus, the adaptation model (Q-Table or Q-Network) of each robotic mission should be modified to better meet the new requirements. We propose a reasoning technique called Case-Based Reasoning (see Section 2.3) as a solution to this problem.

In this problem, a case includes its context and solution. Its context represents a set of NFRs applied to the robotic mission and the solution is the run-time adaptation model such as Q-Table or Q-Network. Figure 5.6 represents the case base organization for each robotic mission. The computing power consumption mode is the first criteria in order to index the case. In each group of power mode, there are many cases that have other changing requirements. A **new case** means a new set of requirements (**new adaptation request**) and its adaptation model is unknown. Thus, the idea of Case-Based Reasoning is to use the most similar case in a case database to resolve a new case and then update the new case into the database.

The case similarity is defined by Euclidean distance. Assume that we have two cases a and b with their contexts of m non-functional requirements denoted by c_a and c_b as follows:

$$c_a = \{nfr_k^a\}_{k=1}^m \tag{5.3}$$

$$c_b = \{nfr_k^b\}_{k=1}^m \tag{5.4}$$

The Euclidean distance defines the similarity between two cases $sim(c_a, c_b)$ as follows:

$$sim(c_a, c_b) = \sqrt{\sum_{k=1}^{m} (nfr_k^a - nfr_k^b)^2} \tag{5.5}$$

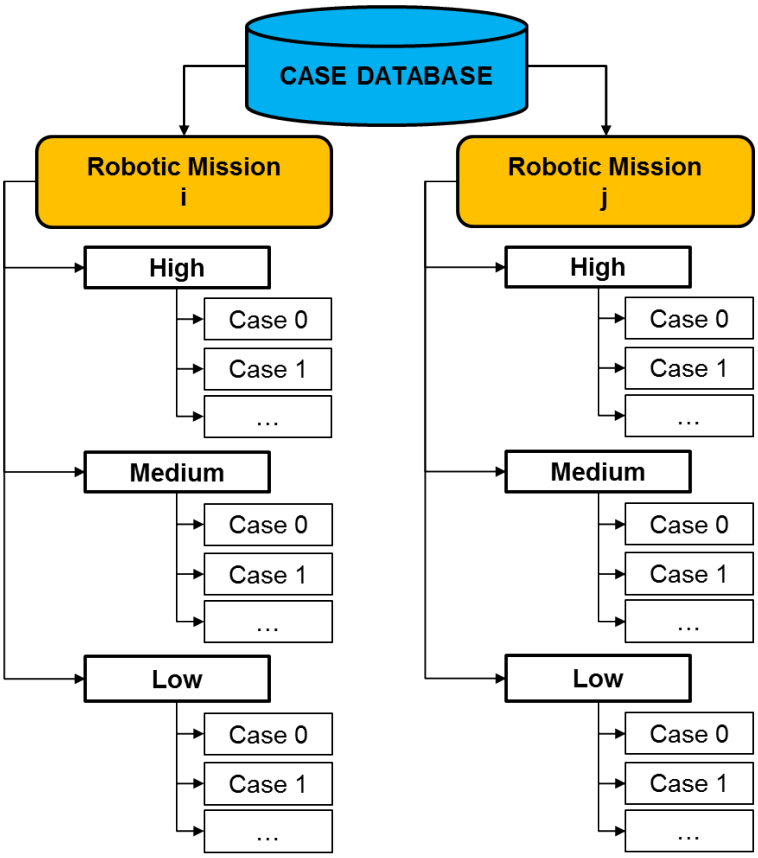


FIGURE 5.6: Case-Based Reasoning for adapting to changing NFRs. For the case base of each mission, the case is first indexed by the computing power mode (*High*, *Medium*, and *Low*) and then by other non-functional requirements.

The smaller the metric $sim(c_a, c_b)$ is, the more similar the two cases are. Therefore, the case in database most similar to a new case is the case that has the smallest distance to this new case.

The cycle of Case-Based Reasoning for each robotic mission is illustrated in Figure 5.7. There are four main steps:

- **Retrieve**: searches the most similar case in case database following the similarity metric in Equation 5.5;
- **Reuse**: applies the adaptation model (Q-Table, Q-Network) of the most similar case to the Mission Manager. Then, the Mission Manager uses this model for local run-time adaptation;
- **Revise**: equipped with an online learning capability strives to evolve the proposed adaptation model if the case is existing in the case database or to generate a new adaptation model from the proposed model if the case is not existing in the case database;
- **Retain**: adds the new case composed of context and refined solution into case database.

Based on the CBR paradigm, transfer learning and online learning capability, the adaptation capability of the robotic missions will evolve over time to deal with more dynamic non-functional requirements as well as dynamic operational conditions.

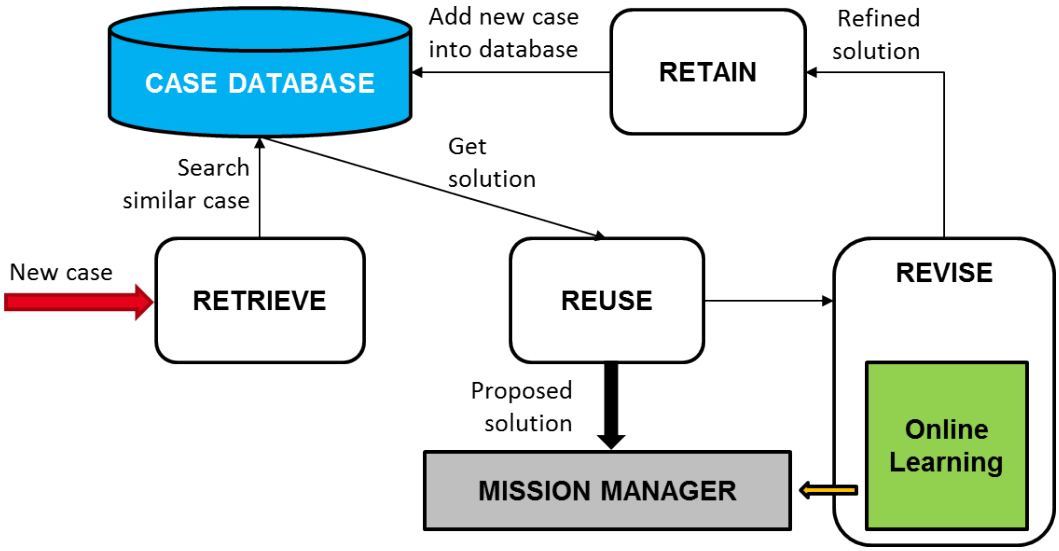


FIGURE 5.7: Case-Based Reasoning cycle for each robotic mission: Retrieve, Reuse, Revise and Retain. The online learning capability is included in the revise step.

5.4 Decision-Making of Multi-Mission Manager

5.4.1 Reallocating Computing Power Budget

The problem of reallocating the computing power budget for each robotic mission is mainly based on the two assumptions A1 and A2 (Section 5.2.1). We propose a priority-based reservation algorithm to choose the appropriate power mode for each robotic mission under the dynamic constraints of maximum power budget and dynamic mission priorities. The MAPE paradigm as presented in Chapter 2 is also applied for explaining this decision-making problem.

Monitoring

The first metric to be monitored is the maximum computing power budget reserved to all robotic missions $P^{\sum}_{c\_max}(t)$. In fact, we consider that this metric is derived directly from the current battery state of charge level $SoCh(t)$. It means that there is a mapping rule between the two quantities. We assume that this rule is available for our Multi-Mission Manager. In the future, we can obtain this rule by considering exactly the power consumption of all components in the robotic platform.

The other metrics are the priorities of the robotic missions: $\{Pr^{M1}(t), ..., Pr^{Mn}(t)\}$. These priorities can be internally modified or can be reconfigured by the robot users.

Analyzing and Planning

In this phase, a priority-based reservation algorithm is implemented to choose the appropriate power mode for each robotic mission with the current observation. In this priority-based search, we sort the missions by ascending priority order and the mission with the least priority will be firstly impacted. The power mode of each mission in this ascending priority order is searched in the set of $\{High, Medium, Low\}$. When all missions are at the Low power mode but Equation 5.1 is not still satisfied,

we will choose the *Off* mode for the mission with the smallest priority. The search process is iterated until the constraint in Equation 5.1 is guaranteed.

The power mode search algorithm can be found in Algorithm 2. This algorithm is generalized for any number of robotic missions ($n\_active\_missions$) and any number of power modes (n_power_modes). For example, in the assumption A2, we assume 4 computing power modes, so $n\_power\_modes = 4$. The possible power budgets of a mission are saved into an array as presented in Equation 5.2, where the index 0 indicates the *High* mode and the index ($n\_power\_modes - 1$) means the *Off* mode. Then, the power budgets of all active missions are organized into an 2D array $power\_missions$ in that order descending priority, where the index 0 indicates the mission with the highest priority and the index ($n\_active\_missions - 1$) indicates the mission with the lowest priority. The output of the search process is an array $found\_modes$ indicating the power mode of each mission.

Algorithm 2: Power mode search algorithm of the Multi-Mission Manager.

Input: $P^{\Sigma}_{c\_max}(t)$
Input: $n\_active\_missions$, n_power_modes
Input: $power\_missions[n\_active\_missions][n\_power\_modes]$
Output: $found\_modes[n\_active\_missions]$

1: $found\_modes[n\_active\_missions] \leftarrow \{0\}$
2: $n \leftarrow 0$
3: $found \leftarrow False$
4: **while** $n < n\_active\_missions$ **do**
5: $m \leftarrow 0$
6: **while** $m < n\_active\_missions - n$ **do**
7: $sum \leftarrow 0$
8: **for** $i = 0$ **to** $n\_active\_missions - 1 - n$ **do**
9: $sum \leftarrow sum + power\_missions[i][found\_modes[i]]$
10: **end for**
11: **if** $sum \leq P^{\Sigma}_{c\_max}(t)$ **then**
12: $found \leftarrow True$
13: **return** $found\_modes[n\_active\_missions]$
14: **end if**
15: **if** $found\_modes[n\_active\_missions - 1 - n - m] < n\_power\_modes - 2$ **then**
16: $found\_modes[n\_active\_missions - 1 - n - m] \leftarrow$
$found\_modes[n\_active\_missions - 1 - n - m] + 1$
17: **else**
18: $m \leftarrow m + 1$
19: **end if**
20: **end while**
21: $found\_modes[n\_active\_missions - 1 - n] \leftarrow n\_power\_modes - 1$
22: **for** $i = 0$ **to** $n\_active\_missions - 1 - n - 1$ **do**
23: $found\_modes[i] \leftarrow 0$
24: **end for**
25: $n \leftarrow n + 1$
26: **end while**

Executing

When the power mode of each mission is reconfigured by the Multi-Mission Manager, the case-based reasoning technique will use the power mode as the first indicator in order to retrieve the most appropriate adaptation model for the robotic mission (see *Retrieve* step in Section 5.3.2).

5.4.2 Throttling Quality of Service

The desired quality of service of the robotic mission can be user-defined or set as a default value acknowledged by the Mission Manager. While the Mission Manager strives to trade off all the non-functional requirements, the computing power budget is considered as a hard constraint and should not be surpassed. We propose a run-time adaptation that throttles the desired quality of service to guarantee the computing power budget reserved to each robotic mission.

Monitoring

The computing power consumption of each mission is monitored in m successive episodes. It means that at the episode t of the mission Mi we have collected a set of metrics $\{P_c^{Mi}(t-m+1), ..., P_c^{Mi}(t-1), P_c^{Mi}(t)\}$ as well as their corresponding power budgets.

Analyzing and Planning

By evaluating the level of violation or satisfaction of the computing power requirement, this run-time adaptation will decide to degrade or promote the desired quality of service level compared to the current desired quality of service. The level of violation or satisfaction of computing power requirement $\Delta P_c^{Mi}(t)$ is defined by

$$\Delta P_c^{Mi}(t) = 1 - \frac{P_c^{Mi}(t)}{P_{c\_ref}^{Mi}(t)} \tag{5.6}$$

$\Delta P_c^{Mi}(t) \geq 0$ means the power requirement at the time t is satisfied and $\Delta P_c^{Mi}(t) < 0$ means that it is currently violated. The averaged level of violation or satisfaction within m episodes $\Delta P_c^{Mi,avg}(t)$ is calculated by

$$\Delta P_c^{Mi,avg}(t) = \frac{\sum_{j=1}^{m} \Delta P_c^{Mi}(t-j+1)}{m} \tag{5.7}$$

We propose now a set of rules for determining the adaptation of desired quality of service for the mission Mi:

- If the power requirement is satisfied within m successive episodes, we will promote the desired quality of service with a coefficient of $\Delta P_c^{Mi,avg}(t) > 0$;
- If the power requirement is violated within m successive episodes, we will degrade the desired quality of service with a coefficient of $\Delta P_c^{Mi,avg}(t) < 0$.

The desired quality of service of the mission Mi at the time t is therefore regulated as follows

$$QoS_{ref}^{Mi}(t) = (1 + \Delta P_c^{Mi,avg}(t)) \times QoS_{ref}^{Mi}(t-1) \tag{5.8}$$

To avoid that the quality of service can be degraded to 0, we propose an arbitrary lower limit of $QoS_{ref}^{Mi}(t)$ such as $10\% \times QoS_{ref}^{Mi,norm}$, so we have the following relation:

$$10\% \times QoS_{ref}^{Mi,norm} \leq QoS_{ref}^{Mi}(t) \leq QoS_{ref}^{Mi,norm} \tag{5.9}$$

In fact, the choice of m, number of successive episodes, as well as the calculation method of $\Delta P_c^{Mi,avg}(t)$ will influence the reactivity of this decision. In this study, we strive to propose a simple (nearly linear) calculation and a simple rule set to demonstrate the role of this decision. Choosing a really optimal calculation method needs a further study and it is currently out of scope of this thesis.

Executing

Once the desired quality of service of the robotic mission is changed, the case-based reasoning uses it now as the second indicator to choose the most appropriate adaptation model in the current case database for the corresponding Mission Manager (see *Retrieve* step in Section 5.3.2).

5.4.3 Triggering Online Learning

With the two above decisions given by the Multi-Mission Manager, an adaptation request or a set of desired non-functional requirements applied to each Mission Manager is divided into two situations:

- **The request is existing in the case database of this mission**. The adaptation model (Q-Table or Q-Network) is already learned and can be directly reused to control the robotic mission. However, in order to cope with more scenarios that are unanticipated in the learning phase, the online learning can be deployed to improve the adaptation model;
- **The request is not existing in the case database of this mission**. In this situation, we have a new case and the optimal adaptation model is unknown. Thus, the adaptation model of the most similar case can be used as an initialization point for the online learning and the objective is to find the new adaptation model for this new case.

In this decision-making problem, the Multi-Mission Manager will decide when to start and stop the online learning for each Mission Manager:

- **Start criteria**. For each mission, the Multi-Mission Manager will monitor the level of NFRs violation within m successive episodes (for sake of simplicity, the same m as the decision of throttling QoS is used). In fact, this level is the negative part of the reward function obtained at the end of each episode. If these consecutive values are lower than a (negative) threshold, the Mission Manager's current adaptation model does not seem to adapt to the new scenarios. Thus, the Multi-Mission Manager will decide to start the online learning for tuning the adaptation model of the mission;
- **Stop criteria**. There are three criteria to stop the online learning in this case: (1) new power mode, (2) new QoS case or (3) end of permitted timeout for the online learning. The criteria (1) and (2) will require the case-based reasoning to choose new appropriate adaptation model, the online learning must therefore be stopped. The criteria (3) relates the fact that we define a timeout (number of learning episodes for example) for the online learning.

At the end of each online learning process, the case with its adaptation model must be stored in the case database of the mission. Thus, the number of adaptation models of each robotic mission will be gradually increased upon the mission deployment. However, the number of adaptation models should be limited to guarantee the memory efficiency. We will detail this in the next section of implementation and validation.

5.5 Implementation and Validation

5.5.1 Simulated Scenario

In this chapter, we deploy two robotic missions in parallel: an autonomous navigation mission denoted as $M1$ and a video server mission denoted as $M2$ as presented for validating the mission managers in the previous chapter. The computing power consumption modes of these missions are defined in Table 5.2. In fact, the choice of these modes is based on the characterization results in Chapter 3 and the interpretation of power modes in the assumption A2 (see Section 5.2.1). We consider also in this scenario that the priority of the navigation mission is higher than the one of the video server mission. The normal level of QoS of the navigation mission is defined as $QoS_{ref}^{M1,norm} = 0.6m/s$ and the one of the video server mission is $QoS_{ref}^{M2,norm} = 800pixels$ (an image resolution of 800×800). The other non-functional goals for the two missions are kept the same as presented in Table 4.9.

TABLE 5.2: Computing power consumption modes of the navigation mission and the video server mission.

Mode	Mission M1	Mission M2
High	$P_{c\_ref\_h}^{M1} = 4W$	$P_{c\_ref\_h}^{M2} = 8W$
Medium	$P_{c\_ref\_m}^{M1} = 3W$	$P_{c\_ref\_m}^{M2} = 7W$
Low	$P_{c\_ref\_l}^{M1} = 2W$	$P_{c\_ref\_l}^{M2} = 6W$
Off	0	0

The system-level constraint such as the maximum computing power capacity allocated to the robotic missions $P_{c\_max}^{\Sigma}(t)$ is also simulated. It is decreasing $1W$ each 60 minutes from $12W$ to $8W$ within 300 minutes as shown in Table 5.3.

TABLE 5.3: Simulated maximum computing power capacity allocated to the robotic missions.

Time t	1 - 60	61 - 120	121 - 180	181 - 240	241 - 300
$P_{c\_max}^{\Sigma}(t)$	$12W$	$11W$	$10W$	$9W$	$8W$

5.5.2 Implementation

Initializing Case Database

We build off-line the case database for each mission. We assume that the case is defined only by the computing power mode and the desired level of quality of service and the other non-functional requirements are kept unchanged. The case database is initialized as shown in Figure 5.8. The adaptation models are found by applying the

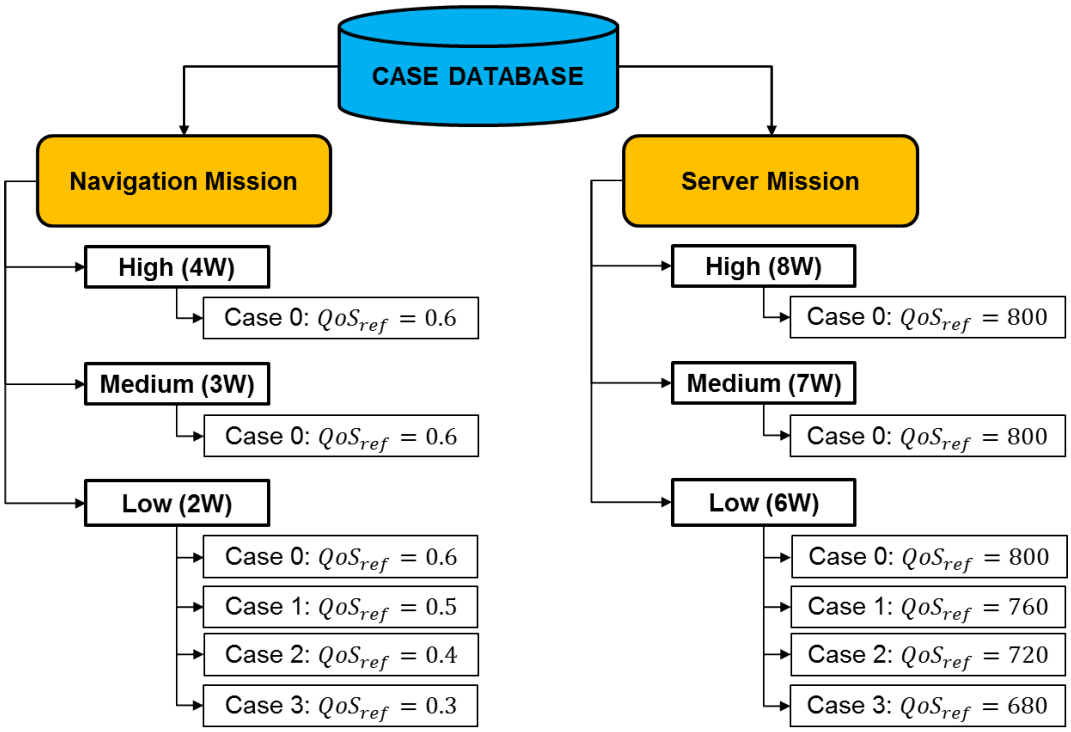


FIGURE 5.8: An initialization of the case database for the navigation mission and the server mission.

transfer learning technique as presented in Section 4.8.7. In fact, for each computing power mode, we limit the number of desired quality of service to 4 levels:

$$QoS_{ref}^{M1} \in \{0.6, 0.5, 0.4, 0.3\} m/s \tag{5.10}$$

$$QoS_{ref}^{M2} \in \{800, 760, 720, 680\} pixels \tag{5.11}$$

This will reduce the memory requirement to save the case database as well as simplify the organization and the access to the case database. A complete case database of each robotic mission would be composed of 12 adaptation models (Q-Networks). In fact, a memory footprint of $2.2MB$ is used to save a Q-Network in the file system. Thus, a case database of 12 Q-Networks of a robotic mission will occupy $12 \times 2.2MB = 26.4MB$. In a context of n robotic missions, the memory footprint in the file system is $n \times 26.4MB$.

Reallocating Computing Power Budget

Based on the current maximum power capacity of the system (Table 5.3) and the priority of each mission (the priority of the navigation mission is higher than the one of the video server mission in this scenario), the reservation algorithm (Algorithm 2) will decide to reallocate the computing power budget for each mission. The resulting power mode for each mission is presented in Table 5.4. The allocation algorithm guarantees that the total power budget reserved to the two missions is less than or equal to the maximum computing power capacity (Equation 5.1).

TABLE 5.4: Computing power budget allocated dynamically to each robotic mission (H: High mode, M: Medium mode, L: Low mode).

Time t	**1 - 60**	**61 - 120**	**121 - 180**	**181 - 240**	**241 - 300**
$P^{M1}_{c\_ref}(t)$	H, 4W	H, 4W	H, 4W	M, 3W	L, 2W
$P^{M2}_{c\_ref}(t)$	H, 8W	M, 7W	L, 6W	L, 6W	L, 6W

Throttling Quality of Service

For this decision-making problem, the Multi-Mission Manager monitors the computing power consumption of each robotic mission within $m = 3$ successive episodes to decide to degrade or promote the desired level of quality of service as presented in Section 5.4.2. In fact, the number of successive episodes, $m = 3$ in this case, is empirically chosen and it has an impact on the reactivity of this decision as mentioned in Section 5.4.2. Once the computing power mode or the level of QoS has been changed, the Multi-Mission Manager uses the case-based reasoning technique to choose the most appropriate adaptation model for each robotic mission.

5.5.3 Results and Discussions

Figure 5.9 presents the characteristic of computing power consumption of the two mentioned robotic missions controlled by the Multi-Mission Manager. We can realize that this characteristic is adapted corresponding to the current maximum computing power capacity (Table 5.3). When the power capacity is high such as the case of 12W, 11W or 10W, the power constraint is lighter and the adaptation is therefore more feasible. Whereas, the case of 8W is extremely low, the adaptation is more difficult and it is sometimes found that the actual power consumption exceeds the reference value.

The characteristics of computing power consumption and quality of service of two missions are presented in Figure 5.10, 5.10a and 5.10c for the navigation mission, and 5.10b and 5.10d for the video server mission. The computing power budget of

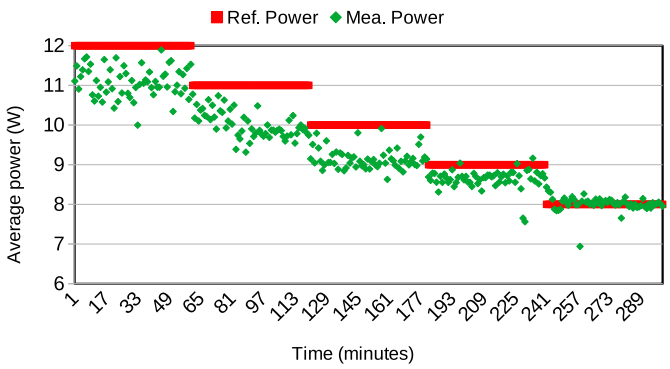
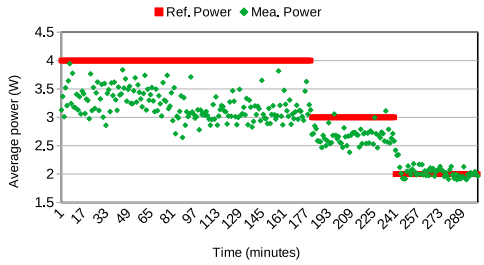
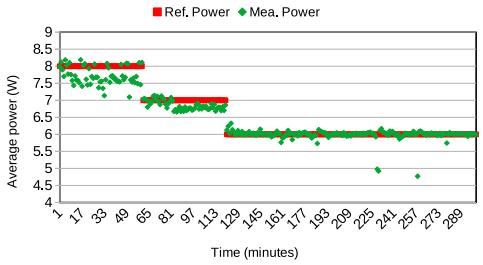


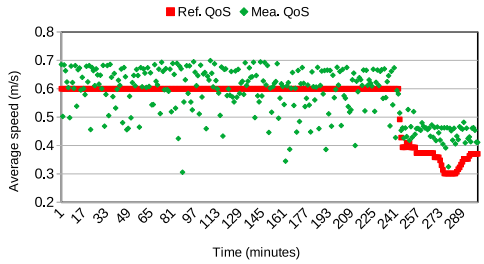
FIGURE 5.9: System-level computing power consumption controlled by the Multi-Mission Manager. The red points indicate the maximum computing power allocated to two missions and the green points are the measured values or the actual computing power (averaged during an episode or a decision epoch of 1 minute of the Multi-Mission Manager) consumed by two missions.



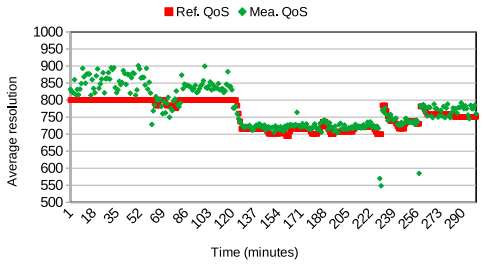
(A) Computing power consumption of Navigation Mission.



(B) Computing power consumption of Server Mission.



(C) Quality of service of Navigation Mission.



(D) Quality of service of Server Mission.

FIGURE 5.10: Computing power consumption and quality of service of each mission controlled by the Multi-Mission Manager. The red points indicate the reference values or the adaptation request (computing power budget, desired quality of service) of each mission. The green points are respectively the measured values (averaged during an episode of 1 minute of each Mission Manager) of each mission.

each mission is allocated by the priority-based reservation algorithm as shown in Table 5.4. Under each computing power mode, by observing the power consumption of each mission over $m = 3$ successive episodes, the level of quality of service of each mission is throttled by the decision of throttling QoS of the Multi-Mission Manager. Based on the computing power mode as the first indicator and the desired quality of service as the second indicator, the adaptation model of each Mission Manager is respectively retrieved in the case database by the CBR technique.

For the two missions, we can see that under each different power mode, the mission gives different characteristics of quality of service. For example, in the *High* mode of the navigation mission, the QoS can reach a maximum value of $0.7m/s$, while the value for the *Medium* mode is $0.68m/s$ and for the *Low* mode is $0.48m/s$ (Figure 5.10c). The same characteristic is found for the video server mission when in the *High* mode, the QoS can reach a level of 900, a lower level for the *Medium* mode is seen and a degradation of the QoS is clearly visible in the *Low* mode (Figure 5.10d). In fact, the actual mission's quality of service is also influenced by other dynamic factors. Our run-time adaptation strives to guarantee the desired quality of service or to maximize it if possible within the permitted computing power budget.

In addition to the run-time adaptation of the mission manager, the decision to throttle the mission's quality of service of the Multi-Mission Manager can be considered reactive in the event of a violation of the computing power budget. Indeed, the computing power consumption of each mission is monitored over $m = 3$ successive episodes. If the consumption of all 3 episodes exceeds the budget, the desired QoS

will be degraded. For example, for the navigation mission, when the computing power budget decreases from $3W$ to $2W$ at the episode 241, the computing power consumption of three episodes: 241, 242 and 243 exceeds $2W$ (Figure 5.10a), the desired quality of service is therefore reduced to $0.5m/s$ (Figure 5.10c). Whereas, if the consumption of all 3 episodes is less than or equal to the budget, the desired QoS will be promoted (at the episode 280 of the navigation mission or the episode 262 of the video server mission for example).

In this simulated scenario, we have built offline the case database for the two robotic missions. The validation shows the expected results and the online learning is not necessary to be triggered in this scenario. Thus, we tend to apply the online learning technique when considering the experimentation deployed on the real robot and more validation results will be presented in the next chapter.

5.6 Summary

In this chapter, we present the Multi-Mission Manager that deals with the context of multiple missions with different functional, non-functional requirements and shared resources in the same robotic system. The Multi-Mission Manager takes the dynamic system-level constraints and the dynamic priorities of missions as inputs and decides to reallocate the computing power budgets for the robotic missions, to throttle the quality of service of missions or to trigger the online learning for evolving or generating the adaptation models. A simulated scenario with two missions and a global computing power budget decreasing in time validates the power budget reallocation and quality of service throttling methodology.

As the last step in validating our methodology, the next chapter will present the experimentation on a real mobile robotic platform. The DQN-based mission managers and the multi-mission manager will be deployed in the real framework and their performance and efficiency will be evaluated.

Chapter 6

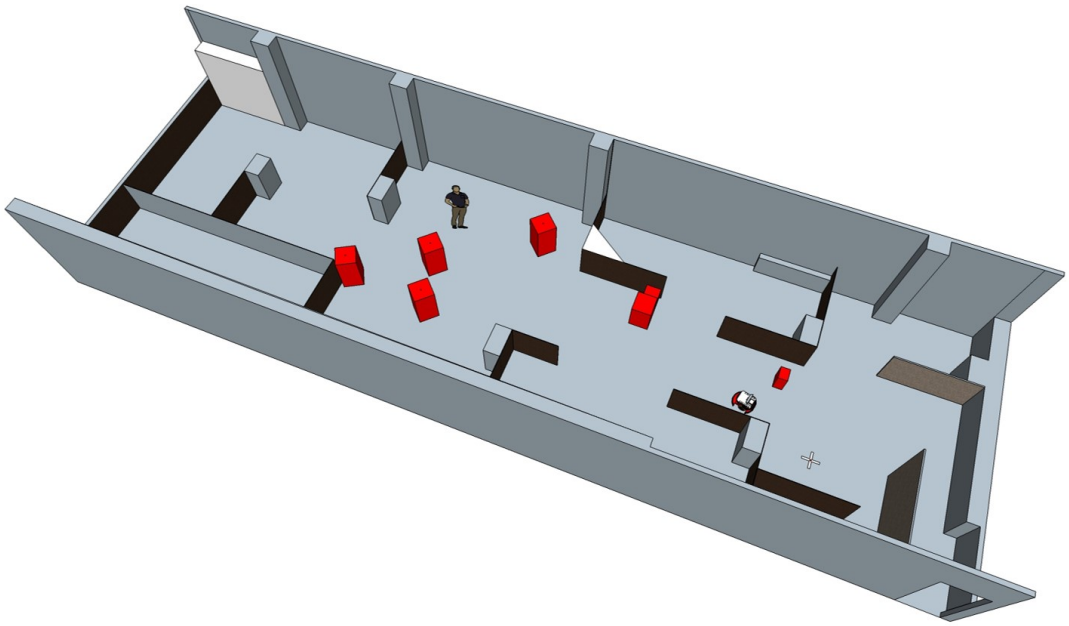
Validation on A Real Environment

Contents

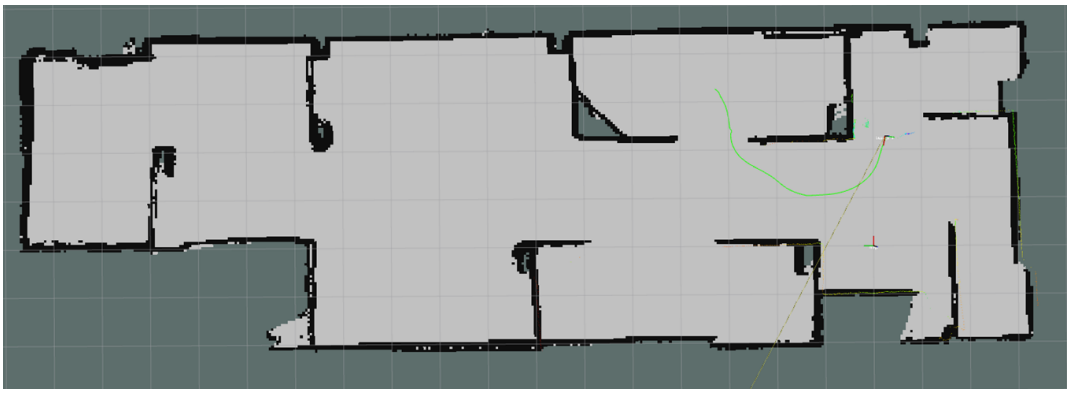
Our proposed methodology for the DQN-based Mission Managers and the Multi-Mission Manager has been implemented and validated for a navigation mission and a video server mission in the simulation framework in the previous chapters. In this chapter, we apply our methodology and implement a prototype for the real mobile robotic system as presented in Section 3.5.1 with all three motivational missions composed of the two above missions and a semantic environment understanding mission (see Section 3.6) in real environmental conditions.

This chapter is organized as follows. Section 6.1 describes the experimentation setup. Our experimentation is then divided into two main parts. The first experimentation in Section 6.2 aims to validate the effectiveness and robustness of our proposed methodology by applying the obtained results of the navigation mission and the video server mission in the simulation framework to the real robotic framework. The second experimentation in Section 6.3 tends to demonstrate the scability of our management framework by proposing a more complex multi-mission context, with the integration of all the three missions mentioned above. Finally, Section 6.4 concludes the chapter.

(A) A corner of our real environment.



(B) Global view of our real environment. The red boxes and people in the environment can be considered as dynamic obstacles for the robot.



(C) Static occupancy grid map of the real environment built by our robot using *move_base* and *gmapping* ROS packages.

FIGURE 6.1: An area of $20.5m \times 6.5m$ used as the real environment for our experimentation.

6.1 Experimentation Setup

6.1.1 Real Environment Setup

Figures 6.1a and 6.1b present our installation for an area of $20.5m \times 6.5m$ as the real environment in the experimentation. Its static occupancy grid map shown in Figure 6.1c is built by using two ROS packages *move_base*[1] and *slam_gmapping*[2]. This static map is then used for the autonomous navigation mission. The structures such as the walls or the bulkheads are kept static and taken into account in the static map. At run-time, we can add some obstacles such as carton boxes or people that can move around in the environment, and we call them as dynamic obstacles. The other conditions of the environment such as lighting or texture can be changed during robot operation. Briefly, the three factors of the environment dynamics as shown in Figure 1.3 can occur.

6.1.2 Robotic System Setup

The real robotic framework is the one presented in Section 3.5.1 with two main sensors (a LIDAR laser scanner and a Kinect sensor) and a NVIDIA Jetson AGX Xavier embedded platform. The Xavier as well as the robot motors and low level controller electronics are powered by the same battery system. The power consumption model of the platform components is also validated with the results measured by a Yocto-Watt wattmeter (Appendix D). We deploy on this platform the three robotic missions presented and characterized in Section 3.6: an autonomous navigation mission, a video server mission and a semantic environment understanding mission. The DQN-based Mission Managers and the Multi-Mission Manager are also deployed on this embedded platform.

We also set up a remote PC station that connects with the Xavier board via a local wifi network as depicted in Figure 6.2. Thanks to the distributed characteristic of the Robot Operating System, the information exchange between the robot and the remote station is convenient with the robot as ROS Master and the remote station

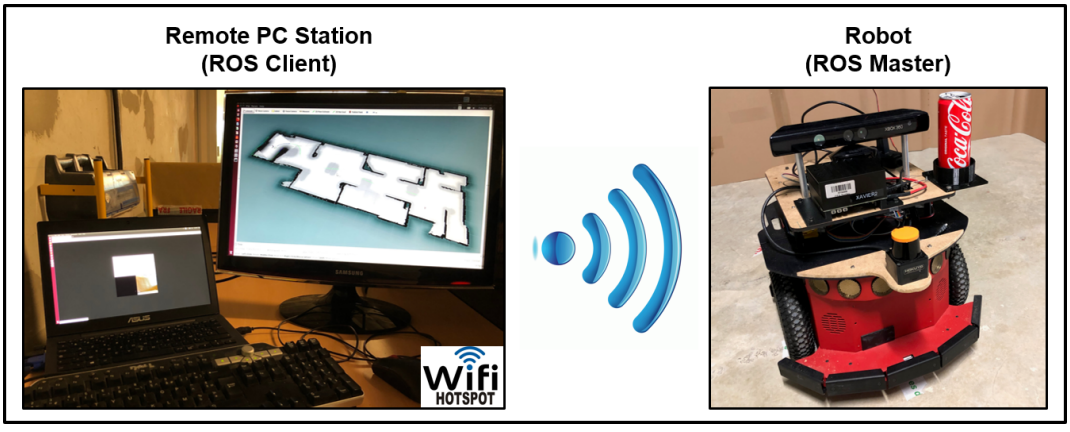


FIGURE 6.2: Robot setup in the real experimentation. The robot is the ROS Master and a remote PC station set as a ROS Client. The connection between two systems is realized via a local Wifi network.

[1]See http://wiki.ros.org/move_base
[2]See http://wiki.ros.org/gmapping

as ROS Client[3]. Indeed, the operation of the robot is independent of the remote PC station.

6.2 From Simulation to Real Experimentation

The objective of this experimentation is to validate the effectiveness and robustness of our proposed DQN-based mission managers and the multi-mission manager by applying the obtained results of the navigation mission and the video server mission in the simulation framework to the real robotic framework. The presentation of this experimentation is organized as follows. Section 6.2.1 introduces the experimentation methodology and followed by the obtained results and discussions in Section 6.2.2. Finally, the efficiency of DQN-based mission managers in terms of power consumption on the embedded platform is also evaluated in Section 6.2.3.

6.2.1 Experimentation Methodology

Reusing Learned Adaptation Models From the Simulation Framework

As in the simulation framework, we assume that the robotic system is in a long time continuous operation. With the same characterization procedure presented in Chapter 3, we propose the non-functional goals for a navigation mission and a video server mission in Tables 6.1 and 6.2. The computing power mode for each mission is also defined in four modes: *High (H)*, *Medium (M)*, *Low (L)* and *Off (O)* (see the interpretation of power modes in the assumption A2 in Section 5.2.1).

TABLE 6.1: Non-functional goals for Navigation Mission in the real experimentation.

NFRs	Navigation Mission
G_{perf_1}	$V_{mean\_min} = 0.5m/s$
G_{energy_1}	$P_{s\&a\_mean\_max} = 17.0W$
G_{energy_2}	$P_{c\_mean\_max} = 1.25W(H) - 1.1W(M) - 0.95W(L) - 0W(O)$
G_{others_1}	$D_{obs\_min} = 0.7m$

TABLE 6.2: Non-functional goals for Server Mission in the real experimentation.

NFRs	Server Mission
G_{perf_1}	$RES_{mean\_min} = 800.0pixels$
G_{energy_1}	
G_{energy_2}	$P_{c\_mean\_max} = 3.25W(H) - 3.0W(M) - 2.75W(L) - 0W(O)$
G_{others_1}	$FPS_{mean\_min} = 24.5Hz$

Chapters 4 and 5 have introduced and validated our methodology in the simulation framework for these two robotic missions. We leverage the obtained learning results in simulation (as depicted in Figure 5.8) to apply them to these missions in the real robotic framework with a 1-1 mapping presented in Figure 6.3. The considered QoS in the case of the navigation mission is the mean robot speed (in m/s) whereas the considered QoS of the video server mission is the mean resolution of the encoded images (in *pixels*). Thus, we do not need a pure learning phase as in simulation, we

[3]See http://wiki.ros.org/ROS/Tutorials/MultipleMachines

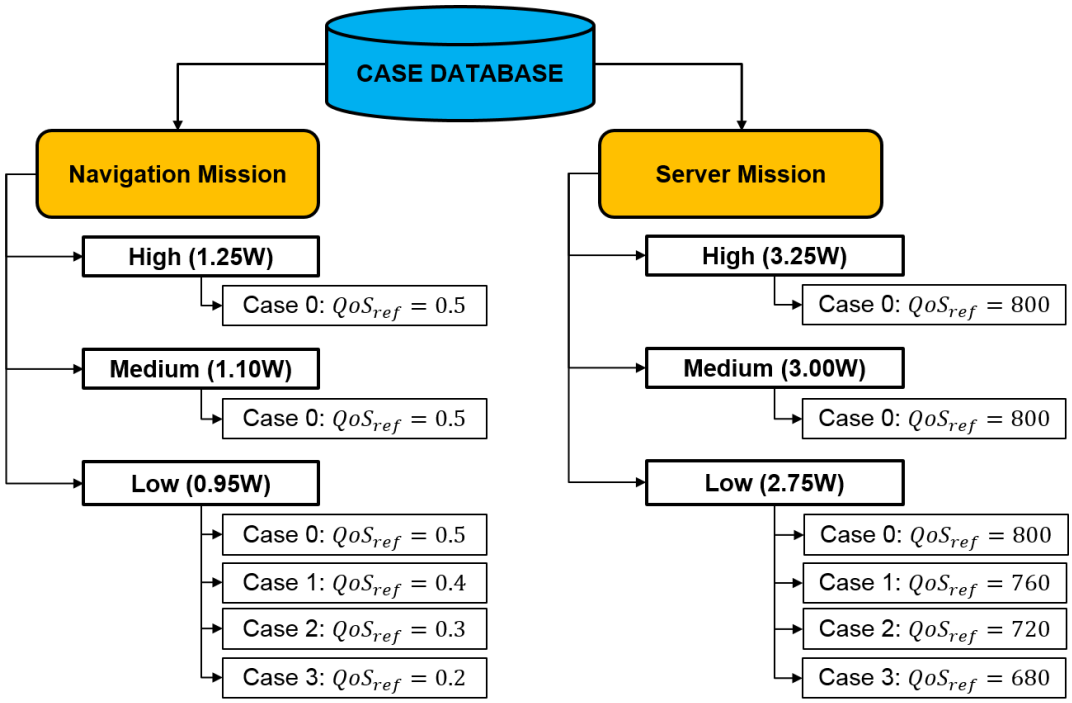


FIGURE 6.3: An initialization of the case database for the navigation mission and the server mission in the real experimentation. The learned adaptation models in the simulation framework are reused in the real experimentation.

use the pre-learned models of the mission managers and we add the online learning capability to help evolving the adaptation policy over the deployment time in case the models do not fit the new contexts.

Moreover, the online learning capability has been activated in our real experimentation, which depends on the level of NFRs violation to refine and evolve the adaptation policy so that more unexpected dynamic real-world scenarios can be taken into account (see Section 5.4.3). The level of NFRs violation is monitored within $m = 3$ successive episodes (this value is empirically choosen as mentioned in the simulated scenario in Section 5.5) and if it is less than or equal to a certain threshold (set in this case to -0.05), the online learning will be activated. The three conditions such as new computing power mode, new case of QoS and end of timeout (see Section 5.4.3 for more details about these criterias) are also applied to deactivate the online learning. The timeout determines that the online learning must be stopped after a fixed number of episodes that we assume sufficient (after conducting the experimentation) to adapt and enhance the adaptation policy. This timeout is set to 60 episodes (equivalent to 60 minutes).

Robotic Autonomy Under Real Battery Constraints

Taking into account the real condition of the robot's battery, the robot can be autonomous for 3 or 4 hours and we should recharge the battery if the battery voltage is lower than $11.5V$ (Appendix B). Thus, we will monitor the battery voltage. If the low battery threshold is reached, we will force the robot to return to the docking station (the root of the map in our scenario) to be recharged. Currently, we lack an autonomous recharging solution for our robotic platform that requires a manual recharging and limits the flexibility of the deployment.

Based on the current battery voltage, we also establish a rule between the battery voltage and the maximum power capacity allowed for all the robotic missions (see the assumption A1 in Section 5.2.1). The robot operates under the battery voltage between $12.5V$ and $11.5V$. Thus, we propose to divide the battery into 5 states as shown in Table 6.3 depending on the battery voltage. Consequently, the maximum power capacity is also defined at 5 levels. The maximum power capacity for the scenarios of a navigation mission and a video server mission is defined in Table 6.4.

TABLE 6.3: Real battery voltage conditions divided into 5 states.

State	**0**	**1**	**2**	**3**	**4**
Voltage	$> 12.3V$	$> 12.1V$	$> 11.9V$	$> 11.7V$	Others

TABLE 6.4: Maximum power capacity vs. battery state in case of two missions: a navigation mission and a video server mission.

Battery State	**0**	**1**	**2**	**3**	**4**
$P^{\sum}_{c\_max}$	$4.5W$	$4.25W$	$4.00W$	$3.85W$	$3.70W$

Experimental Scenarios and Expectations

We deploy a navigation mission denoted as $M1$ and a video server mission denoted as $M2$ at the same time and apply our management methodology such as DQN-based Mission Managers and Multi-Mission Manager to the robotic system. In this experimentation part, we propose to deploy five scenarios. The objective of this proposition is to demonstrate different functionalities of the Multi-Mission Manager as well as the adaptation capability of the Mission Managers. The five experimental scenarios and their objectives are described as follows:

- **Scenario 1: Static Mission Priorities**. In this scenario, we deploy a navigation mission and a video server mission at the same time. Thus, we have a multi-mission context of two missions. We assume also in this scenario that the priority of the navigation mission is always higher than the one of the video server mission. The dynamic system-level constraint such as the maximum power capacity as shown in Table 6.4 is also applied. The objective of this scenario is to demonstrate the dynamic characteristic of robotic operational conditions and the adaptive management of our mission managers and multi-mission manager;

- **Scenario 2: Dynamic Mission Priorities**. In this case, we demonstrate a scenario where the mission's priorities are dynamic. At some moments during operation, we will exchange the priority level between the navigation mission and the video server mission. This scenario will demonstrate the way how the multi-mission manager copes with the dynamic mission's priorities in a multi-mission context;

- **Scenario 3: Power Budget Are Not Enough For Two Missions**. With an assumption that the maximum power capacity is not enough to operate both two missions, we aim to demonstrate the adaptive decision of the multi-mission manager to guarantee this modest budget;

- **Scenario 4: Variability Management of A Video Server Mission**. This is a scenario specific to the video server mission. The objective is to assess the performance of our proposed run-time adaptation in comparison with a static configuration;
- **Scenario 5: Run-Time Adaptation vs. Static Configuration**. In this case, we strive to compare the obtained performance of the two robotic missions as well as the autonomy duration or the battery duration between our run-time adaptation and the static configurations of the missions.

The next section will present the obtained results of these scenarios and we will also provide some insightful discussions.

6.2.2 Experimentation Results

The implementation details and results of each experimental scenario will be presented in this section. We provide below the description of the measures or the metrics that will be used to evaluate the experimentation. These measures are classified into two main groups: system-level measures and mission-level measures.

- **System-level measures/metrics**:
  - *Battery voltage*: the current robot's battery voltage given by the *RosAria* ROS node (Appendix B);
  - *Total power budget*: the maximum computing power capacity reserved for all the robotic missions that are currently active on the robotic system. It is defined depending on the battery voltage;
  - *Total consumed power*: the total computing power consumption of all the robotic missions.
- **Mission-level measures/metrics**:
  - *Ref. Power*: the computing power budget (reference value) reserved to the mission. This metric is regulated by the decision of reallocating the budget of the multi-mission manager;
  - *Mea. Power*: the computing power consumption (measured value) of the mission;
  - *QoS*: the quality of service of the mission, mean robot velocity for the navigation mission, and mean resolution of the encoded images for the video server mission;
  - *Ref. QoS*: the desired quality of service (reference value) applied to the mission. This metric is regulated by the decision of throttling QoS of the multi-mission manager;
  - *Mea. QoS*: the measured quality of service of the mission;
  - *Level of NFRs violation*: the level of non-functional requirements violation;
  - *Violation threshold*: the level of NFRs violation is compared with this threshold to check the condition of activating the online learning. It is a parameter of the multi-mission manager;
  - *Triggering online learning*: the metric indicating whether the online learning is activated (1) or not (0).

In general, in the following experimentation, the reference (or objective) values (for QoS or Power) will be depicted in red in the figures and the actual measured values will be in green.

Scenario 1: Static Mission Priorities

- *Hypothesis:* The two robotic missions, a navigation mission and a video server mission, are deployed in the same time, and we assume that the priority of the navigation mission is higher than the one of the video server mission and these priorities are kept unchanged during operation. In this scenario, we deploy three different runs at the three different times and analyze their obtained results. The duration of each run is counted from the beginning of the robot operation until the battery recharging requirement. Hence, the deployment is strongly dependent on the charging level of the robot's batteries. Each deployment is also subject to different operational conditions at random. For example, we change randomly the appearance of the dynamic obstacles in the environment. Thus, we can realize the different behaviors of the mission managers and the multi-mission manager in these three runs.
- *Analysis:* The results of the three different runs are sequentially presented in Figures 6.4, 6.5 and 6.6. The system-level and mission-level characteristics of each run are presented in the following paragraphs.

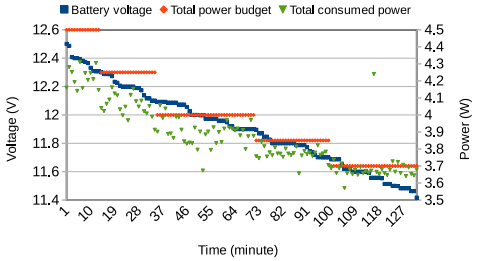
Analysis of Run 1. The deployment results of this run are presented in Figure 6.4 with the system-level measures and the mission-level measures of the two missions. The dynamic system-level constraint and the computing power budget allocated dynamically to each robotic mission during operation are shown in Table 6.5. The autonomy duration of this run is 132 minutes until the battery recharging.

TABLE 6.5: Run 1: dynamic system-level constraint and computing power budget allocated dynamically to each robotic mission.

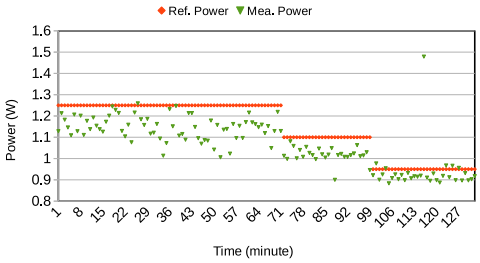
Time t	**1 - 13**	**14 - 34**	**35 - 71**	**72 - 99**	**100 - 132**
$Voltage(t)$	$> 12.3V$	$> 12.1V$	$> 11.9V$	$> 11.7V$	Others
$P^{\sum}_{c\_max}(t)$	$4.5W$	$4.25W$	$4.00W$	$3.85W$	$3.70W$
$P^{M1}_{c\_ref}(t)$	$H, 1.25W$	$H, 1.25W$	$H, 1.25W$	$M, 1.10W$	$L, 0.95W$
$P^{M2}_{c\_ref}(t)$	$H, 3.25W$	$M, 3.00W$	$L, 2.75W$	$L, 2.75W$	$L, 2.75W$

We can realize that for each robotic mission, under different computing power budgets, the obtained quality of service is also different. For example, the QoS of the navigation mission can achieve $0.6m/s$ in the high and medium power modes (Figure 6.4d), and the QoS of the server mission can achieve $840pixels$ (Figure 6.4e), but they are significantly reduced in the low modes. The desired QoS of the navigation mission is not throttled in this run because the condition of $m = 3$ successive episodes of computing power consumption objective violation is not fulfilled. Whereas, the desired QoS of the server mission is throttled to guarantee the low computing power budget. For example, when the power mode of the server mission changes from high, medium to low mode (from the episode 35), keeping the desired QoS of 800 leads to a computing power consumption surpassing the power budget of $2.75W$. Thus, the desired QoS is reduced according to the equation 5.8. Inversely, if the computing power consumption remains lower than the power budget for $m = 3$ successive episodes, the desired QoS can be increased to give a better QoS level (for example at episode 57). Figures 6.4b and 6.4c indicate that in almost all the episodes, the computing power consumption of each mission follows well the

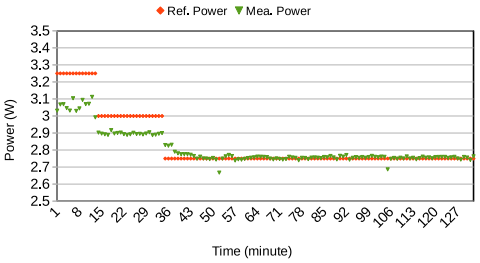
reference value. Consequently, the total power consumption is guaranteed and do not surpass the total power budget (Figure 6.4a). By observing Figures 6.4f and 6.4g, we can see that the level of NFRs violation is always greater than the negative violation threshold that we set and the online learning is therefore not activated during the operating time, except one triggering for the navigation mission at the end of the run (episode 130) due to consecutive violations. The learning was stopped because of the end of the robot autonomy.



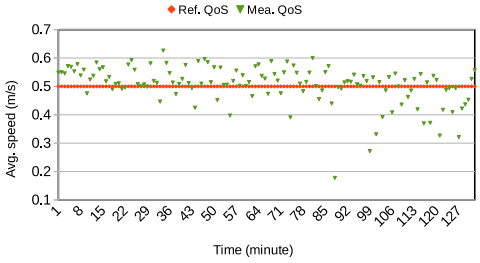
(A) System-level computing power consumption controlled by the Multi-Mission Manager



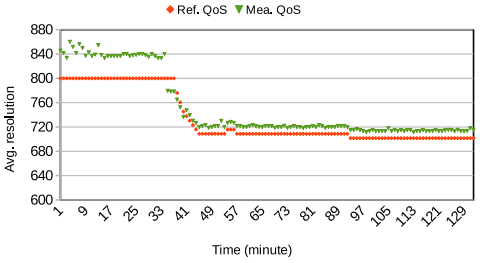
(B) Computing power consumption of Navigation Mission.



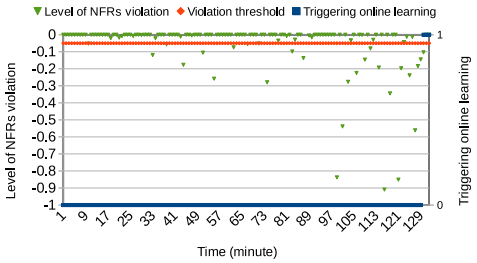
(C) Computing power consumption of Server Mission.



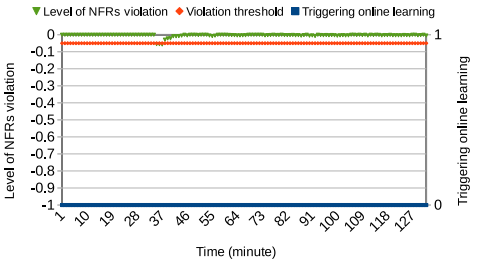
(D) Quality of service of Navigation Mission.



(E) Quality of service of Server Mission.



(F) NFRs violation and online learning triggering of Navigation Mission.



(G) NFRs violation and online learning triggering of Server Mission.

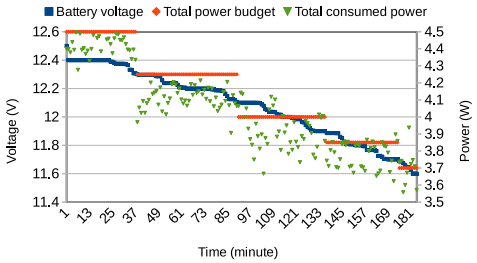
FIGURE 6.4: Run 1: computing power consumption, quality of service and online learning of each mission controlled by the Multi-Mission Manager.

Analysis of Run 2. The deployment results of this run are presented in Figure 6.5, and the dynamic system-level constraint and the computing power budget allocated to each robotic mission during operation are shown in Table 6.6. The autonomy duration of this run is 184 minutes.

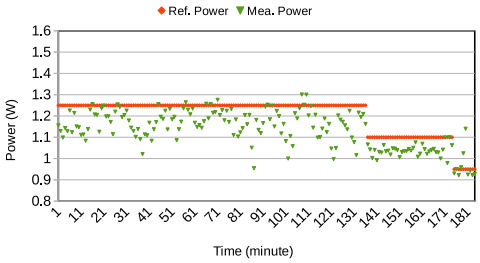
TABLE 6.6: Run 2: dynamic system-level constraint and computing power budget allocated dynamically to each robotic mission.

Time t	**1 - 37**	**38 - 90**	**91 - 136**	**137 - 174**	**175 - 184**
$Voltage(t)$	$> 12.3V$	$> 12.1V$	$> 11.9V$	$> 11.7V$	Others
$P^{\Sigma}_{c\_max}(t)$	$4.5W$	$4.25W$	$4.00W$	$3.85W$	$3.70W$
$P^{M1}_{c\_ref}(t)$	$H, 1.25W$	$H, 1.25W$	$H, 1.25W$	$M, 1.10W$	$L, 0.95W$
$P^{M2}_{c\_ref}(t)$	$H, 3.25W$	$M, 3.00W$	$L, 2.75W$	$L, 2.75W$	$L, 2.75W$

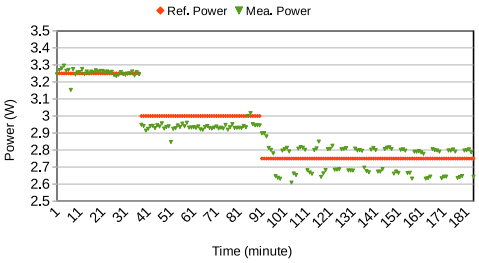
For the navigation mission, Figures 6.5d and 6.5b indicate the same characteristics of quality of service and computing power consumption as the run 1 in the high and medium power modes. At episode 111, we can see in (Figure 6.5d) that the multi-mission manager decides to throttle the QoS from $0.5m/s$ to $0.48m/s$ reacting to a power reference violation for three consecutive episodes (see figure 6.5b) and then at episode 114 it decides to raise back the QoS to $0.5m/s$. In the low power mode, the desired QoS of this mission is reduced to $0.45m/s$ from the episode 179 to the end of the navigation mission (Figure 6.5d). For the video server mission, in the low power mode, we can realize that the desired quality of service is regularly throttled and the computing power consumption is very sensitive to the changing of the desired QoS. Once the computing power consumption in $m = 3$ successive episodes surpasses the budget, the desired QoS is reduced to a lower value. With this new QoS reference value, the power consumption of the mission returns to be less than the budget and this leads to a new higher desired QoS for the next decision epoch (see decision-making of throttling QoS in Section 5.4.2 for more details). Thus, the characteristic of computing power consumption and quality of service of the video server mission in this low power mode is very oscillatory (Figures 6.5e and 6.5c). In fact, the decision of throttling the quality of service can be considered reactive to respect the constraint of computing power budget. However, in some cases, this reactive decision can lead to an oscillating characteristic as we realized. The choice of m and $\Delta P^{Mi,avg}_{c}(t)$ as mentioned in Section 5.4.2 can be a reason for this oscillation and we need a further study to find the optimal calculation method. At the system level, the total power consumption is maintained to be less than or equal to the total budget in almost all the episodes of the operating time (Figure 6.5a). In this run, the condition of activating the online learning is triggered for the navigation mission at the interval between episodes 88 and 178 (Figure 6.5f). The condition of a timeout of 60 minutes has been triggered to stop the first online learning, while the changing of power mode from medium to low mode stops the second online learning that started at the episode 178. For the video server mission, the level of NFRs violation is less than the violation threshold during many times (Figure 6.5g). However, the online learning is not triggered because of regularly throttling the desired quality of service.



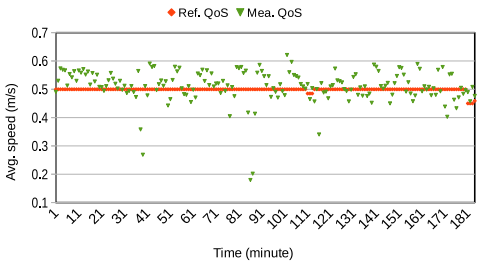
(A) System-level computing power consumption controlled by the Multi-Mission Manager



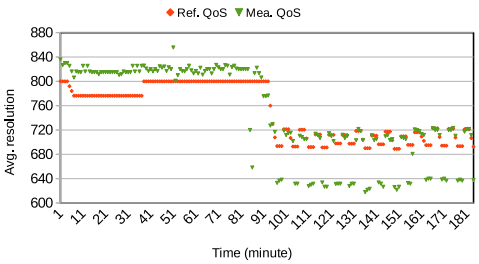
(B) Computing power consumption of Navigation Mission.



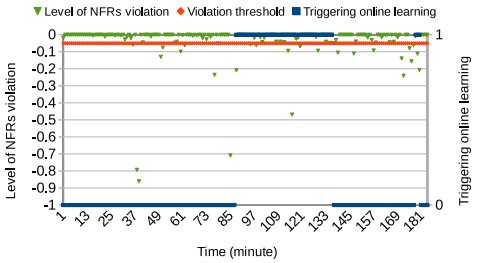
(C) Computing power consumption of Server Mission.



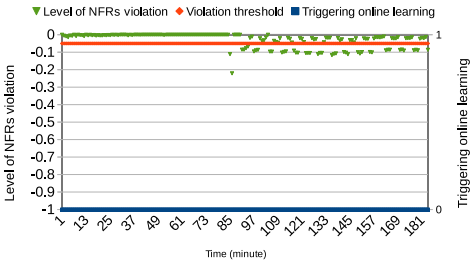
(D) Quality of service of Navigation Mission.



(E) Quality of service of Server Mission.



(F) NFRs violation and online learning triggering of Navigation Mission.



(G) NFRs violation and online learning triggering of Server Mission.

FIGURE 6.5: Run 2: computing power consumption, quality of service and online learning of each mission controlled by the Multi-Mission Manager.

Analysis of Run 3. The deployment results of this run are presented in Figure 6.6, and the dynamic system-level constraint and the computing power budget allocated to each robotic mission during operation are shown in Table 6.7. The autonomy duration of this run is 199 minutes.

TABLE 6.7: Run 3: dynamic system-level constraint and computing power budget allocated dynamically to each robotic mission.

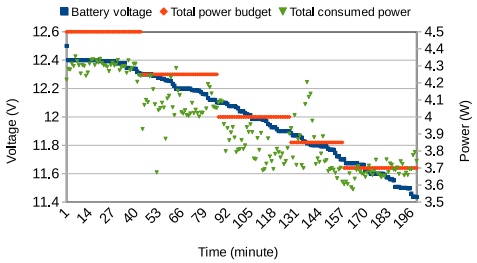
Time t	**1 - 43**	**44 - 86**	**87 - 127**	**128 - 157**	**158 - 199**
$Voltage(t)$	$> 12.3V$	$> 12.1V$	$> 11.9V$	$> 11.7V$	Others
$P^{\Sigma}_{c\_max}(t)$	$4.5W$	$4.25W$	$4.00W$	$3.85W$	$3.70W$
$P^{M1}_{c\_ref}(t)$	$H, 1.25W$	$H, 1.25W$	$H, 1.25W$	$M, 1.10W$	$L, 0.95W$
$P^{M2}_{c\_ref}(t)$	$H, 3.25W$	$M, 3.00W$	$L, 2.75W$	$L, 2.75W$	$L, 2.75W$

The characteristics of the navigation mission (Figures 6.6d, 6.6b and 6.6f) are analyzed the same way as in the runs 1 and 2. Of course, the different environmental conditions such as dynamic obstacles in the three runs lead to different behaviors of the navigation mission. For the video server mission (Figures 6.6e, 6.6c and 6.6g), since the moment of 45 minutes, the user's visualization request[4] has been changed as presented in Table 6.8 and this makes the learned adaptation model difficult to adapt. Thus, the combination of throttling the desired QoS and online learning makes the adaptation more feasible and stable from the episode 162. More discussions specific to the video server mission will be presented in Scenario 4. The system-level measures are depicted in Figure 6.6a. We can see that the total power consumption surpasses the total budget on the time interval between episodes 128 and 141 because of the difficulty of managing the computing power consumption of the video server mission.

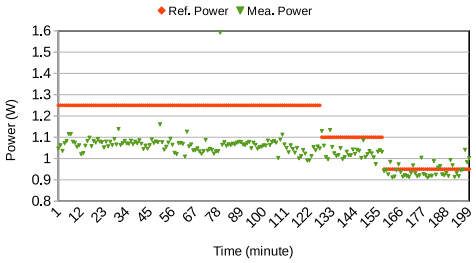
TABLE 6.8: Variability in user requests of the video server mission.

Time t	**1 - 44**	**45 - 199**
User requests	1 of (512×512)	1 of (800×800)

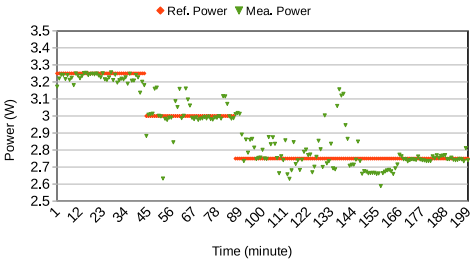
[4]See image parameters (width x height) on `http://wiki.ros.org/web_video_server`



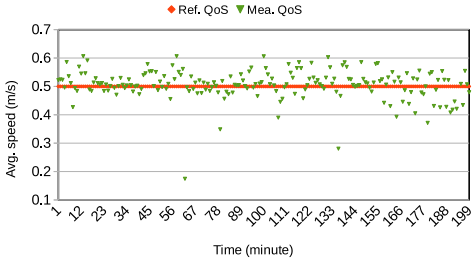
(A) System-level computing power consumption controlled by the Multi-Mission Manager



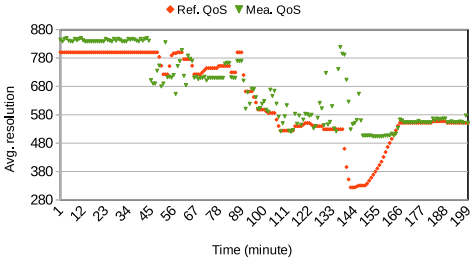
(B) Computing power consumption of Navigation Mission.



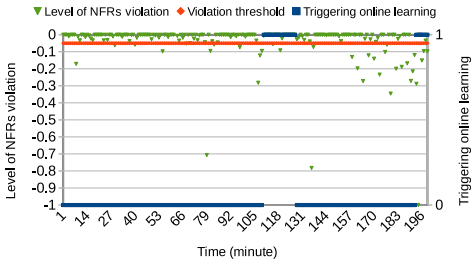
(C) Computing power consumption of Server Mission.



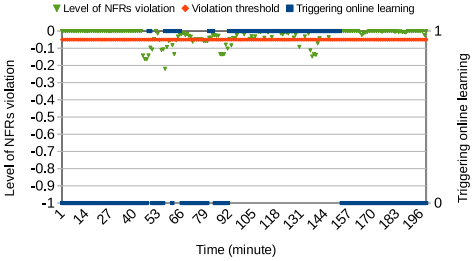
(D) Quality of service of Navigation Mission.



(E) Quality of service of Server Mission.



(F) NFRs violation and online learning triggering of Navigation Mission.



(G) NFRs violation and online learning triggering of Server Mission.

FIGURE 6.6: Run 3: computing power consumption, quality of service and online learning of each mission controlled by the Multi-Mission Manager.

- *Conclusion:* Firstly, we can realize that the results obtained in the three different runs of this scenario are consistent with the results obtained in simulation (see Section 5.5), which indicate the robustness of the methodology that we propose. Secondly, the three decisions of the Multi-Mission Manager (reallocating the computing power budget, throttling the desired QoS and triggering the online learning) are validated in our real experimentation. If the first two decisions to reallocate the mission's computing power budget and to regulate the QoS can be considered reactive to the system-level constraint (the maximum computing power capacity in this case), so that this constraint is strictly guaranteed, the third decision to trigger the online learning is considered as a catalyst to evolve the adaptation policy in a long-term operation. It is not obvious to see the good performance of the online learning after a certain learning time, but it needs an evaluating methodology for a long-term operation.

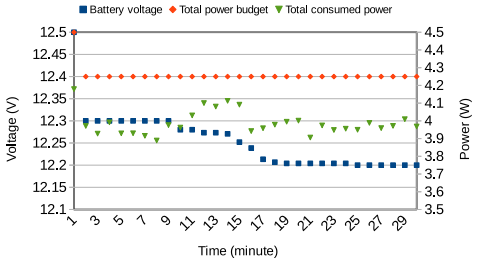
Scenario 2: Dynamic Mission Priorities

- *Hypothesis:* While the three runs in the first scenario are deployed with static mission priorities where the priority of the navigation mission is always higher than that of the server mission, we now present a scenario of dynamic mission priorities where the user reacts at run-time with the system by explicitly modifying them as presented in Table 6.9.

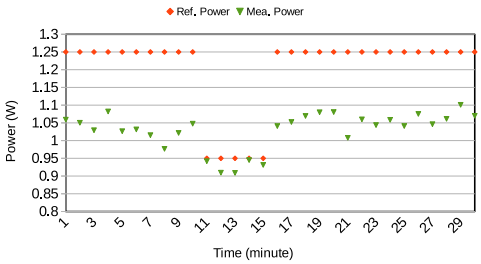
TABLE 6.9: Dynamic mission's priorities during operation.

Time t	**1 - 10**	**11 - 15**	**16 - 30**
Priority of navigation	1	0	1
Priority of server	0	1	0

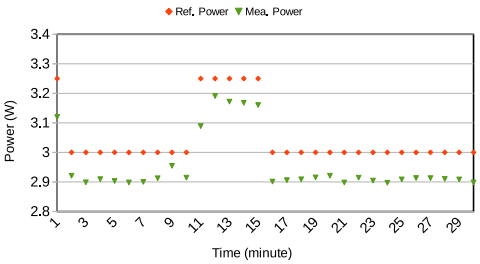
- *Analysis:* In this scenario of 30 minutes (Figure 6.7), the navigation mission starts with the highest priority and then the priorities of the two missions are reversed in the interval from 11 to 15 minutes. From 16 minutes on, the priorities of the two missions return to their initial levels. We can see that our multi-mission manager dynamically takes this change into account to reallocate the appropriate computing power budget for each mission. The decision of the multi-mission manager must guarantee that the mission with a higher priority will have a higher computing power consumption mode and that the constraint in Equation 5.1 is satisfied. Thus, from 11 to 15 minutes, the video server mission is operating at the high computing power mode while the power mode of the navigation mission is reduced to the low mode. We can also see that the characteristic of the computing power consumption of each mission is controlled to respect the reserved power budget.
- *Conclusion:* In a multi-mission context, the priority level of each mission can be different and it can even be changed during operation by some internal or external events. Our multi-mission manager has taken into account the dynamic priorities of the missions to reallocate the appropriate power budget for each mission and then contribute to the guarantee of the system-level constraints: the maximum computing power capacity in this case.



(A) System-level computing power consumption controlled by the Multi-Mission Manager



(B) Computing power consumption of Navigation Mission.

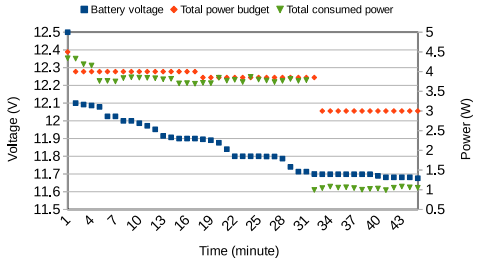


(C) Computing power consumption of Server Mission.

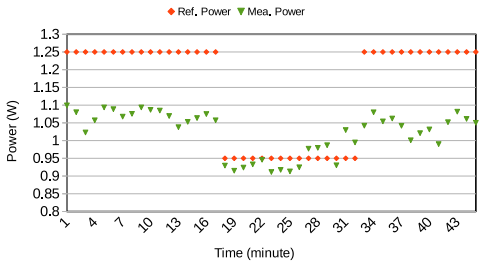
FIGURE 6.7: Multi-Mission Management in a scenario of dynamic mission priorities. The priority of the navigation mission is higher than that of the server mission from 1 to 10 minutes. Then, from 11 to 15 minutes, the mission priorities are reversed and the computing power budget for each mission is therefore reallocated. Similarly, from 16 minutes to the end, the mission priorities are changed and the power budget is also reallocated for each mission.

Scenario 3: Power Budget Are Not Enough For Two Missions

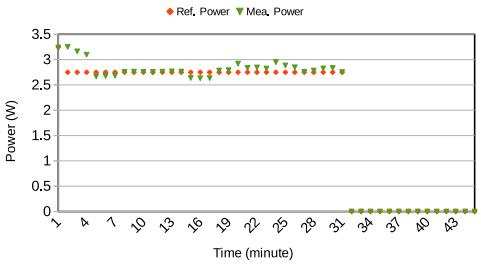
- *Hypothesis:* In this scenario, we would like to demonstrate that our multi-mission manager can handle extremely low computing power budget even by turning some missions to non-active modes. To do that, we assume that the total computing power budget that can be shared between the robotic missions when the battery voltage level reaches $11.7V$ (Table 6.3) is set to $3W$ and the priority of the navigation mission is higher than the one of the video server mission.
- *Analysis:* The results of this scenario are depicted in Figure 6.8. We can see that the two missions are active within the interval between 1 and 31 minutes and the computing power budget for each mission is reallocated by the decision of the multi-mission manager depending on the total power budget. From the episode 32, the total power budget of $3W$ is not enough to deploy the two missions, even in their low modes. In this case, the multi-mission manager decides to shutdown the video server mission according to the equation 5.1 and to reallocate the entire power budget to the navigation mission that changes its computing power mode to the high mode.
- *Conclusion:* This scenario demonstrates the capability of our multi-mission manager to be aware of the system-level constraints as well as the priority level of each mission in the multi-mission context to prioritize the more important mission while respecting the constraint.



(A) System-level computing power consumption controlled by the Multi-Mission Manager



(B) Computing power consumption of Navigation Mission.



(C) Computing power consumption of Server Mission.

FIGURE 6.8: Multi-Mission Manager in a scenario of a deactivated mission. When the battery voltage is greater than $11.7V$, two missions are in normal operation. When the battery voltage is less than or equal to $11.7V$ from 32 minutes, the power budget of $3.00W$ is not enough for two missions. Thus, the server mission is deactivated by the multi-mission manager and the navigation mission returns to the high computing power mode because the priority of the navigation mission is higher than that of the server mission.

Scenario 4: Variability Management of A Video Server Mission

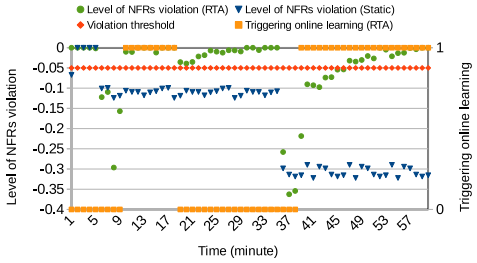
- *Hypothesis:* We aim to evaluate the performance of our run-time adaptation framework to manage the variability of the robotic mission, a video server mission in this case, in comparison to the static configuration. In this scenario, a video server mission is deployed for 60 minutes. From 1 to 5 minutes, there is only one visualization request of a resolution 512×512[5]. From 6 to 35 minutes, there are two requests of a resolution 512×512. From 36 to 60 minutes, there are also two requests, but one of resolution 512×512 and one of resolution 1024×1024. These variabilities are summarized in Table 6.10.

TABLE 6.10: Variability in user requests of a video server mission.

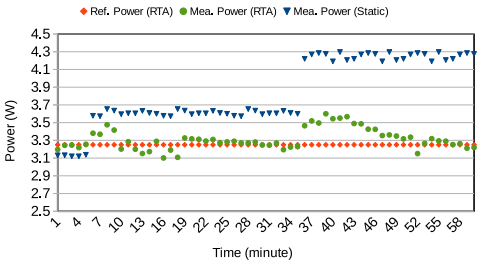
Time t	**1 - 5**	**6 - 35**	**36 - 60**
User requests	1 of (512×512)	2 of (512×512)	1 of (512×512), 1 of (1024×1024)

The static configuration here is defined by $crop\_size = 400$ and our run-time adaptation includes the video server mission manager and the multi-mission manager with the non-functional objectives determined in the high computing power

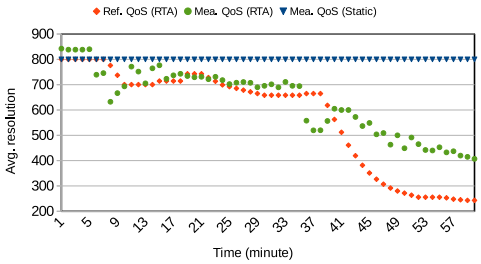
[5]See image parameters (width x height) on `http://wiki.ros.org/web_video_server`



(A) Level of NFRs violation and online learning



(B) Computing power consumption.



(C) Quality of service.

FIGURE 6.9: A comparison between run-time adaptation and static configuration of the video server mission. From 1 to 5 minutes: 1 visualization request of 512×512. From 6 to 35 minutes: 2 visualization requests of 512×512. From 36 to 60 minutes: 2 visualization requests, one of 512×512 and another of 1024×1024.

mode as described in Table 6.2.
- *Analysis:* Figure 6.9 shows the results of the static configuration and the run-time adaptation. With the static configuration, the quality of service as average resolution is always 800, however the power consumption increases from $3.10W$ to $3.60W$ and $4.30W$ corresponding to the three mentioned changes of the mission. Our RTA takes these changes into account while respecting the non-functional objectives to regulate the desired level of QoS (Figure 6.9c) and/or to trigger the online learning to minimize the level of NFR violations (Figure 6.9a). We also see that the mission's power consumption is controlled to respect the high computing power mode of $3.25W$ (Figure 6.9b).
- *Conclusion:* This scenario highlights again the advantage of our run-time adaptation methodology by trading off the computing power consumption and the quality of service. Under different power budgets, the desired quality of service of the robotic mission should be regulated to guarantee that the consumption does not surpass the budget. In the case where the power budget is defined as a strict constraint, using the run-time adaptation is indispensable.

Scenario 5: Run-Time Adaptation vs. Static Configuration

- *Hypothesis:* In this scenario, we aim to compare our proposed run-time adaptation (RTA) methodology including DQN-based mission managers and the adaptive hierarchical multi-mission manager (the same implementation as in the scenario 1) with two static configurations of the two robotic missions. Indeed, these static configurations are chosen in Section 4.8.5 while validating the DQN-based mission managers, a configuration of $\{0.8m/s, 10Hz\}$ for the navigation mission and a configuration

of $\{400pixels\}$ for the video server mission. For each case, static or RTA, we strive to keep the same operational conditions such as dynamic obstacles, initial battery capacity, etc. In each case, the two missions are also deployed in parallel and the robotic system is operating until the battery recharging. No variability in the user requests for the video server mission has been made. We will evaluate the autonomy duration of each case (static and RTA) as well as the obtained performance of the two robotic missions.

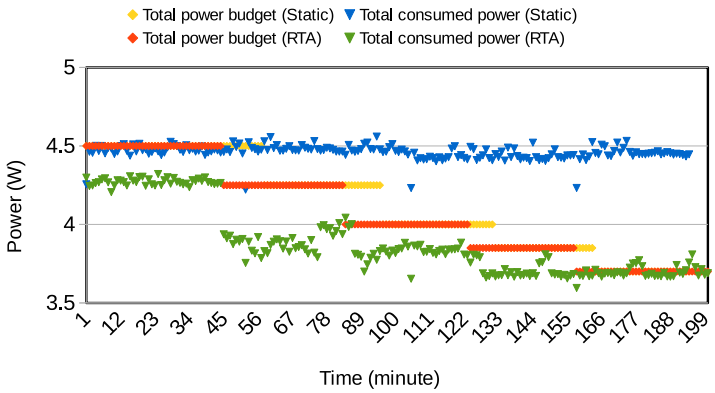


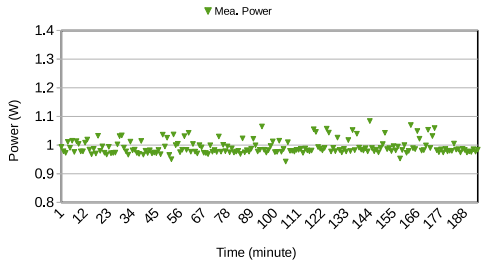
FIGURE 6.10: System-level computing power consumption.

- *Analysis:* Figure 6.10 indicates the autonomy duration and the total computing power consumption of each case. Indeed, the autonomy duration of the static case is 194 minutes while the one of the RTA case is a bit longer with 200 minutes. In terms of the total computing power consumption of each case, we can see two different characteristics: a fairly constant line (in blue) for the static case and a controlled line (in green) respecting the total power budget for the RTA case. In fact, the total power budget of the two cases is defined as the rule in Table 6.4. If the total power budget is considered as a hard constraint, our proposed RTA guarantees well this constraint, while the static configurations may never achieve this objective. As we can see in terms of total consumed power, our RTA methodology can achieve a maximum gain of $4.5W/3.7W \approx 1.22$ in comparison with the static configurations.

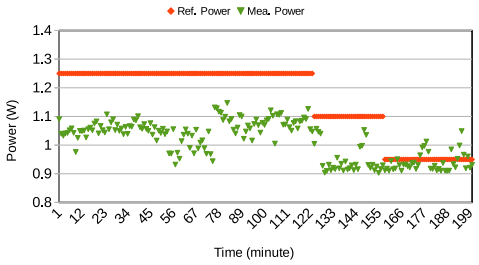
Figure 6.11 presents the computing power consumption and the quality of service of the navigation mission in the two cases, static and RTA. For the static case, we can see that the computing power consumption is fairly constant and the obtained quality of service is mostly higher than the desired value. For the RTA case, the characteristic of the power consumption and the quality of service is controlled to minimize the violation of non-functional requirements. It is also important to note that in this scenario we set only a simple configuration of dynamic obstacles in the environment, the static case therefore can give the stable characteristic of power and QoS. If we choose another configuration of dynamic obstacles, the obtained results of the static case will be different. Hence, a controlled characteristic respecting non-functional requirements given by our run-time adaptation will be appreciated.

Similar to the navigation mission, the characteristic of the computing power consumption and the quality of service of the video server mission is shown in Figure 6.12. The constant characteristic is also found for the static case and the RTA case gives the controlled characteristic to guarantee the changing of reserved computing power budget.

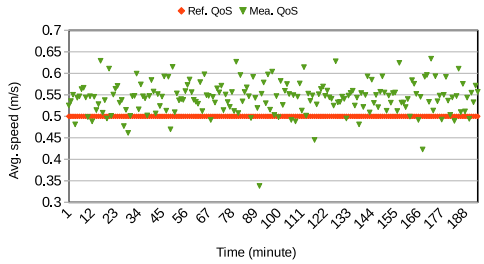
- *Conclusion:* For concluding this scenario, we can realize that the static configuration can give the better characteristic in some situations, but the controlled characteristic



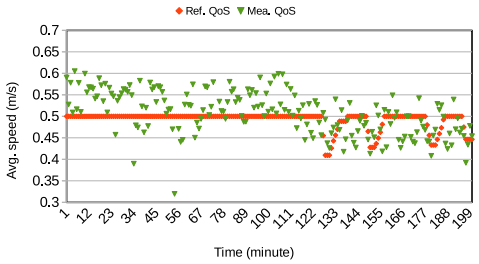
(A) Computing power consumption (Static).



(B) Computing power consumption (RTA).

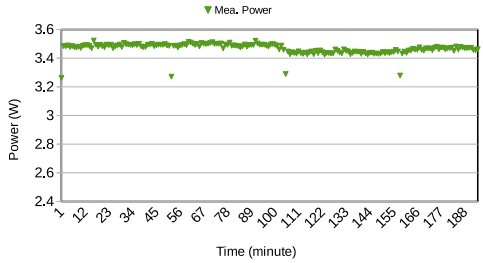


(C) Quality of service (Static).

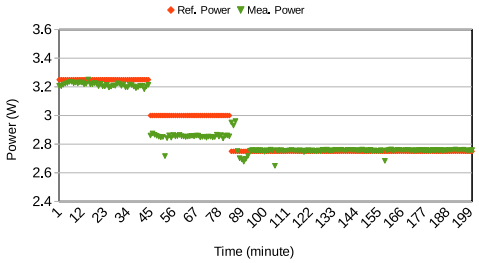


(D) Quality of service (RTA).

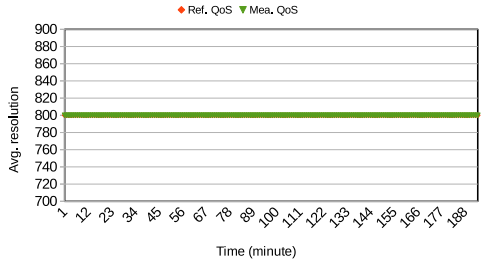
FIGURE 6.11: Computing power consumption and quality of service of the navigation mission



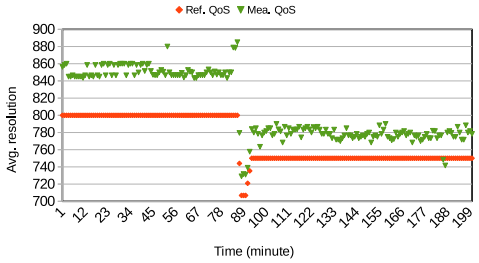
(A) Computing power consumption of (Static).



(B) Computing power consumption (RTA).



(C) Quality of service (Static).



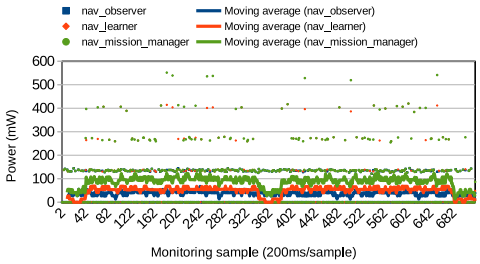
(D) Quality of service (RTA).

FIGURE 6.12: Computing power consumption and quality of service of the video server mission

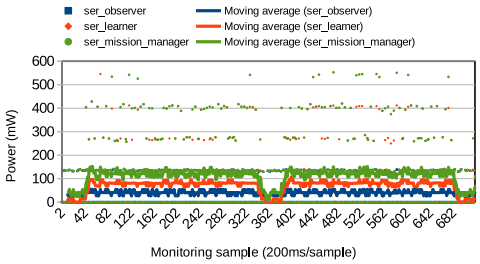
given by our run-time adaptation framework is necessary to react to more dynamic operational conditions while guaranteeing the non-functional requirements applied to the robotic missions.

6.2.3 Efficiency of Mission Managers and Multi-Mission Manager

Figures 6.13 and 6.14 characterize the power consumption of the DQN-based mission managers in the learning and planning phase in the real NVIDIA Jetson Xavier embedded platform. In the learning phase, an average of 100mW and 120mW is the corresponding power consumption of the navigation and server mission managers. These values are respectively reduced to 70mW and 75mW in the planning phase of the two mission managers without Q-Network updating process. In fact, the power consumption is depending on the monitoring and adaptation period. The monitoring period of the two missions is chosen as 200ms, so we can see that the same power consumption is found for the two missions both in the learning and planning phase. The adaptation period of the navigation mission is 2s and the one of the video server mission is 1s. Hence, we can realize that the power consumption of the video

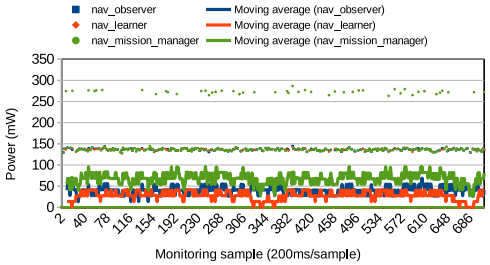


(A) Power consumption of Navigation Mission Manager: 100mW in average.

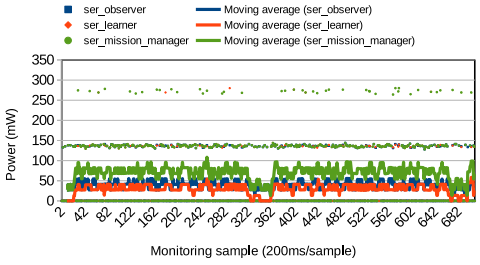


(B) Power consumption of Server Mission Manager: 120mW in average.

FIGURE 6.13: Power consumption of Mission Managers in the learning phase.



(A) Power consumption of Navigation Mission Manager: 70mW in average.



(B) Power consumption of Server Mission Manager: 75mW in average.

FIGURE 6.14: Power consumption of Mission Managers in the planning phase.

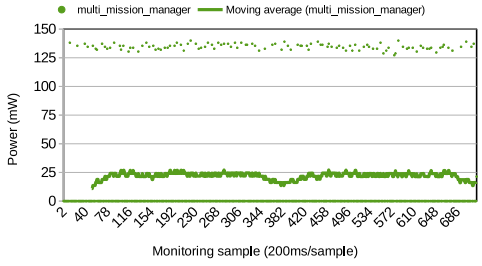


FIGURE 6.15: Power consumption of Multi-Mission Manager: 25mW in average.

server mission manager is higher than the one of the navigation mission manager. Figure 6.15 indicates the power consumption of the multi-mission manager. We can realize that this multi-mission manager consumes approximately $25mW$ in average. These energy overhead costs can be considered low enough to be deployed in the embedded context.

6.3 Towards A More Complex Multi-Mission Context

In this experimentation, we would like to propose a more complex multi-mission context with the addition of a semantic environment understanding mission. Figure 6.16 illustrates an example of the functionality of this third mission while the robot is navigating. To integrate this mission into the multi-mission manager, we need firstly to deploy the learning phase to find the adaptation models for the semantic mission manager in Section 6.3.1. The experimentation methodology and our expectations are finally presented in Sections 6.3.2.

6.3.1 Semantic Environment Understanding Mission Manager

In this section, we present the experimental results of a DQN-based semantic environment understanding mission manager that was not deployed in our simulation framework because of the complexity of adding realistic objects and persons in the

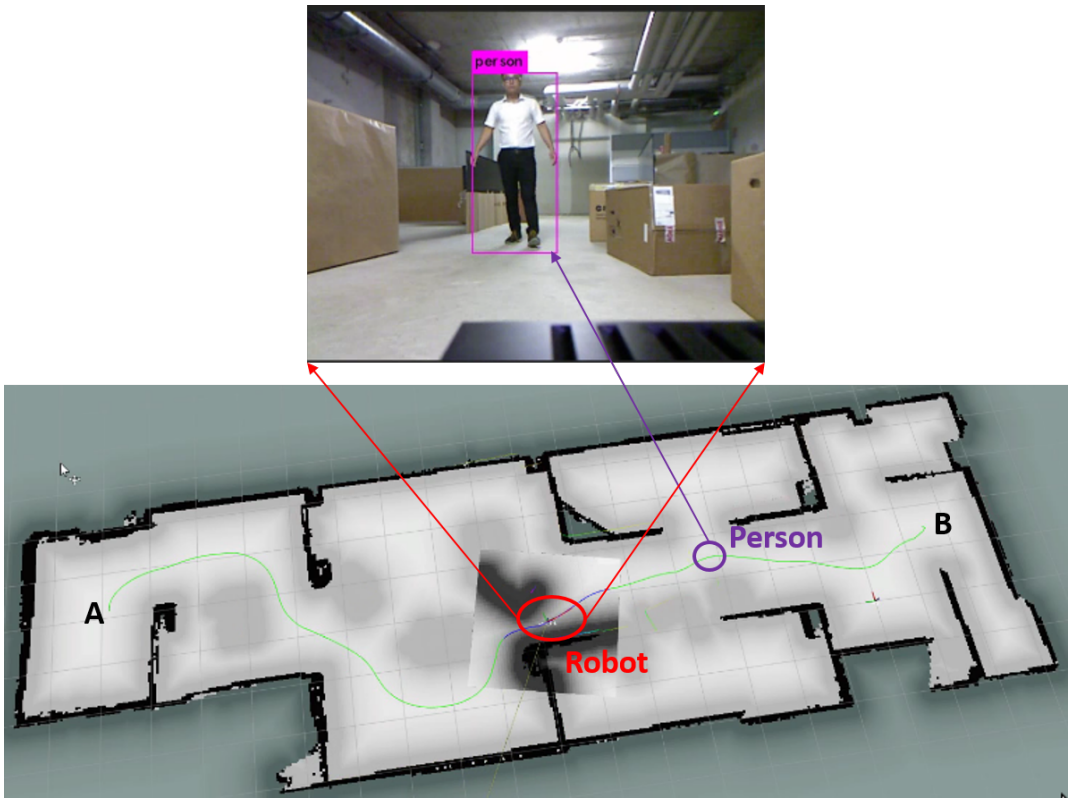


FIGURE 6.16: Integration of a semantic environment understanding mission into the multi-mission context on the real robotic platform. The robot is navigating from A to B and the semantic mission is in charge of detecting and recognizing the objects including people in the environment.

simulation environment. The characterization of this mission was presented in Section 3.6.3. We will state and formulate the problem for this mission manager and then implement and deploy the learning phase in the Xavier platform. The adaptation models (Q-Networks) learned during this learning phase will be then used as an initialization for the online learning process.

Problem Statement

The functional objective of the semantic environment understanding mission is to detect and recognize the objects including people in front of the robot (Figure 6.16). A set of three non-functional goals $G_{Sem\_Mission}$ can be applied to this mission:

$$G_{Sem\_Mission} = \{mAP_{mean\_min}, P_{c\_mean\_max} FPS_{mean\_min}\} \tag{6.1}$$

Where,

- mAP_{mean_min} means the miminum mean detection accuracy;
- $P_{c\_mean\_max}$ is the maximum mean computing power consumption;
- FPS_{mean_min} is the minimum mean frame rate.

The configuration knobs of the semantic mission are defined by the type of YOLO models used for the inference: $K_{Sem\_Mission} = \{type\_net\}$, where $type\_net \in \{Yolov3-tiny, Yolov3-160, Yolov3-288, Yolov3-416\}$ (see Section 3.6 for more details about these YOLO models).

Problem Formulation

Following the generic pattern in Section 4.3.3, we will formulate the problem of the DQN-based semantic mission manager with three main steps below.

State Space Formulation. In this mission, three internal observations related to the three non-functional requirements will be monitored: *mean detection accuracy*, *mean computing power consumption* and *mean frame rate*. Then, the external situation is evaluated by the *number of persons detected in the scene*. We now propose 4 run-time metrics representing the state space of the DQN-based semantic mission manager:

- mp_1 =*inverse of mean detection accuracy*;
- me_2 =*mean computing power consumption*;
- mo_1 =*inverse of mean frame rate*;
- m_{ext} =*mean number of persons detected by YOLO*.

Action Space Formulation. The action space is also defined as the configuration knobs $K_{Sem\_Mission} = \{type\_net\}$ as shown in Table 6.11.

TABLE 6.11: Action space of Deep Q-Learning based Semantic Mission Manager: configuration knobs $K_{Sem\_Mission} = \{type\_net\}$.

K1	K2	K3	K4
Yolov3-tiny	Yolov3-160	Yolov3-288	Yolov3-416

Reward Function Formulation. The reward function determines the level of non-functional goal violation of the mission. Therefore, three run-time metrics mp_1,

me_2 and mo_1 are used to calculate the reward as mentioned in two equations 4.16 and 4.17. These goals are considered with the same importance weights and these weights are set to 1.0.

Implementation

As a capability of generalization of our methodology, the same architecture and technique of Q-Network as shown in Table 4.8 are used for this third mission by only changing the size of the input and output layers. The Q-Network of the semantic mission manager has 4 inputs and 4 outputs. Thus, the weight matrix θ_{sem} has

$$(4 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 4 + 4) = 8900$$

weights. These weights are also floating-point numbers of 8 *bytes*, the Q-Network of the semantic mission will therefore occupy approximately $8900 \times 8\, bytes = 70\, kBytes$.

Learning Phase Deployment. The management of the mission is also realized within episodes of 1 minute like the other missions. The adaptation time step is set to 4 seconds for this mission, so there are $60/4 = 15$ time steps or decision epochs within an episode.

TABLE 6.12: Non-functional goals for Semantic Mission in the real experimentation.

NFRs	**Semantic Mission**
G_{perf_1}	$mAP_{mean\_min} = 50.0$
G_{energy_1}	
G_{energy_2}	$P_{c\_mean\_max} = 21.0W(H) - 20.5W(M) - 20.0W(L) - 0W(O)$
G_{others_1}	$FPS_{mean\_min} = 25.0Hz$

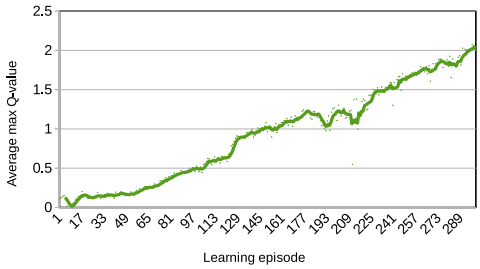
Based on the real characterization results presented in Chapter 3, the NFRs defined for the semantic mission are shown in Table 6.12. Three computing power consumption modes are also defined. We deploy the learning phase for the high mode and then apply the transfer learning technique (see Section 4.8.7) for the medium and low modes. Due to the high power consumption of this mission, to have long enough learning phases, we have recorded a set of navigation scenes (with objects and persons) and their timestamped sensor data using *rosbag*[6] and proceeded to the learning on the Xavier board (plugged to the AC power) replaying the scene databases.

TABLE 6.13: Parameters for exploration and exploitation for the learning phase of a DQN-based Semantic Mission Manager.

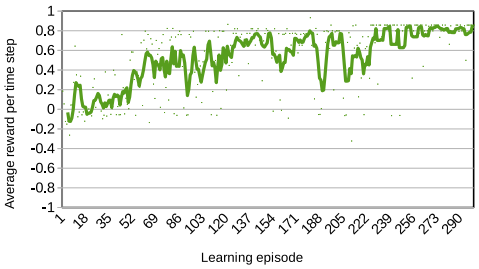
ϵ_{max}	ϵ_{min}	ϵ_{decay}
0.5	0.01	0.001

Learning Results. Figure 6.17 and 6.18 show the results of the learning phase for the semantic mission manager at the high computing power mode during 300 learning episodes with the exploration-exploitation parameters defined in Table 6.13. We can realize the evolution of the average max Q-value and the average reward

[6]See http://wiki.ros.org/rosbag



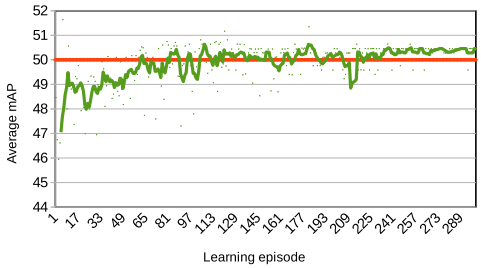
(A) Average max Q-value of Semantic Mission.



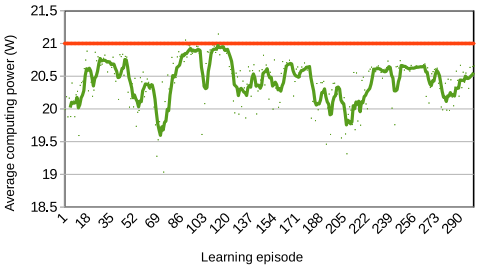
(B) Average reward per time step of Semantic Mission.

(C) Level of NFRs violation of Semantic Mission.

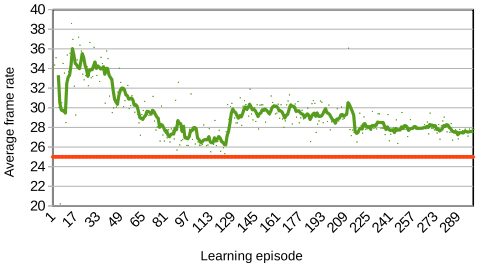
FIGURE 6.17: Learning phase results of a DQN-based semantic mission manager at the *High* computing power mode.



(A) Average detection accuracy measured by mAP.



(B) Average computing power consumption.



(C) Average frame rate.

FIGURE 6.18: Learning phase results of a semantic mission manager at the *High* computing power mode: non-functional requirements.

per time step as well as the reduction of the level of NFRs violation over learning episodes in Figure 6.17. If we increase the number of learning episodes, the Q-values can continue increasing. However, we accept the current performance with

a constant trend of the obtained reward and the level of NFRs violation. Figure 6.18 shows the characteristic of the three non-functional goals of the mission over learning episodes with the red lines indicating the reference values and the green lines indicating the measured values. At this high computing power mode, the constraints of the computing power and the frame rate are quite light, so they can be satisfied throughout the learning time. While the detection accuracy as a desired quality of service requires the entire learning time to be satisfied.

Discussion

Unlike DQN-based mission managers of the navigation mission and the video server mission whose learning phase can be deployed in the simulation framework and reused in the real framework, the learning phase of the DQN-based semantic mission manager is deployed in the real framework using the recorded scene databases. The generic pattern of the problem formulation and the architecture of Q-Network can be easily applied to this mission. The Q-value and reward features are similar to those of the navigation and video server mission managers presented in Chapter 4. Thus, the effectiveness of our proposed mission manager based on DQN is validated once again. The learned adaptation models of this mission are then integrated into the multi-mission context. The next sections will present the methodology of the second experimentation.

6.3.2 Experimentation Methodology

A Context of Three Robotic Missions in Parallel

Section 6.2 presents the experimentation of a navigation mission and a video server mission in parallel. The functionalities of our multi-mission manager have been successfully validated by the five different scenarios. In this experimentation, with the

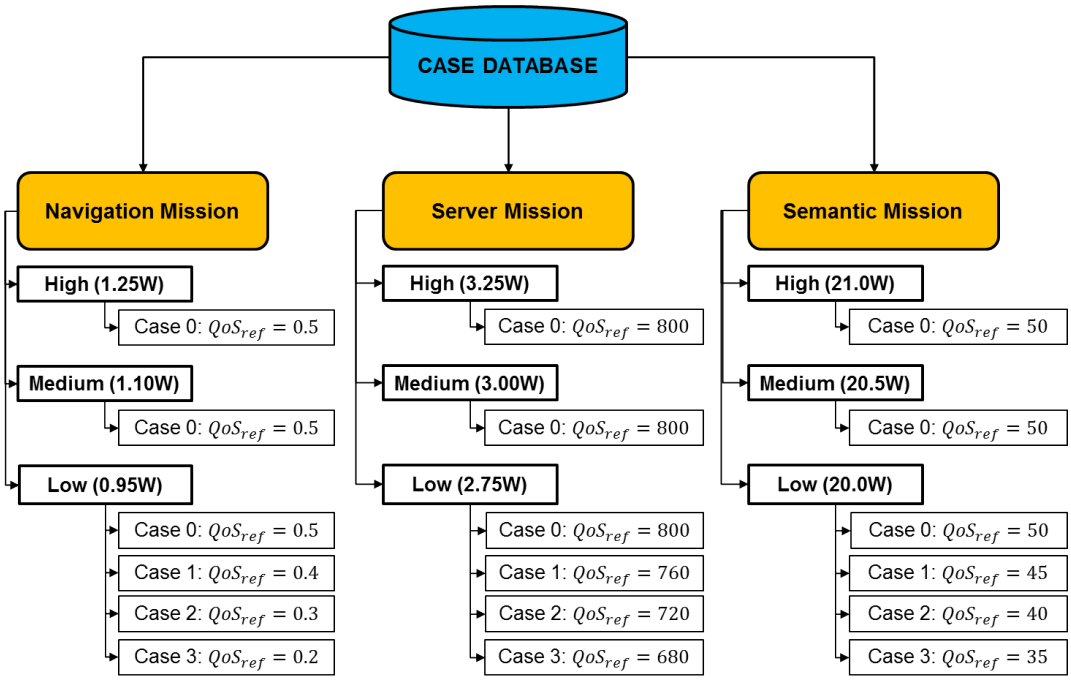


FIGURE 6.19: An initialization of the case database for the navigation mission, the server mission and the semantic mission in the real experimentation.

addition of the third mission such as a semantic environment understanding mission, we propose a more complex scenario where three missions including a navigation mission, a server mission and a semantic mission are deployed in parallel. An initialization of the case database in this scenario is presented in Figure 6.19. This case database can evolve at run-time using the online learning capability of our methodology as presented in the previous chapter. The system-level constraint such as maximum computing power capacity is also simulated as shown in Table 6.14. Indeed, these values are chosen in a subjective way in order to demonstrate the different behaviours of the multi-mission context. Based on this power capacity as well as the current mission priorities, our multi-mission manager will decide to reallocate the computing power budget or change the computing power mode for each mission.

TABLE 6.14: Maximum power capacity vs. battery state in case of three missions: a navigation mission, a video server mission and a semantic mission.

Battery State	**0**	**1**	**2**	**3**	**4**
$P^{\sum}_{c\_max}$	26.0W	24.0W	22.0W	10.0W	5.0W

Perspective

The objective of this experimentation is to further demonstrate the generalization of our management methodology in the multi-mission context. We have successfully integrated the semantic mission into the mutli-mission context and our robotic system has been available for this demonstration. However, at the time of writing this manuscript, the results of this experimentation are not available and they are not documented in this manuscript, and we hope to complete this demonstration and present it at the time of the thesis defense.

6.4 Summary

This chapter presents a set of experimentations of a mobile robotics platform in a real environment aiming to validate the proposed QoS and power management methodology. The validation scenarii demonstrate the reactivity to internal events (battery decreasing power budget) or external ones (user changes in mission's priorities, in mission's requests) with the objective of guaranteeing the total system power budget. The robustness of our methodology is proven by the coherency between simulation and real-world results. In addition, through the ability to learn online, our mission managers can deal with many real unforeseen scenarios and evolve their adaptation policy over time. These managers therefore contribute to the long-term autonomy of mobile robotic systems. Indeed, a maximum gain of 1.22 (equivalent to an efficiency gain of 22%) in terms of system-level computing power consumption is achieved while comparing our run-time adaptation methodology with the static configurations. In a context of three robotic missions including a navigation mission, a server mission and a semantic mission, an extra control subsystem power consumption (with three mission managers and a multi-mission manager) is estimated as 245mW representing 0.94% of the 26W total power budget reserved to all the three missions. Finally, a perspective of a more complex multi-mission context is opened up, which will constitute an interesting demonstration during the thesis defense.

Chapter 7

Thesis Conclusions

Contents

The last chapter will conclude our thesis with a general discussion in Section 7.1. In Section 7.2, we provide an evaluation on the maintainability of our management framework including the modularity, the reusability and the extendability, and on the generalization of our proposed methodology. Finally, we will discuss the current limitations of our methodology and propose future directions for the research in Section 7.3.

7.1 General Conclusions

The objective of this thesis is to contribute to the autonomy of mobile robotic systems by guaranteeing the non-functional requirements such as quality of service and energy efficiency. The fundamental problem is how to reconfigure dynamically some key parameters of the robotic missions under dynamic operational conditions while respecting the non-functional requirements. Our research is realized via three main steps as well as three main contributions: Robotic Mission Characterization, NFR-Aware Self-Adaptive Mission Manager and Multi-Mission Manager.

- **Robotic Mission Characterization**. In our thesis, we approach a robotic mission from a new perspective where the mission is at the highest level of abstraction in a robotic system. At this level, we focus on the overall performance and resource consumption of all mission components such as sensors, actuators and high-level applications. For a robotic mission, the functional and non-functional requirements can be applied. The non-functional requirements we consider for a robotic mission are mission-specific performance such as quality of service, energy consumption for sensing, acting and computing parts and other requirements such as safety, security, etc. In fact, many dynamic factors in the operational conditions of the robotic systems can have an impact on these non-functional requirements. A reconfiguration capability is therefore required to ensure these requirements. In a robotic system, the reconfiguration techniques can be applied at the mission level, such as adapting appropriate

components and their parameters. At the computing system level, the mission's computing resources can be reallocated or the operating point can also be adjusted. In general, the result of the robotic mission characterization is a set of reconfiguration techniques used to modify the non-functional properties of the mission. As case studies, we characterized three robotic missions: an autonomous navigation mission, a video server mission and a semantic environment understanding mission. These characterization results highlighted the motivation for the adaptation at run-time with the NFR-Aware Self-Adaptive Mission Manager;

- **NFR-Aware Self-Adaptive Mission Manager**. Our second contribution addresses the Self-Adaptive Mission Manager. With a set of non-functional requirements such as quality of service and energy consumption applied to the mission, the self-adaptive mission manager determines how to reconfigure the mission under dynamic operational conditions to ensure these requirements. This manager is composed of three main elements: a monitoring element, a decision-making element and a reconfiguring element. The monitoring element is responsible for perceiving the internal and external context of the mission and the reconfiguring element changes dynamically the mission configuration. The decision-making element that has the adaptation policy mapping states and actions is the core of the mission manager. In our thesis, we formulated the decision-making problems as Markov Decision Processes and proposed the algorithms based on Q-Learning and Deep Q-Learning to resolve these problems. The methodology was validated both in simulation and real frameworks and the results proved the robustness of our methodology. The energy consumption costs of the mission managers have been characterized and they have also proven the feasibility of deployment in an embedded context such as a NVIDIA Jetson Xavier embedded platform;

- **Adaptive and Hierarchical Multi-Mission Manager**. Finally, we also proposed a hierarchical management for managing the multi-mission context with shared resources and different priorities. At the global level, a multi-mission manager monitors the state of the shared resources, the state of the missions and their priorities to decide whether to reallocate resources to missions, promote or degrade their quality of service or to trigger the online learning capability of the mission managers. The problem of changing non-functional requirements is also resolved by the case-based reasoning technique. We have successfully validated the multi-mission manager for both simulated and real scenarios of a navigation mission and a video server mission in parallel.

7.2 Maintainability and Generalization of Our Framework

7.2.1 Maintainability

Our methodology has been prototyped both in the simulation and in the real robotic framework. In this section, we provide a discussion on the maintainability of our management framework with three main axis including the modularity, the reusability and the extendability as follows:

- **Modularity**. Our framework is developed with the aids of the robotic firmware ROS that is component-based, modularized and distributed. Thus, our system

is modularized and organized in the form of ROS nodes. For example, a mission manager employs three ROS nodes: a node for the monitoring part, a node for the decision-making part and a node for the executing part, while the multi-mission manager employs a single ROS node where the monitoring, decision-making and executing parts are realized inside this node. The communication between these nodes is realized via the message-passing mechanism. Changing the algorithm in a node do not influence the others as long as the message format is the same between publishers and subscribers;

- **Reusability**. Our ROS-based framework also facilitates the reusability. Indeed, for the mission managers, while the monitoring and executing parts are specific to the missions with different monitored metrics and key parameters, the decision-making parts can be reused. For example, the algorithm and the architecture of Q-Networks have been the same for the three mentioned robotic missions;

- **Extendability**. This point discusses the extendability as well as the scalability of the multi-mission manager when we have to integrate a new robotic mission. To consider the run-time adaptation capability for a new mission, we must follow a generic workflow including mission characterization to find the mission knowledge such as the characteristic of quality of service, energy consumption, configuration knobs, reinforcement learning based problem formulation according to the generic pattern as described in Figure 4.2 and learning phase deployment to find its adaptation models. Once the mission's adaptation models are found, we will integrate this mission into the multi-mission context. This generic workflow is described in Figure 7.1. Currently, our proposed multi-mission manager is not a plug-and-play mechanism. It means that we need to add some extra codes related to the new mission such as integrating the case database of this mission, monitoring of the adaptation response and sending the adaptation request to this new mission manager. However, the algorithms (Algorithms 1 and 2) and the principles of the three decisions (reallocating the power budget, throttling the desired quality of service and triggering the online learning) of the multi-mission manager would not be changed. Moreover, we tried to provide some generic functions to make the implementation more efficient. To reduce this current limitation and increase the scalability of our multi-mission manager, the ROS *pluginlib*[1] that supports the dynamic loading of software modules is one of our perspective.

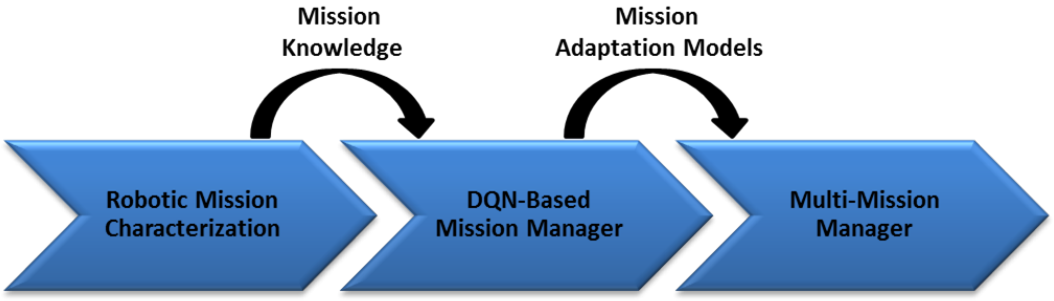


FIGURE 7.1: A generic workflow to integrate a new robotic mission into the adaptive and hierarchical multi-mission context management.

[1] http://wiki.ros.org/pluginlib

7.2.2 Generalization

The three above points (modularity, reusability and extendability) discuss the aspect of the software implementation of our management framework. In this section, we aim to discuss the generalization of our proposed methodology with the following points:

- **Mission Characterization**. The characterization process may be different for each mission. For example, for a navigation mission and a video server mission, we can deploy the characterization process in the simulation framework, while for a semantic environment understanding mission, we must record a scene database and deploy the characterization on the real robot computing platform (NVIDIA Xavier). This difference is due to the fact that it is easy to simulate a mission in a simulator or not, for example, adding real objects and people detected by the semantic mission in a Gazebo simulator is not trivial. In addition, if a mission requires the calculation on a GPU hardware (or a specialized computing hardware in general) such as the semantic mission and the simulation framework does not have this type of hardware, we must deploy the characterization process on the real robot's computing platform. Thus, understanding the domain of a new robotic mission is necessary to choose the best way to characterize the mission;

- **Mission Knowledge**. As presented in Section 4.3, the mission knowledge is obtained through an offline characterization process and is used for the run-time adaptation algorithm (Figure 7.2). Knowledge of the mission is obviously specific to each mission. When considering the mission quality of service aspect (or mission-specific performance), the average robot speed is chosen to imply the quality of service of a navigation mission, while the average resolution of encoded output images is chosen for a video server mission and the neural network detection accuracy is defined as the quality of service of a semantic environment understanding mission. These metrics depend a lot on the mission, sometimes on the wishes of the system designer, and we also need a mission-specific way to monitor them. For example, we need the information provided by odometers to estimate the average speed of the robot for the navigation mission, or we need to probe the application codes to obtain the average resolution of the encoded images of the video server mission, or a linear estimation model of the detection accuracy can be useful for the semantic mission. Thus, for a new robotic mission, we may not understand the application algorithm, but we need to know its concept of quality of service

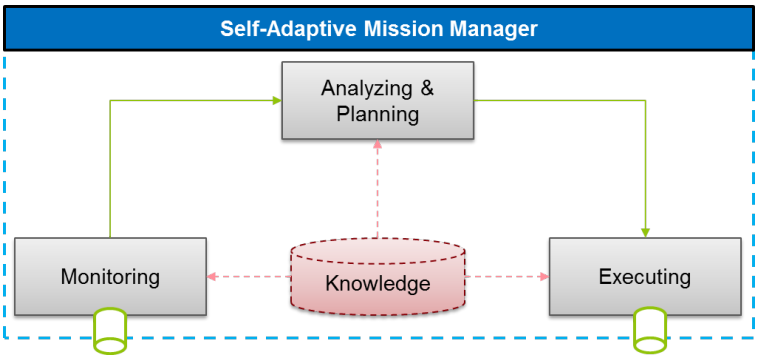


FIGURE 7.2: Mission knowledge used for the run-time adaptation algorithm.

and how we can monitor it. There is no common tool or probe to monitor this type of mission-specific metric. This is similar to other mission-specific metrics, such as the distance to obstacles for a navigation mission, the number of people detected by a semantic mission, etc. For metrics related to the energy consumption of a robotic mission, the sensing and acting part is specific to the type of sensors and actuators. For the computing power part, with the help of PowerAPI (Appendix C), we tried to provide a common tool to monitor the computing power consumption of a mission as long as the mission processes are identified. The other mission specific knowledge parameters are the configuration knobs that have real impact on the main non-functional requirements. Once we have set the performance, energy and extra goals of the new mission to integrate along with its configuration knobs, the Monitoring, Analyzing and Planning and Executing blocks of Figure 7.2 can be simply adapted to target this new mission;

- **Multi-Mission Context**. In the multi-mission context where multiple robotic missions can be deployed simultaneously and on the same robotic platform, we have assumed that each mission is independent of each other, and it can be characterized in isolation of the others. The generalization of this procedure is viable while this assumption is verified and the interference between the missions can be ignored. The interference occurs mainly in the computing platform of the robotic system where a robotic mission is deployed in the form of a multi-process computing workload. An interference management at the computing platform level is currently outside the scope of our thesis. Thus, to minimize this type of interference, we must ensure that the total computing workload of all the robotic missions does not cause an overload (for example CPU or memory overload) for the computing platform;

- **Online Planning**. In our thesis, we target a hybrid planning strategy (see Section 2.2.3) where the adaptation plan/policy is found via an offline learning phase in simulation (such as for the navigation mission and the video server mission), or on a real platform with a recorded scene database (such as the semantic environment understanding mission), and this policy can then evolve during mission operation on the real robotic platform thanks to the online training capability. However, if we cannot deploy the offline learning phase to find the adaptation policy (for example, a recorded scene database is not available for the semantic mission), we can deploy the learning phase (with the online learning) during mission operation on the real robotic platform thanks to our proposed approach base on the reinforcement learning. This planning strategy is therefore known as online planning. However, in this case, we must accept an additional cost in terms of energy and time for the learning algorithm to converge.

7.3 Limitations and Future Work

Although our methodology has been successfully validated in both simulation and real-world experimentation, our first original study on the problem of quality of service and energy management in robotic systems still has some drawbacks as follows:

- Firstly, our perspective on the characterization of a robotic mission is different from traditional approaches, which are usually based on robotic tasks, control architecture or robotic planner. With this approach, we strive to hide the

complex details of a robotic system and improve the generalization of the management framework. However, this perspective is not always consistent with the approaches of other robotics researchers. Thus, a quantifiable assessment with the state of the art is our current difficulty. In addition, the characterization process is currently not automated. A systematic approach is proposed for this process. However, it is a non-trivial problem. Therefore, one of the future work is to propose an automated characterization process that facilitates the quantifiable evaluation with the state of the art and better convinces the robotics community about our proposed terminologies;

- Secondly, our management methodology is data-driven, not event-driven as mentioned in Section 3.4. This means that the necessary metrics are monitored, not events. We can predict which metrics to be changed by an event, but we cannot use the monitored metrics to determine which events have occurred. Thus, reasoning on the decisions given by the mission managers is difficult, even impossible. Another future work on combining data-event and root cause analysis to resolve this difficulty is also planned;

- Thirdly, as we can see in Figure 3.6, many configuration knobs can be used to handle the problem of guaranteeing non-functional requirements. We limit the scope of this thesis to the adaptation in the parameter space of mission's applications. However, the adaptation at the level of sensors such as acquisition frequency and operating mode, and the adaptation at the computing platform level such as computing core allocation, power modes, etc can contribute significantly to the energy efficiency of the whole robotic system;

- Finally, as mentioned in Chapter 4, the choice of hyper-parameters of the Deep Q-Learning algorithm deserves a careful effort to ensure the learning performance and convergence speed. In addition, a methodology to assess the performance of the online learning on the long-term adaptation will be welcomed;

Our thesis focuses on two properties of autonomous systems such as the mission's quality of service and energy efficiency. Our methodology assumes the normal operation of these systems and the management framework dynamically reconfigures some key mission parameters to address the dynamic operational conditions that influence non-functional requirements. These dynamic conditions did not take into account critical events such as possible failures of sensors, actuators, computing hardware, or unreliability of high-level applications. These critical events can lead to accidents or serious damage, for example in the case of self-driving cars. Making an autonomous system safer and more reliable is really a big concern in the robotics research community. The run-time adaptation is also necessary in these scenarios due to many dynamic operational conditions. However, the safety constraint is now a critical and harsh constraint. The improvement of our run-time management framework to guarantee safety constraints is therefore one of our research perspectives.

Appendix A

Characteristics of NVIDIA Jetson AGX Xavier

Main Characteristics

Launched in 2018, Jetson AGX Xavier is a high-performance embedded platform designed by NVIDIA (Figure A.1). This platform, equipped with an 8-core 64-bit ARM processor, a 512-core Volta GPU and deep learning and vision accelerators, as well as its development kit, allows us to develop and deploy end-to-end AI robotics applications with real-time performance and low power consumption. Its main tech specifications are found in Table A.1.

FIGURE A.1: NVIDIA Jetson AGX Xavier (reprinted from NVIDIA).

TABLE A.1: Tech specifications of NVIDIA Jetson Xavier (collected from NVIDIA).

AI Performance	32 TOPs
GPU	512-core Volta GPU with Tensor Cores
CPU	8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3
Memory	16GB 256-bit LPDDR4x | 137GB/s
Storage	32GB eMMC 5.1
DL Accelerator	(2x) NVDLA Engines
Vision Accelerator	7-way VLIW Vision Processor
Encoder/Decoder	(2x) 4Kp60 | HEVC/(2x) 4Kp60 | 12-Bit Support
Size	105 mm x 105 mm
TDP	30W (typical 20W)

System-Level Monitor

The Jetpack in the Jetson AGX Xavier development kit (in its version 4.2) provides a utility called *tegrastats* that reports memory usage, processor usage and power consumption. The main statistics provided by this utility are presented in Table A.2.

TABLE A.2: Main statistics provided by *tegrastats* utility.

Component	Statistics
RAM	use of RAM, total amount of RAM available, largest size of free block, amount of free blocks of largest size
CPU	load statistics of each core, frequency of each core, temperature, current power consumption, average power consumption
GPU	temperature, current power consumption, average power consumption
DLA & VA	current power consumption, average power consumption
DDR	current power consumption, average power consumption
SOC	current power consumption, average power consumption
SYS5V	current power consumtpion, average power consumption

The power consumption of the Xavier can then be estimated by summing the power consumption of components such as CPU, GPU, DLA&VA, DDR, SOC and SYS5V as the following equation:

$$P_{Xavier} = P_{CPU} + P_{GPU} + P_{DLA\&VA} + P_{DDR} + P_{SOC} + P_{SYS5V} \quad \text{(A.1)}$$

However, this power model does not take into account external devices connected to the Xavier such as LIDAR, Kinect sensors, etc. In fact, the communication between Xavier and external devices consumes a lot of energy. Thus, a more precise model would be defined as follows:

$$P_{Xavier} = P_{CPU} + P_{GPU} + P_{DLA\&VA} + P_{DDR} + P_{SOC} + P_{SYS5V} + P_{Comm\_Ext\_Devices} \quad \text{(A.2)}$$

Where $P_{Comm\_Ext\_Devices}$ is the power consumption reserved for communicating with the external devices. We can use a Yocto-Watt wattmeter to estimate this power consumption (Appendix D).

```
header:
  seq: 32
  stamp:
    secs: 1558001731
    nsecs: 807278304
  frame_id: ''
ram_usage: 5974.0
cores_usage: [2.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0]
cores_freq: [2265.0, 2265.0, 2265.0, 2265.0, 2265.0, 2265.0, 2265.0, 2265.0]
gpu_temperature: 26.5
cpu_temperature: 27.0
current_gpu_power: 1083.0
average_gpu_power: 1082.0
current_cpu_power: 464.0
average_cpu_power: 623.0
current_soc_power: 2321.0
average_soc_power: 2334.0
```

FIGURE A.2: Tegra statistics published in a ROS message.

Publish System Statistics in a ROS Message

For the objective of run-time monitoring, we develop a ROS node named *tegrastats_ros_node* and publish periodically some necessary statistics provided by *tegrastats* in a ROS message as described in Figure A.2. The default monitoring period of the *tegrastats* is 1000ms. However, we provide a finer monitoring period of 200ms. Thus, the system statistics are periodically published every 200ms.

Appendix B

Characteristics of Pioneer-3DX

Main Characteristics

Pioneer-3DX designed by Adept Mobile Robots is a small lightweight two-wheel two-motor differential drive robot ideal for the research and the education [MobileRobots, 2006]. This mobile base is equipped with sonars, bumpers, wheel encoders, a microcontroller for low-level control, and battery modules. Its main characteristics are presented in Table B.1.

TABLE B.1: Main characteristics of Pioneer-3DX.

Operation	Robot Weight: 9kg Operating Payload: 17kg
Diff. Drive Movement	Max. Forward/Backward Speed: 1.2m/s Rotation Speed: $300^o/s$
Power	Run Time: 8-10 hours w/3 batteries without accessories, 3-4 hours with accessories and onboard computer Charge Time: 12 hours Available Power Supplies: 5V @ 1.5A, 12V @ 2.5A
Batteries	Supports up to 3 at a time Voltage: 12V Capacity: 7.2Ah (each)
Battery Voltage State	Full charge: $\geq 12.5V$ Low battery threshold: $11.5V$ Shutdown threshold: $11.0V$

ROSARIA

ARIA provides a software framework for controlling as well as receiving data from all MobileRobots platforms. And ROSARIA[1] is a wrapper of ARIA for interfacing with the ROS system. A ROS node named *RosAria* will subscribe the velocity commands to control motors and publish the information related to all embedded sensors and the state of motors. For the batteries, it publishes the battery voltage and the state of battery recharge (recharging or not), but the remaining state of charge is not given. We need therefore a watt-meter to measure the power consumption from the robot's batteries.

[1] See http://wiki.ros.org/ROSARIA

Appendix C

Process-Level Power Estimation with PowerAPI

Overview of PowerAPI

PowerAPI[1] is a system library providing a programming interface (API) to monitor at run-time the power consumption of software at the granularity of system processes [Noureddine, Rouvoy, and Seinturier, 2015]. Each process is considered as an independent entity. Thus, if an application or a software has two processes, the power consumption of this application is the sum of power consumption of these two processes. The power consumption of a software $P_{software}$ is estimated from two sources as defined follows

$$P_{software} = P_{comp} + P_{comm}, \tag{C.1}$$

where, P_{comp} means the CPU power consumed by software and P_{comm} is equal to the power consumed by the network card for transmitting software's data.

Process-Level Power Monitor ROS Node

We develop a ROS node named *powerapi_ros_node* to monitor and publish the power consumption of a specific process to the ROS system. This ROS node takes two parameters, one for the PID of process that we want to monitor and other for the monitor interval.

Mission's Computing Power Consumption

Mission Not Using GPU

As mentioned in Section 3.2.2, a robotic mission includes many application processes. In this case, these applications do not use the GPU. By determining the PIDs of these processes, we can monitor the power consumption of these processes with the *powerapi_ros_node* and then, the power consumption of the robotic mission is the sum of power consumption of these processes.

Mission Using GPU

In this case, the mission's applications are executed both in the CPU and GPU. PowerAPI provides only the computing power consumption on the CPU. So, we need

[1] See https://github.com/powerapi-ng/powerapi-scala

another tool for estimating the power consumption on the GPU. In our thesis, we assume that there is only a mission executed in the GPU, so the total power consumption of GPU is reserved for this mission. This GPU power consumption can be measured by the *tegrastats* utility as presented in Appendix A. Finally, the mission's computing power consumption is the sum of CPU power consumption estimated by PowerAPI and GPU power consumption estimated by *tegrastats*.

Appendix D

Power Measurement with Yocto-Watt

Yocto-Watt[1] is a digital wattmeter that allows us to monitor the power consumption of electrical devices. The energy consumption information can then be transferred to the computer via USB.

Measuring Overall Power Consumption

Figure D.1 indicates the installation of Yocto-Watt in the Pioneer-3DX mobile base to measure the power consumption of the robot batteries. The battery output is connected to the Yocto-Watt input and the Yocto-Watt output is connected to the power board that powers the entire robotic system. We then connect the Yocto-Watt with the Xavier via a USB cable and use the C++ API to obtain the Yocto-Watt measurements.

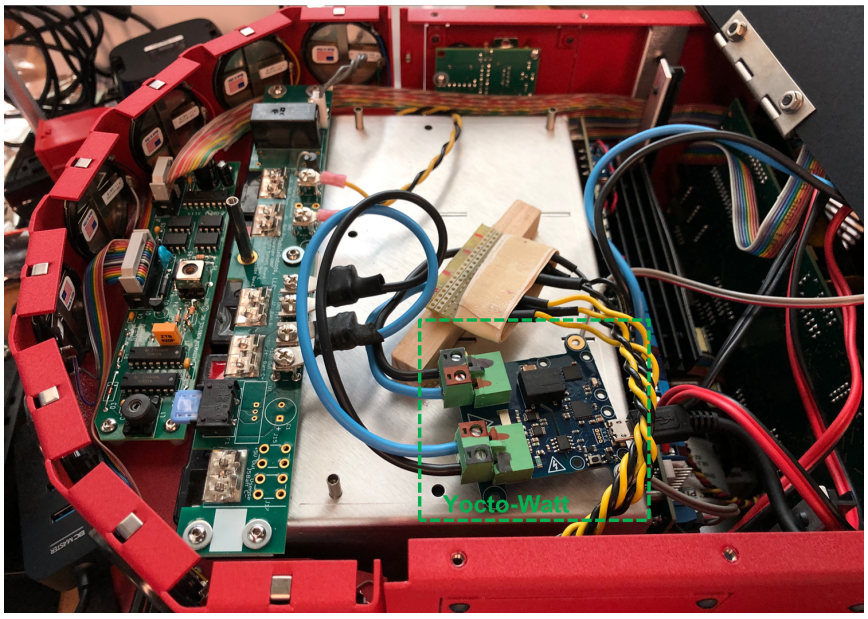


FIGURE D.1: Install Yocto-Watt in Pioneer-3DX mobile base.

Yocto-Watt measures the overall power consumption of the components powered by the robot's batteries (P_{Batt}):

$$P_{Batt} = P_{motors} + P_{embedded\_processor} + P_{Kinect} + P_{LIDAR} + P_{Xavier} \quad \text{(D.1)}$$

[1]See http://www.yoctopuce.com/EN/products/capteurs-electriques-usb/yocto-watt

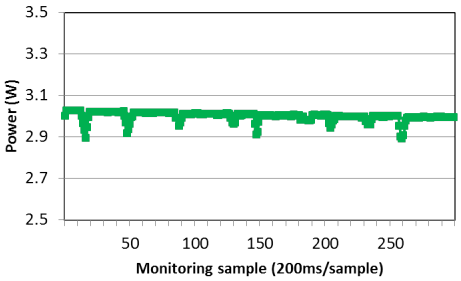
Where P_{motors} is the power consumption of motors, $P_{embedded\_processor}$ is the power consumption of embedded microcontroller inside the mobile base for low-level controller, P_{Kinect}, P_{LIDAR} and P_{Xavier} are correspondingly the power consumption of Kinect, LIDAR sensors and Xavier board.

Estimating Each Component's Power Consumption

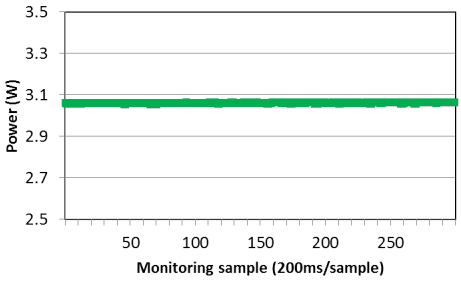
By using Yocto-Watt, we would like also to estimate the power consumption of the components such as Kinect sensor, LIDAR scanner as well as the power consumption reserved for the communication between these components and the Xavier board ($P_{Comm\_Ext\_Devices}$ as mentioned in Equation A.2). For sake of simplicity, the power consumption of these components are then considered as static (see Section 3.2.3). The power consumption of the robot motors and their embedded processor is considered in an unified part and it represents the acting power consumption of the navigation mission $P_{acting}^{navigation}$:

$$P_{acting}^{navigation} = P_{motors} + P_{embedded\_processor} \tag{D.2}$$

For measuring the power consumption of a component such as Kinect sensor or LIDAR scanner, the Yocto-Watt is intertwined between the component and its power supply. The measured results of the two components are shown in Figure D.2. We can realize that the power consumption of the two components is almost unchanged.



(A) Power consumption of Kinect sensor: $3.00 \pm 0.02W$.



(B) Power consumption of LIDAR scanner: $3.06 \pm 0.00W$.

FIGURE D.2: Estimated power consumption of Kinect and LIDAR sensors using Yocto-Watt.

To estimate the power consumption reserved for the communication between the Xavier board and its external devices such as Kinect, LIDAR and robot's embedded processor, we place the Yocto-Watt between the Xavier board and its power supply. Then, we connect each external device in turn to the Xavier board and calculate the difference in power consumption between without and with the external device. We determine four cases:

- **Init**: This case indicates the initial configuration of the Xavier board with the external devices such as USB hub, USB wifi and Yocto-Watt;
- **Init + Kinect**: This case includes the initial configuration and a Kinect sensor connecting to the Xavier board;
- **Init + LIDAR**: This case includes the initial configuration and a LIDAR scanner connecting to the Xavier board;

- **Init + Robot's Processor**: This case includes the initial configuration and a robot's embedded processor connecting to the Xavier board.

The measurement results of the four cases are presented in Figure D.3. Finally, the power consumption reserved for the communication between the Xavier board and each individual component is calculated by the difference in power consumption of these cases and is shown in Table D.1.

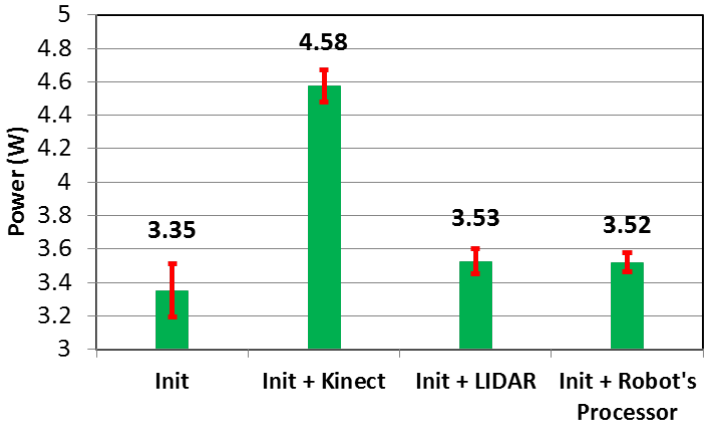


FIGURE D.3: Power consumption reserved for the communication between the Xavier board and its external devices.

TABLE D.1: Power consumption for the communication between the Xavier board and its external devices.

P_{Comm_Init}	P_{Comm_Kinect}	P_{Comm_LIDAR}	$P_{Comm\_RobotProcessor}$
$3.35W$	$1.23W$	$0.18W$	$0.17W$

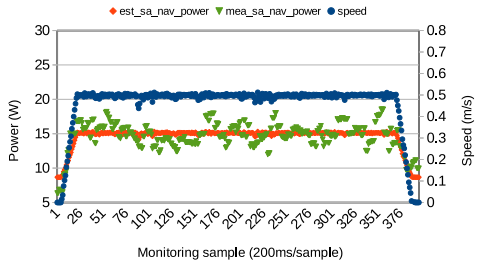
Validation of Sensing and Acting Power Consumption Model of Navigation Mission

We aim to provide an assessment on the model of sensing and acting power consumption of the navigation mission that we used for the simulation framework as presented in Table 3.1. In fact, this model is defined as

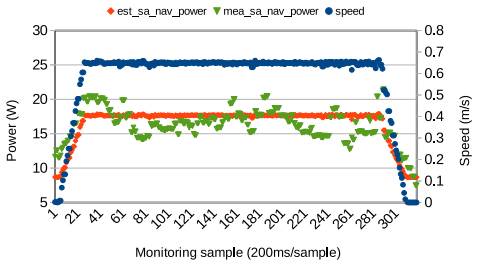
$$P^{navigation}_{sensing\&acting} = P_{motors} + P_{Controller} + P_{LIDAR} = 6.25v^2 + 9.79v + 8.67W \quad \text{(D.3)}$$

Where $P^{navigation}_{sensing\&acting}$ is the estimated sensing and acting power consumption of the navigation mission (in $Watts$), P_{motors} is the estimated robot motors power consumption, $P_{Controller}$ is the estimated robot controller power consumption (in $Watts$), P_{LIDAR} is the estimated LIDAR laser scanner power consumption (in $Watts$), and v is the robot speed (in m/s). And we now use the measurement by a Yocto-Watt wattmeter to compare with this power model.

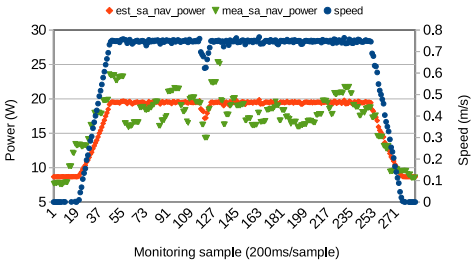
In Figure D.4, we present three cases of the maximum robot speed (v_{max}), 0.5, 0.65 and $0.8m/s$. We can see that at the different robot speeds, the sensing and acting power consumption of the navigation mission is also different and we can say that the robot speed is a knob to change the characteristic of the power consumption. When comparing with the power estimation model, we can realize that



(A) At the robot speed of $0.5m/s$.



(B) At the robot speed of $0.65m/s$.



(C) At the robot speed of $0.80m/s$.

FIGURE D.4: Validation of sensing and acting power consumption model of a navigation mission: *speed* indicates the current speed of the robot in m/s (blue points), *est_sa_nav_power* is the estimated value according to the equation D.3 (red points) and *mea_sa_nav_power* indicates the measured value (green points).

the estimated values are fairly constant when the robot speeds are less variable. Whereas, the measured values are quite variable even the robot speeds are less variable. However, if we consider the average values of the estimation model and the measurement, we can obtain two nearly equal values. Indeed, in our management framework, we consider the average values, not the instant values. Thus, the power estimation model is verified and we can use this model both in the simulation framework and in the real framework.

Appendix E

List of Publications

The following is a list of publications resulting from the research work of this thesis. Some of them have been published in national and international conferences and another is accepted to appear in an international journal.

International Journal

- Dinh-Khanh HO, Karim BEN CHEHIDA, Benoit MIRAMOND, and Michel AUGUIN, *Learning-Based Adaptive Management of QoS and Energy for Mobile Robotic Missions*, International Journal of Semantic Computing, vol. 13, no. 4, 2019.

International Conference

- Dinh-Khanh HO, Karim BEN CHEHIDA, Benoit MIRAMOND, and Michel AUGUIN, *QoS and Energy-Aware Run-time Adaptation for Mobile Robotic Missions: A Learning Approach*, 2019 IEEE International Conference on Robotic Computing, 10.1109/IRC.2019.00039;
- Dinh-Khanh HO, Karim BEN CHEHIDA, Benoit MIRAMOND, and Michel AUGUIN, *Towards A Multi-Mission QoS and Energy Manager for Autonomous Mobile Robots*, 2018 IEEE International Conference on Robotic Computing, 10.1109/IRC.2018.00057.

National Conference

- Dinh-Khanh HO, Karim BEN CHEHIDA, Benoit MIRAMOND, and Michel AUGUIN, *Towards A Multi-Mission QoS and Energy Manager for Autonomous Mobile Robotic Systems*, 13th National Conference on Software and Hardware Architectures for Robots Control, sharc2018.sciencesconf.org.

We are currently working on the publication of our original materials of the DQN-based Mission Manager, the Multi-Mission Manager and the real experimentation results in IEEE Transactions on Robotics (T-RO).

Bibliography

Aamodt, Agnar and Enric Plaza (1994). "Case-based reasoning: Foundational issues, methodological variations, and system approaches". In: *AI communications* 7.1, pp. 39–59.

Abdulmajeed, Wael R. and Revan Zuhair Mansoor (2014). "Implementing Autonomous Navigation Robot for building 2D Map of Indoor Environment". In: *International Journal of Computer Applications* 92.1, pp. 7–13.

Abeni, L. and G. Buttazzo (2001). "Hierarchical QoS management for time sensitive applications". In: *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*, pp. 63–72. DOI: 10.1109/RTTAS.2001.929866.

Ahmad, Aakash and Muhammad Ali Babar (2016). "Software architectures for robotic systems: A systematic mapping study". In: *Journal of Systems and Software* 122, pp. 16 –39. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.08.039.

Bekey, George A. (2005). *Autonomous Robots: From Biological Inspiration to Implementation and Control*. The MIT Press.

Berenz, Vincent, Fumihide Tanaka, and Kenji Suzuki (2011). "Autonomous battery management for mobile robots based on risk and gain assessment". In: *Artificial Intelligence Review* 37, pp. 217–237.

Berkenkamp, Felix, Andreas Krause, and Angela P. Schoellig (2016). "Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics". In: *ArXiv* abs/1602.04450.

Bihlmaier, Andreas and Heinz Wörn (2014). "Increasing ROS Reliability and Safety Through Advanced Introspection Capabilities". In: *Proceedings of the INFORMATIK 2014*, pp. 1319–1326.

Biswas, D. et al. (2017). "Machine learning for run-time energy optimisation in many-core systems". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 1588–1592. DOI: 10.23919/DATE.2017.7927243.

Brugali, D., R. Capilla, and M. Hinchey (2015). "Dynamic Variability Meets Robotics". In: *Computer* 48.12. ISSN: 0018-9162. DOI: 10.1109/MC.2015.354.

Brugali, Davide et al. (2018). "Model-Based Development of QoS-Aware Reconfigurable Autonomous Robotic Systems". In: *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pp. 129–136.

Brutzman, D. et al. (2018). "Ethical Mission Definition and Execution for Maritime Robots Under Human Supervision". In: *IEEE Journal of Oceanic Engineering* 43.2, pp. 427–443. ISSN: 0364-9059. DOI: 10.1109/JOE.2017.2782959.

Cano, J. et al. (2016). "Automatic configuration of ROS applications for near-optimal performance". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2217–2223. DOI: 10.1109/IROS.2016.7759347.

Ceballos, Nelson David Muñoz, Jaime Alejandro Valencia, and Nelson de J. Londoño Ospina (2010). "Quantitative Performance Metrics for Mobile Robots Navigation". In: *Mobile Robots Navigation*. InTech. DOI: 10.5772/8988.

Chasparis, G. et al. (2013). "Distributed management of CPU resources for time-sensitive applications". In: *2013 American Control Conference*, pp. 5305–5312. DOI: 10.1109/ACC.2013.6580666.

Chasparis, G. C. (2015). "Reinforcement-learning-based efficient resource allocation with demand-side adjustments". In: *ECC*. DOI: 10.1109/ECC.2015.7331004.

Chen, Yu Fan et al. (2017). "Socially aware motion planning with deep reinforcement learning". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350.

Cheng, M., J. Li, and S. Nazarian (2018). "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers". In: *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 129–134. DOI: 10.1109/ASPDAC.2018.8297294.

Chikurtev, Denis (2015). "Optimizing the Navigation for Mobile Robot for Inspection by Using Robot Operating System". In:

Clemons, J. et al. (2011). "MEVBench: A mobile computer vision benchmarking suite". In: *2011 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 91–102. DOI: 10.1109/IISWC.2011.6114206.

Cucinotta, T. et al. (2010). "On the Integration of Application Level and Resource Level QoS Control for Real-Time Applications". In: *IEEE Transactions on Industrial Informatics* 6.4, pp. 479–491. ISSN: 1551-3203. DOI: 10.1109/TII.2010.2072962.

Cui, Y. et al. (2014). "A new fault tolerance method for field robotics through a self-adaptation architecture". In: *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, pp. 1–6. DOI: 10.1109/SSRR.2014.7017646.

Dressler, F. and G. Fuchs (2005). "Energy-aware operation and task allocation of autonomous robots". In: *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo '05.* Pp. 163–168. DOI: 10.1109/ROMOCO.2005.201418.

Edwards, Ross and Nelly Bencomo (2018). "DeSiRE: Further Understanding Nuances of Degrees of Satisfaction of Non-functional Requirements Trade-off". In: *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '18. Gothenburg, Sweden: ACM, pp. 12–18. ISBN: 978-1-4503-5715-9. DOI: 10.1145/3194133.3194142.

Farahnakian, F. et al. (2014). "Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning". In: *PDP*. DOI: 10.1109/PDP.2014.109.

Filieri, Antonio et al. (2017). "Control Strategies for Self-Adaptive Software Systems". In: *ACM Trans. Auton. Adapt. Syst.* 11.4, 24:1–24:31. ISSN: 1556-4665. DOI: 10.1145/3024188.

Floreano, D. et al. (1999). "Design, Control, and Applications of Autonomous Mobile Robots". In: *Advances in Intelligent Autonomous Systems*. Ed. by Spyros G. Tzafestas. Dordrecht: Springer Netherlands, pp. 159–186. ISBN: 978-94-011-4790-3. DOI: 10.1007/978-94-011-4790-3_8.

Forouher, Dariush, Jan Hartmann, and Erik Maehle (2014). "Data Flow Analysis in ROS". In: *Proceedings of ISR/Robotik 2014*.

France, R. and B. Rumpe (2007). "Model-driven Development of Complex Software: A Research Roadmap". In: *Future of Software Engineering (FOSE '07)*, pp. 37–54. DOI: 10.1109/FOSE.2007.14.

Georgas, John C. and Richard N. Taylor (2009). "Policy-Based Architectural Adaptation Management: Robotics Domain Case Studies". In: *Software Engineering for Self-Adaptive Systems*. Ed. by Betty H. C. Cheng et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 89–108. ISBN: 978-3-642-02161-9. DOI: 10.1007/978-3-642-02161-9_5.

Gherardi, L. and N. Hochgeschwender (2015). "RRA: Models and tools for robotics run-time adaptation". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1777–1784. DOI: 10.1109/IROS.2015.7353608.

Gspandl, S. et al. (2012). "A dependable perception-decision-execution cycle for autonomous robots". In: *2012 IEEE International Conference on Robotics and Automation*, pp. 2992–2998. DOI: 10.1109/ICRA.2012.6225078.

Guiochet, Jérémie, Mathilde Machin, and Hélène Waeselynck (2017). "Safety-critical advanced robots: A survey". In: *Robotics and Autonomous Systems* 94, pp. 43–52. DOI: 10.1016/j.robot.2017.04.004.

Hamza, Ameer and Nora Ayanian (2017). "Forecasting Battery State of Charge for Robot Missions". In: *Proceedings of the Symposium on Applied Computing*. SAC '17. Marrakech, Morocco: ACM. ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019705.

Hellmund, A. et al. (2016). "Robot operating system: A modular software framework for automated driving". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1564–1570. DOI: 10.1109/ITSC.2016.7795766.

Hernandez-Sosa, D. et al. (2005). "Runtime self-adaptation in a component-based robotic framework". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2700–2705. DOI: 10.1109/IROS.2005.1545578.

Hireche, C. et al. (2018). "BFM: a Scalable and Resource-Aware Method for Adaptive Mission Planning of UAVs". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6702–6707. DOI: 10.1109/ICRA.2018.8460944.

Ho, D. et al. (2018). "Towards a Multi-mission QoS and Energy Manager for Autonomous Mobile Robots". In: *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pp. 270–273. DOI: 10.1109/IRC.2018.00057.

Hoffmann, Henry et al. (2011). "SEEC: A framework for self-aware management of multicore resources". In:

Holz, Dirk, Luca Iocchi, and Tijn van der Zant (2013). "Benchmarking Intelligent Service Robots through Scientific Competitions: The RoboCup@Home Approach". In: *AAAI Spring Symposium: Designing Intelligent Robots*.

Hu, B. et al. (2017). "Cloudroid: A Cloud Framework for Transparent and QoS-Aware Robotic Computation Outsourcing". In: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 114–121. DOI: 10.1109/CLOUD.2017.23.

Huang, Jeff et al. (2014). "ROSRV: Runtime Verification for Robots". In: *Runtime Verification*. Ed. by Borzoo Bonakdarpour and Scott A. Smolka. Cham: Springer International Publishing, pp. 247–254. ISBN: 978-3-319-11164-3.

Inglés-Romero, Juan F. et al. (2013). "Dealing with Run-Time Variability in Service Robotics: Towards a DSL for Non-Functional Properties". In: *CoRR* abs/1303.4296.

Islam, F. M. M. u. et al. (2015). "A Framework for Learning Based DVFS Technique Selection and Frequency Scaling for Multi-core Real-Time Systems". In: *IEEE HPCC*. DOI: 10.1109/HPCC-CSS-ICESS.2015.313.

Jaiem, Lotfi et al. (2016a). "A Step Toward Mobile Robots Autonomy: Energy Estimation Models". In: *Towards Autonomous Robotic Systems: 17th Annual Conference, TAROS 2016, Sheffield, UK, June 26–July 1, 2016, Proceedings*. Ed. by Lyuba Alboul, Dana Damian, and Jonathan M. Aitken. Cham: Springer International Publishing, pp. 177–188. ISBN: 978-3-319-40379-3. DOI: 10.1007/978-3-319-40379-3_18.

Jaiem, Lotfi et al. (2016b). "Toward performance guarantee for autonomous mobile robotic mission: An approach for hardware and software resources management". In: *Annual Conference Towards Autonomous Robotic Systems*. Springer, pp. 189–195.

Kabir, A. M. et al. (2017). "A systematic approach for minimizing physical experiments to identify optimal trajectory parameters for robots". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 351–357. DOI: 10.1109/ICRA.2017.7989045.

Kato, S. et al. (2015). "An Open Approach to Autonomous Vehicles". In: *IEEE Micro* 35.6, pp. 60–68. ISSN: 0272-1732. DOI: 10.1109/MM.2015.133.

Kephart, J. O. and D. M. Chess (2003). "The vision of autonomic computing". In: *Computer* 36.1, pp. 41–50. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1160055.

Kim, Dongsun and S. Park (2009). "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software". In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 76–85. DOI: 10.1109/SEAMS.2009.5069076.

Krupitzer, Christian et al. (2015). "A Survey on Engineering Approaches for Self-adaptive Systems". In: *Pervasive Mob. Comput.* 17.PB, pp. 184–206. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2014.09.009.

Kunze, L. et al. (2018). "Artificial Intelligence for Long-Term Robot Autonomy: A Survey". In: *IEEE Robotics and Automation Letters* 3.4, pp. 4023–4030. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2860628.

Lee, Malrey, Mahmoud Tarokh, and Matthew Cross (2010). "Fuzzy logic decision making for multi-robot security systems". In: *Artificial Intelligence Review* 34.2, pp. 177–194. ISSN: 1573-7462. DOI: 10.1007/s10462-010-9168-8.

Li, Y. et al. (2016). "Toward QoS-Aware Cloud Robotic Applications: A Hybrid Architecture and Its Implementation". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pp. 33–40. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0028.

Lillicrap, Timothy P et al. (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

Lim, Harold, Aman Kansal, and Jie Liu (2011). "Power Budgeting for Virtualized Data Centers". In: *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'11. Portland, OR: USENIX Association, pp. 5–5.

Liu, J. and Y. Chen (2019). "HAP: A Hybrid QoS Prediction Approach in Cloud Manufacturing combining Local Collaborative Filtering and Global Case-based Reasoning". In: *IEEE Transactions on Services Computing*, pp. 1–1. ISSN: 1939-1374. DOI: 10.1109/TSC.2019.2893921.

Liu, Ning et al. (2017). "A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 372–382.

Macías-Escrivá, Frank D. et al. (2013). "Self-adaptive systems: A survey of current approaches, research challenges and applications". In: *Expert Systems with Applications* 40.18, pp. 7267 –7279. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2013.07.033.

Maggio, Martina et al. (2012). "Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems". In: *ACM Trans. Auton. Adapt. Syst.* 7.4, 36:1–36:32. ISSN: 1556-4665. DOI: 10.1145/2382570.2382572.

Mao, Hongzi et al. (2016). "Resource Management with Deep Reinforcement Learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets

'16. Atlanta, GA, USA: ACM, pp. 50–56. ISBN: 978-1-4503-4661-0. DOI: 10.1145/3005745.3005750.

Mei, Yongguo et al. (2006). "Deployment of mobile robots with energy and timing constraints". In: *IEEE Transactions on Robotics* 22, pp. 507–522.

Mishra, Nikita et al. (2018). "CALOREE: Learning Control for Predictable Latency and Low Energy". In: *ASPLOS*.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529.

MobileRobots (2006). *Pioneer 3 Operations Manual with MobileRobots Exclusive Advanced Robot Control & Operations Software*.

Morin, B. et al. (2009). "Models@ Run.time to Support Dynamic Adaptation". In: *Computer* 42.10, pp. 44–51. ISSN: 0018-9162. DOI: 10.1109/MC.2009.327.

Noureddine, Adel, Romain Rouvoy, and Lionel Seinturier (2015). "Monitoring Energy Hotspots in Software". In: *Automated Software Engg.* 22.3. ISSN: 0928-8910. DOI: 10.1007/s10515-014-0171-1.

Ondrúška, P. et al. (2015). "Scheduled perception for energy-efficient path following". In: *ICRA*. DOI: 10.1109/ICRA.2015.7139866.

Pandey, A. et al. (2016). "Hybrid Planning for Decision Making in Self-Adaptive Systems". In: *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 130–139. DOI: 10.1109/SASO.2016.19.

Pandey, P. et al. (2018). "Light-Weight Object Detection and Decision Making via Approximate Computing in Resource-Constrained Mobile Robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6776–6781. DOI: 10.1109/IROS.2018.8594200.

Papierok, Sebastian et al. (2008). "Application of reinforcement learning in a real environment using an RBF network". In: Citeseer.

Parasuraman, R. et al. (2014). "Model Based On-Line Energy Prediction System for Semi-autonomous Mobile Robots". In: *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, pp. 411–416. DOI: 10.1109/ISMS.2014.76.

Pinto, T. et al. (2019). "Multi-Agent-Based CBR Recommender System for Intelligent Energy Management in Buildings". In: *IEEE Systems Journal* 13.1, pp. 1084–1095. ISSN: 1932-8184. DOI: 10.1109/JSYST.2018.2876933.

Quigley, Morgan et al. (2009). "ROS: an open-source Robot Operating System". In: *ICRA WS. on Open Source Software*.

Rappaport, Micha (2016). "Energy-Aware Mobile Robot Exploration with Adaptive Decision Thresholds". In: *Proceedings of International Symposium on Robotics*.

Redmon, Joseph et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sadighi, A. et al. (2018). "Design methodologies for enabling self-awareness in autonomous systems". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1532–1537. DOI: 10.23919/DATE.2018.8342259.

Salehie, Mazeiar and Ladan Tahvildari (2009). "Self-adaptive Software: Landscape and Research Challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2, 14:1–14:42. ISSN: 1556-4665. DOI: 10.1145/1516533.1516538.

Shen, Hao et al. (2013). "Achieving Autonomous Power Management Using Reinforcement Learning". In: *ACM TODAES* 18.2. ISSN: 1084-4309. DOI: 10.1145/2442087.2442095.

Shevtsov, Stepan and Danny Weyns (2016). "Keep It SIMPLEX: Satisfying Multiple Goals with Guarantees in Control-based Self-adaptive Systems". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software*

Engineering. FSE 2016. Seattle, WA, USA: ACM, pp. 229–241. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950301.

Shrewsbury, B. et al. (2013). "RESPOND-R test instrument". In: *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6. DOI: 10.1109/SSRR.2013.6719369.

Singh, Amit et al. (2017). "Learning-based run-time power and energy management of multi/many-core systems: current and future trends". In: *Journal of Low Power Electronics*.

Sprunk, Christoph (2015). "Highly Accurate Mobile Robot Navigation". PhD thesis. Albert-Ludwigs-University of Freiburg, Department of Computer Science.

Sprunk, Christoph et al. (2014). "An Experimental Protocol for Benchmarking Robotic Indoor Navigation". In: *ISER*.

Sutton, Richard S. and Andrew G. Barto (2014). *Reinforcement Learning: An Introduction*.

Tai, L. et al. (2017). "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". In: *IROS*. DOI: 10.1109/IROS.2017.8202134.

Tessier, Catherine (2017). "Robots Autonomy: Some Technical Issues". In: *Autonomy and Artificial Intelligence: A Threat or Savior?* Ed. by W.F. Lawless et al. Cham: Springer International Publishing, pp. 179–194. ISBN: 978-3-319-59719-5. DOI: 10.1007/978-3-319-59719-5_8.

Thomas, S. et al. (2014). "CortexSuite: A synthetic brain benchmark suite". In: *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 76–79. DOI: 10.1109/IISWC.2014.6983043.

Twigg, J. N., J. M. Gregory, and J. R. Fink (2016). "Towards online characterization of autonomously navigating robots in unstructured environments". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1198–1205. DOI: 10.1109/IROS.2016.7759201.

Veres, S M (2011). "Knowledge of machines: review and forward look". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 226.1, pp. 3–10. DOI: 10.1177/0959651811408502.

Veres, S M et al. (2011). "Autonomous vehicle control systems: a review of decision making". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 225.2, pp. 155–195. DOI: 10.1177/2041304110394727.

Watkins, Christopher J.C.H. et al. (1992). "Technical Note: Q-Learning". In: *Machine Learning* 8.3. ISSN: 1573-0565. DOI: 10.1023/A:1022676722315.

Weisz, J. et al. (2016). "RoboBench: Towards sustainable robotics system benchmarking". In: *IEEE ICRA*. DOI: 10.1109/ICRA.2016.7487514.

Yan, Zhi, Nicolas Jouandeau, and Arab Ali Cherif (2013). "A Survey and Analysis of Multi-Robot Coordination". In: *International Journal of Advanced Robotic Systems* 10.12, p. 399. DOI: 10.5772/57313.

Zhang, Bin et al. (2014). "Autonomous vehicle battery state-of-charge prognostics enhanced mission planning". In: 5.

Zhang, Wei, Yung-Hsing Lu, and Jianghai Hu (2009). "Optimal solutions to a class of power management problems in mobile robots". In: *Automatica* 45.4, pp. 989–996. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2008.11.004.

Zhao, P. et al. (2018). "A deep reinforcement learning framework for optimizing fuel economy of hybrid electric vehicles". In: *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 196–202. DOI: 10.1109/ASPDAC.2018.8297305.

Zhao, T. et al. (2017). "A Reinforcement Learning-Based Framework for the Generation and Evolution of Adaptation Rules". In: *IEEE ICAC*. DOI: 10.1109/ICAC.2017.47.