Real-time and energy constrained cobotic systems

Mohamed Irfanulla Mohamed Abdulla

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

THESE DE DOCTORAT

NANTES UNIVERSITE

ECOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies du numérique, de l'Information et de la Communication
Spécialité : *MaSTIC-Informatique*

Par
Mohamed Irfanulla MOHAMED ABDULLA

Systèmes cobotiques temps réel sous contraintes d'énergie

Thèse présentée et soutenue à IUT de Nantes – Campus de Carquefou, le 11/10/2023
Unité de recherche : UMR 6004, Laboratoire des Sciences du Numérique de Nantes (LS2N)

Rapporteurs avant soutenance :

Patrick Bonnin Professeur des Universités, Université de Versailles-St-Quentin-en-Yvelines
Ahmad Hably Maître de conférences HDR, INP Grenoble

Composition du Jury :

Président :	Frank Singhoff	Professeur des Universités, Université de Bretagne Occidentale
Examinateurs :	Patrick Bonnin	Professeur des Universités, Université de Versailles-St-Quentin-en-Yvelines
	Ahmad Hably	Maître de conférences HDR, INP Grenoble
Dir. de thèse :	Maryline Chetto	Professeure des Universités, Nantes Université
Co-encadrants :	Audrey Queudet	Maîtresse de conférences HDR, Nantes Université
	Lamia Belouaer	Ingénieure-Docteure, Société E-COBOT, Carquefou

Invité(s)
Sébastien Ecault CEO, Société E-COBOT, Carquefou

Table of contents

List of figures

List of tables

Chapter 1

General Context

As robotic systems become more prevalent in numerous aspects of our society, they are designed to support or supplant human involvement in a wide range of tasks, such as industrial operations. A robotic system is an intricate assembly of components – *hardware* and *software* for managing its operations – that need to be flawlessly integrated to ensure the robotic system functions as intended. In pursuit of a world driven by robotics, academic research, industry, and open-source solutions are working together to offer cost-effective and efficient options for creating, evolving, and operating robotic systems.

Ahmad and Babar (2016) conducted an extensive analysis of research trends within the robotics community, emphasizing that researchers and practitioners are progressively focusing on utilizing software engineering methodologies to simplify complexities and boost efficiency in modeling, developing, maintaining, and evolving robotic systems in a cost-effective manner. Researchers from diverse fields (such as robotics, software engineering, industrial engineering, and artificial intelligence) have employed architectural models to design, rationalize, and construct robotic software. Although robotic hardware modeling and design pose significant challenges in the research community, these issues are systematically tackled by the software system. To preserve the integrity of the software system and adhere to ISO standards, industrial practitioners must follow guidelines such as ISO 25010[1], which is titled "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – *System and software quality models*." This standard outlines models with characteristics and sub-characteristics for both *system* product quality and *software* quality in use, along with practical advice on employing these quality models.

ISO 25010 Standard Overview

ISO/IEC 25010 categorizes software quality into two primary dimensions:

1. *Product Quality*: A product quality model comprising eight characteristics (further divided into sub-characteristics) that pertain to the static properties of

[1] https://iso25000.com/index.php/en/

software and dynamic properties of the software or computing system. Figure A.1 illustrates the product quality model characteristics.

2. *Quality in use*: The quality in use model composed of five characteristics (some of which are further sub-divided into sub-characteristics) that relate to the outcome of interaction when a product is used in a particular context of use. Figure A.2 shows the quality in use model.

These models' characteristics and sub-characteristics offer a consistent vocabulary for defining, measuring, and assessing system and software product quality. Additionally, they provide a set of quality characteristics that can be compared to stated quality requirements to ensure completeness.

Despite the growing importance of software in robotics, current software engineering (SE) practices are considered inadequate, frequently resulting in error-prone software that is difficult to maintain and evolve. García et al. (2020) surveyed and interviewed industrial practitioners in the robotics domain regarding the current state of SE. They discovered that 90% of participants answered "Don't know" about *deterministic* execution, a widely recognized and addressed term in the *real-time computing system community*. In essence, *determinism* ensures that the same input consistently results in the same output. This dissertation will concentrate on employing approaches used by the *real-time computing* society within a robotic system, specifically focusing on *autonomous mobile robots*. Among all the categories in the software system model for mobile robotic systems, *resource constraints* are the primary factor that affects the functionality of the mobile robotic system.

Energy constraints in mobile robotics

This dissertation primarily aims to address the crucial constraint of energy management in mobile robot, a challenge that has substantial implications for the successful operation of the device. Mobile robots are inherently limited by their finite energy supply, and any inefficiencies in energy utilization can considerably hinder their performance, potentially compromising their autonomous function. These issues are further compounded by the limitations of the processing platform's computing power. The specific objective of this dissertation is to investigate and propose solutions for these challenges, with a special emphasis on energy management for mobile robots. The purpose is to enhance their operational efficiency and ensure the effective execution of their autonomous function.

1.1 Dissertation Goal

This dissertation aims to address the identified open challenges faced by industrial mobile robotic systems. These open challenges are divided into three distinct research themes, as illustrated in Figure 1.1.

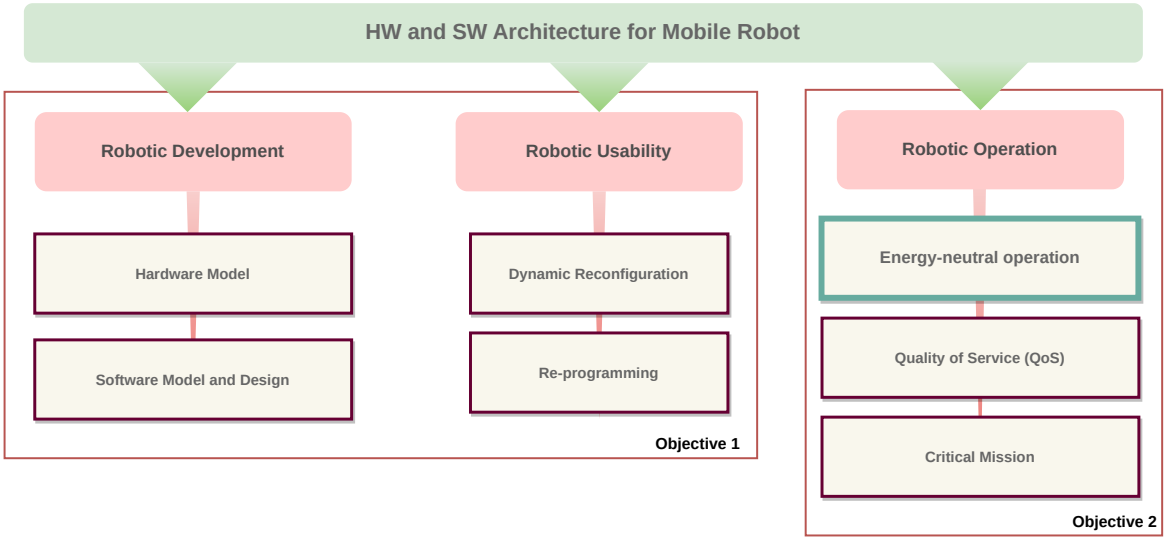


Figure 1.1 Three main themes which are the goals of this dissertation. Highlighted with energy-neutral robotic operation as the primary goal.

1. **Robotic Development**: This theme focuses on the system architecture that supports, adapts, and runs the software architecture, exhibiting performance efficiency. It involves identifying the most suitable system and software architectures to satisfy the product quality model.

   (a) *Hardware Model*: This aspect involves identifying an appropriate processing platform that is capable of handling the computational load and supporting the software architecture. Moreover, it is crucial to meet the peripheral requirements; that is, the computing platform must contain the necessary peripherals to interface with the components comprising a mobile robot.

   (b) *Software Model and Design*: We propose a software design for the computing system that exhibits deterministic behavior, ensuring the functional correctness of the system.

2. **Robotic Usability**: This theme concentrates on the system's usability with human interaction. It builds upon robotic development, aiming to provide an easy-to-use system design that satisfies user requirements.

(a) *Dynamic Reconfiguration*: Specifically, this refers to the ease with which users can reconfigure the system to meet specific requirements without the need to reboot or reprogram.

(b) *(Re)Programming*: The system design should accommodate users or integrators to (re)program the system without physically reaching the system.

3. **Robotic Operation**: Robotic operation is related to solving the problem of functional and performance efficiency in Figure A.1 of the system. The primary focus of this thesis is to meet system requirements by addressing the energy resource constraints. We define three major sub-themes under robotic operation:

(a) *Energy-neutral*: This term, relatively new in the context of robotic systems, focuses on incorporating energy harvesting techniques into mobile robotic systems. This approach allows the robot to recharge its energy storage unit without relying on a charging station. Careful energy management is crucial for making decisions about the robot's activities so as to prevent energy storage depletion, ultimately leading to the so called *energy-neutral* operation.

(b) *Quality of Service (QoS)*: In the context of robotic applications, QoS represents a critical performance metric that orchestrates mission scheduling under the constraints of time and energy. This implies that QoS ensures the use of resources while guaranteeing that missions are completed within their respective deadlines.

(c) *Critical Mission*: Industrial robots are often assigned specific tasks, known as critical missions. The successful execution of these tasks is crucial to the operation of the industry they serve. Simultaneously, these robots must operate within defined energy constraints, making efficient use of energy source to prevent disruption in service. Balancing the successful completion of critical missions with the necessity of energy conservation is a high-priority challenge in industrial robotics.

"Energy neutrality for mobile robots refers to the state in which a robot's energy consumption is perfectly balanced by the energy it harvests or generates from ambient sources, ensuring continuous operation without depleting its energy resources, and ultimately enhancing its performance, longevity, and mission success."

In approaching these three themes, our goal is to create a deterministic and energy-efficient mobile robotic system. To ensure that the mobile collaborative robotic system meets industrial demands, it is vital to thoroughly examine and integrate these themes. We have structured the themes hierarchically. We begin with selecting a suitable framework, then we investigate operation sub-themes, and finally we analyze development and usability. This strategy will foster a complete understanding of the challenges and opportunities in designing a robust and efficient mobile collaborative robotic system for industrial use.

1.2 Problem Statement

In this research, we focus on industrial mobile robotic architecture, as highlighted in the motivation. Mobile robots must guarantee *deterministic execution* while managing *time* and *energy constraints*. Consequently, the requirements are categorized into two sub-themes: *functional requirements* (FR), which pertain to the primary functionality of mobile robots, such as navigation, and *non-functional requirements* (NFR), which relates to deterministic, performance, reliability, and energy utilization.

Mobile robots must respond to dynamic environments in real-time to ensure safe operation. By integrating real-time systems, robots can manage tasks with deterministic execution and adherence to *deadlines*, which are crucial for system stability, safety, and responsiveness. Energy efficiency is vital for mobile robots due to limited power supply. In this dissertation, we investigate strategies for optimizing energy utilization without compromising real-time performance. We examine various real-time system aspects, such as energy-aware scheduling algorithm, and its applicability to mobile robot applications. Additionally, we explore challenges and opportunities in implementing energy-aware real-time systems in mobile robots, assessing trade-offs between performance, reliability, and power consumption.

We outline the major challenges:

1. Development of a mobile robotic architecture, i.e., providing an architecture that satisfies both FR and NFRs.

2. Support for real-time task dependency for realistic applications, i.e., enhancement of energy-aware scheduling algorithms for real-time energy harvesting systems to incorporate task dependency applications.

3. Implementation of energy-aware scheduling algorithms within a real-time operating system, i.e., achieving energy-neutral operation.

Ultimately, this research aims to provide valuable insights and contribute to the development of efficient, reliable, and energy-aware robotic architecture capable of meeting the non-functional requirements while satisfying their functional requirements. Although our focus is on addressing the challenges faced by the specific industrial mobile robot developed by E-COBOT, named HUSKY[2], the contributions made in this work can be extended to other robotic systems with appropriate modifications.

Challenges

Numerous challenges have been identified through the review of the current state of the art and during the course of this research. In this section, we will highlight these challenges, emphasizing their importance as the primary focus of this dissertation. The ultimate goal of our contributions is to address the goals listed in Section 1.1, which involves designing and developing a mobile robotic architecture that satisfies robotic operations through an energy-aware and deterministic approach, meeting industrial standard requirements.

Challenge 1

Designing the Hardware/Software Architecture for a Mobile Robot with Real-Time Capabilities

Designing the *hardware* and *software* architecture of a mobile robots with *real-time* capabilities requires careful consideration of various factors that influence the overall performance and reliability. These factors include selecting suitable processing platform that can handle real-time processing, computational load, and compatibility with the chosen operating system and middlewares. Additionally, the use of a Real-Time Operating System (RTOS) that provides deterministic execution and allows for effective task scheduling is essential. Implementing a suitable task scheduling algorithm that meets the real-time requirements of the robotic device and developing a modular software architecture that separates the device into independent, interchangeable components or modules are also crucial. This modular approach enables easier maintenance, testing, and future upgrades of the robotic device.

Many mobile robot architectures utilize a distributed computing platform. Here, separate units manage hardware control, while others handle computations for navigation. However, this approach can lead to delays in data processing and control, which can limit the real-time capabilities of the robot. For instance, in traditional

[2]https://e-cobot.com/en/husky-smart-mobile-cobot-3/

architectures, sensor data acquisition and processing may be handled by separate units, introducing latency and making it challenging to ensure timely and accurate decision-making. This latency can lead to safety issues in critical applications, such as in industrial environments, where robots are expected to operate in real-time to avoid accidents and ensure productivity.

Challenge 2

Dynamic Reconfigurability and Re-programmability for Enhanced Usability According to User Requirements

Dynamic reconfigurability and re-programmability can be challenging in mobile robotic architectures. Physically accessing the robot to make changes can lead to a complete shutdown, requiring a complete restart, which is not ideal for industrial integration. This issue can be critical, especially in applications that require fast reconfiguration or redeployment of the robot's tasks. Therefore, there is a need for a thorough guide to designing mobile robots with real-time capabilities. Such a guide should ensure dynamic reconfigurability and re-programmability, enabling real-time changes in the robot's tasks without requiring a complete shutdown or access to device physically.

Challenge 3

Incorporating Energy-Aware Scheduling Algorithm with Shared Resource Constraints

One of the significant challenges related to mobile robots is the *energy constraint*, as these systems carry limited energy sources. *Energy management* in the architecture of mobile robots is essential. The focus of our research is on *energy-aware real-time task scheduling*, which enables the robotic device to function in an energy-neutral manner. An *energy-neutral operation* balances its energy demand with the energy scavenged from renewable sources. This approach is superior to energy-saving approaches, which still rely on conventional energy sources where the mobile robots must rest at charging stations. An energy-neutral operation makes the robot completely autonomous and free from charging station halts, increasing the productivity of the industrial sector.

To address this challenge, we will investigate optimal energy-aware real-time scheduling algorithm namely Earliest Deadline Harvest (ED-H) to implement within the mobile robotic architecture. However, one shortcoming of this algorithm is that it initially considers only independent real-time tasks, limiting their applicability for more realistic

applications like mobile robots. Therefore, it is essential to incorporate methods to consider dependent tasks, particularly tasks with shared resource constraints.

Challenge 4

Implementing the Optimal ED-H Energy-Aware Scheduling Algorithm within a RTOS

Implementing the energy-aware scheduling algorithm within a Real-Time Operating System (RTOS) is an open challenge. The integration of such algorithms into a RTOS will require a clear understanding of the existing scheduling policies and mechanisms. Additionally, it is crucial to ensure that the energy-aware scheduling algorithm can coexist with other scheduling policies without negatively impacting the real-time performance of the system.

During the course of the research, we have identified new challenges that must be investigated to develop a mobile robotic device capable of satisfying real-time constraints and achieving energy neutrality. The challenges previously discussed address both timing and energy constraints. However, the granularity of the energy focused on these challenges may not be completely effective for larger systems like mobile robots. In mobile robots, maximum current is often drawn when the robot is actively performing navigation tasks called missions, consuming energy at varying rates depending on the complexity and duration of the operation.

Challenge 5

Implementing the Energy-Aware Scheduling Algorithm to Ensure Mission Applications of Mobile Robots Operate within Energy Constraints while Maintaining Performance and Reliability

Ensuring energy constraints, or the efficient management and allocation of energy resources, is a significant challenge that must be addressed for the successful completion of missions. This challenge becomes more pronounced as the complexity and number of missions increase. One of the key factors contributing to the energy consumption of mobile robots is the nature of the work being performed. For example, mobile robots in warehouse automation are often required to operate continuously, picking and placing items, transporting goods, and navigating through complex warehouse layouts. These works necessitate numerous complex movements and decisions, which can lead to high energy consumption. In this context, energy constraints becomes vital not only for the successful completion of individual mission but also for maintaining the overall

efficiency and cost-effectiveness of warehouse operations. This challenge is exacerbated by the increasing complexity and number of missions, necessitating the development of energy-neutral solutions to extend the robots' performance and longevity.

Challenge 6

Estimating the Duration and Energy Consumption of Mobile Robots Missions

Harmonizing real-time intelligent mission scheduling for mobile robots presents a formidable challenge due to several interrelated factors. The first challenge arises from determining the energy consumption of the robot during a mission, which depends on the hardware design and the navigation strategies employed. The next major challenge emerges when integrating energy harvesting solutions, as the intelligent mission scheduling algorithm must rely on predictions of future energy availability during the mission, making it difficult to accurately estimate the harvested energy.

This dissertation aims to address these key challenges and contribute to the field by providing a thorough guide for designing and integrating a real-time deterministic mobile robot capable of energy-neutral operation. By tackling the complexities of intelligent mission scheduling, energy harvesting, and resource constraints, this work will serve as a valuable guide for researchers and practitioners seeking to develop energy-aware mobile robot solutions that enhance both the functional and non-functional requirements of the mobile robots. The dissertation will present a clear and well-structured analysis of each challenge, along with proposed solutions, supported by experimental results and performance evaluations. This holistic approach will demonstrate the effectiveness of the proposed solutions in achieving energy neutrality and improving the overall system performance, thereby solidifying the mobile robot's ability to meet industrial standards.

1.3 Contributions

In this dissertation, we have identified the key challenges that must be emphasized in order to develop a mobile robotic architecture that adheres to both non-functional and functional requirements, encompassing industrial standards. These challenges stem from the complex interplay between two domains: real-time systems and mobile robotics. Our approach to tackle these challenges will be structured around four concrete contributions, each focusing on a specific aspect of the problem.

Contribution 1 - Architecture

Hardware and Software Design of Industrial Mobile Robot Architecture with Real-time Capabilities.
Addresses : **Challenge 1 and 2**
In : **Chapter 4**

The first contribution of this dissertation encompasses the overall hardware and software architecture design considerations for developing a mobile robot that meets industrial requirements. This contribution will focus on an architecture that brings real-time capabilities to mobile robots. This architecture must satisfy not only the functional requirements, which are typically the requirements of mobile robots, but also the non-functional requirements that ensure the behavior and reliability of mobile robots in robust environments. Furthermore, the architectural design will incorporate dynamically configurable and re-programmable features to meet user requirements.

Contribution 2 - Theoretical Contribution

ED-H Energy-Aware Real-Time Scheduling Algorithm with Shared Resource Constraints
Addresses : **Challenge 3**
In : **Chapter 5**

The second contribution of this dissertation examines the integration of shared resource constraints in the context of energy-aware real-time scheduling algorithms. Shared resources, such as critical sections, present challenges that need to be addressed to ensure efficient management of these resources. By addressing these challenges, we aim to develop a robust scheduling scheme that accounts for resource constraints while guaranteeing time and energy constraints.

Contribution 3 - System Level

Implementation of ED-H Scheduling Algorithm under Xenomai, a real-time co-kernel for Linux distribution
Addresses : **Challenge 4**
In : **Chapter 6**

The third contribution of this dissertation focuses on the implementation of the ED-H energy-aware scheduling scheme. We will offer an extensive guide detailing the implementation of the ED-H scheduling scheme within the scheduler class of the

Xenomai-patched Linux kernel. We will discuss the specific APIs that must be used to schedule real-time tasks under the ED-H scheduling scheme. Moreover, we will present a simulation of this scheduling scheme, incorporating simulated battery and energy harvesting modules.

During our research, we identified that treating the energy-neutral concept at the operating system level of embedded computing devices in mobile robots does not have a significant impact. Therefore, we aim to address this at the application level, encompassing the missions that mobile robots typically perform. This approach will enable the energy-neutral operation of the robotic device, ensuring its operation for mission completion.

Contribution 4 - Application Level

Implementation of ED-H Scheduling Algorithm for Energy-Aware Mobile Robotic Operation
Addresses : **Challenge 5 and 6**
In : **Chapter 7**

The final contribution of this dissertation encompasses the implementation of energy-aware scheduling algorithm for mission scheduling. We will provide a complete account of the implementation of the ED-H scheduling algorithm at the application level. We will discuss the characterization of uncertainties, such as mission duration and energy estimation for the mission. We will validate the proposed algorithm through simulation and real-world experiments. An extensive guide for developers and integrators to utilize the mission scheduling algorithm will be provided. Additionally, we will examine the energy harvesting problem, its limitations, and other challenges, such as calibration procedures to estimate energy consumption.

1.4 Dissertation Overview

The rest of the dissertation is organized as follows:

Chapter 2: This chapter offers a thorough theoretical background in two domains: mobile robots and real-time systems. We present the generic architecture of mobile robots and the middleware ROS (Robot Operating System) to address functional requirements. Subsequently, we delve into the concept of Real-Time Systems, elaborating on all relevant aspects, including software architecture. This foundation provides a solid understanding for the remainder of the dissertation.

Chapter 3: This chapter presents the state-of-the-art foundation for our research in energy management for mobile robots. We encapsulate the energy-efficient approaches employed in mobile robots to achieve energy savings. Then, we delve into the concept of energy-neutral operation, detailing the optimal energy-aware scheduling algorithm ED-H. Additionally, we also discuss the energy harvesting techniques that are commonly utilized, followed by an examination of energy storage systems for mobile robots. This extensive overview sets the stage for a deeper understanding of energy management throughout the dissertation.

Chapter 4: In this chapter, we present a detailed architectural design for a robust mobile robot with real-time capabilities. Furthermore, we provide guidance for developers to identify and create real-time applications that implement deterministic mobile robot applications. We also discuss the applicability of robotic middleware, specifically ROS2, to ensure real-time constraints. This thorough exploration lays the groundwork for understanding and implementing real-time capabilities for industrial mobile robotics.

Chapter 5: In this chapter, we delve into the details of shared resource constraints, discussing the complexities associated with them. Our novel contribution within this chapter is the development of a schedulability test for the ED-H scheduling algorithm, which incorporates shared resource constraints. This in-depth examination and contribution advance our understanding of how to effectively manage shared resources by incorporating resource access protocols with energy-aware scheduling algorithm.

Chapter 6: In this chapter, we present the simulation of the ED-H scheduling algorithm. We describe the scheduling classes of the Xenomai kernel. We then provide the implementation details and limitation of energy-aware scheduling algorithm in our framework.

Chapter 7: In this chapter, we address the context of robotic missions. We outline a methodology based on decision-making techniques for scheduling robotic missions. We describe the implementation of the energy-aware scheduling algorithm for decision-making in mission scheduling. We present the simulation and real-world limitations. Furthermore, we discuss the technology barriers that must be overcome to achieve complete energy-neutral operation.

Finally, we present the conclusion of this research work and discuss future perspectives in the field. This will be followed by a summary of the dissertation in French.

Chapter 2

Background: Mobile Robots and Real-time Systems

This chapter discusses the fundamental background that serves as the foundation for our research. The topics presented are multidisciplinary, spanning from *mobile robots* to *real-time systems*, with an emphasis on *non-functional requirements* for mobile robotic systems. Section 2.1 presents an architectural overview of mobile robotic systems, covering both *functional* and *non-functional* requirements. This is followed by an examination of the mobile robot software framework in Section 2.2. Section 2.3 presents the definition of real-time systems, along with essential concepts that emphasize the *real-time workload*, which provides the real-time task model used in this research. The *processing platform* encompasses the computing system software framework designed to satisfy the non-functional requirements of the mobile robotic system. Additionally, the *scheduling algorithm* highlights the classification and details the priority-based scheduling algorithm. Finally, we conclude this chapter in Section 2.4.

2.1 Mobile Robots

Autonomous Mobile Robots (AMRs) [Siegwart et al. (2011)], equipped with diverse sensor arrays and control systems, have become pivotal in areas like manufacturing, healthcare, and exploration. First designed for navigation and artificial intelligence (AI) capabilities [Nilsson (1969)], they now play a significant role in modern manufacturing by performing repetitive tasks like assembly and material handling, thereby improving productivity and reducing costs. In particular, *mobile collaborative robots* (*cobots*), designed for Industry 4.0, work alongside humans to enhance safety and provide operational flexibility [Vitolo et al. (2022), Sorell (2022)]. The forthcoming Industry 5.0, focusing on a more human-centric, sustainable, and resilient approach, is expected to boost human-robot collaboration, with humans guiding robots in more complex processes [Maddikunta et al. (2022), Maciaszczyk et al. (2023)].

For *cobots* to succeed in the future, they must effectively learn from human input, adapt to evolving tasks, and comply with industrial standards [De Ryck et al. (2020)]. Such progress relies heavily on advancements in AI, sensing, and control systems.

Importantly, these *cobots* should also optimize resource utilization while meeting user requirements, ensuring system and software quality. Throughout this dissertation, we will use the term "mobile robots" to refer to these *collaborative robots* or *cobots*.

2.1.1 Mobile Robot Architecture

The generic architecture of an autonomous mobile robotic system, as depicted in Figure 2.1, can be categorized into four distinct subcategories:

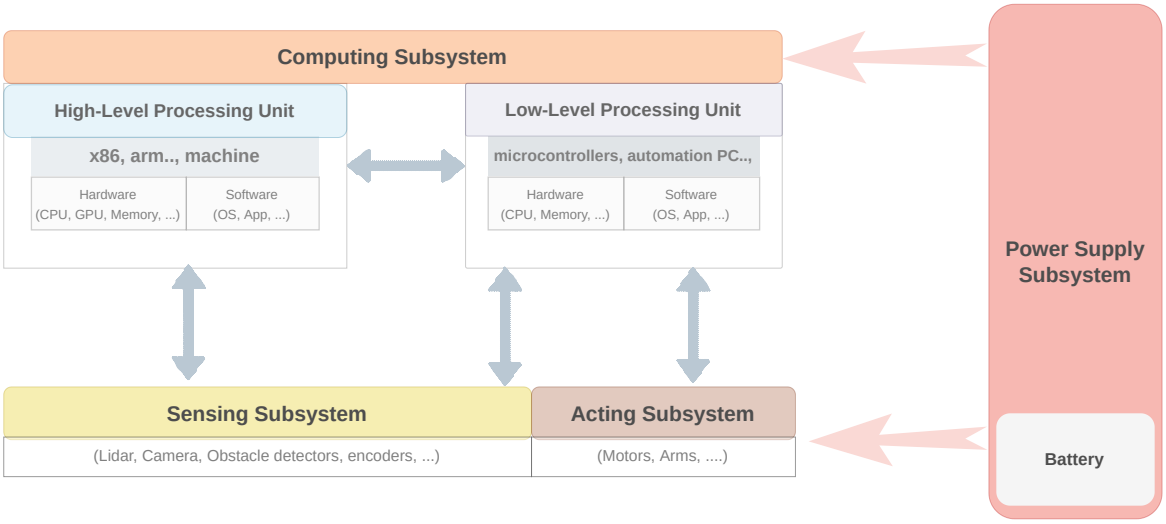


Figure 2.1 Architecture of Mobile Robotic System.

1. *Computing Subsystem*: Encompassing both *high-level* and *low-level* computing systems, this subsystem is responsible for decision-making, artificial intelligence, and controlling the robot's hardware components to facilitate interaction with the environment.

2. *Acting Subsystem*: This subsystem is responsible for the robot's *locomotion*, consisting of the actuation system. It may also interface with supplementary systems containing actuators, such as arm manipulators, to perform a diverse range of tasks.

3. *Sensing Subsystem*: By collecting and processing data from various sensors, the *sensing* subsystem enables the robotic system to perceive and understand its surroundings.

4. *Power Supply Subsystem*: As an essential component of mobile systems, the *powering* subsystem provides a finite power supply that requires periodic recharging. Batteries are the most prevalent power source for autonomous vehicles.

The core of a functional mobile robotic system is formed by these foundational systems.

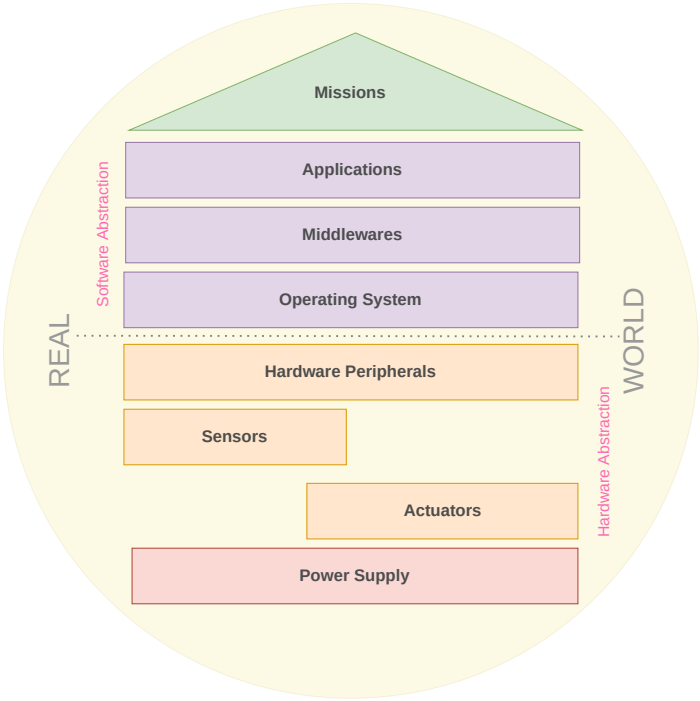


Figure 2.2 Abstraction Level of Mobile Robotic Systems.

As depicted in Figure 2.2, the abstraction levels of the mobile robotic system are divided into two categories: *Hardware* and *Software* abstractions. These levels are hierarchical, with *Missions* representing the topmost level. A mission consists of a predefined set of tasks or objectives for the robot to accomplish, serving as an encapsulator that enables other layers to interact with the real world. Each layer is dedicated to specific functionalities within the system. Reducing the system's complexity can be achieved by addressing both abstractions, rather than individual layers, resulting in a more cohesive and efficient approach to problem-solving.

During the design phase of the robot, the hardware abstraction is addressed, focusing on the robot's taxonomy, which is defined by its *degree of freedom* (e.g., *differential drive* with 2 degrees of freedom or *omnidirectional drive* robot with 3 degrees of freedom) [Siegwart et al. (2011)]. A mathematical representation of the robot is then established, encompassing the *kinematic* and *dynamic* models of the robot to maximize

the system's usefulness in terms of motion [Yun and Yamamoto (1993)]. This process aids in defining control algorithms and calculating energy consumption.

2.1.2 Mobile Robotics Software Requirements

System software architecture serves as the bridge between commercial goals and software systems. The requirements of a mobile robotic system software architecture can be categorized into two: *Functional Requirements* (FR) and *Non-Functional Requirements* (NFR).

1. **Functional Requirements**: Localization, Planning, Motion Control, etc., which are essential for the mobile robotic system to navigate and operate in the real world.

2. **Non-Functional Requirements**: Performance, Resource utilization (time and energy), Reliability, Adaptability, Interoperability, etc., are the requirements that the system/software framework should satisfy to meet user requirements.

Based on system requirements, various components are selected during the design phase. The design phase problem is prevalent in robotics research [Ahmad and Babar (2016)]. Historically, the software abstraction layer exhibited high complexity, with researchers developing their own solutions. Over time, there has been a push towards universal software architectures in the robotics community to address challenges such as software *reusability* and *lack of standardization* [García et al. (2020)]. Several generic *middleware* systems have been developed by research groups and communities, which operate on operating systems.

2.2 Mobile Robotic Framework

The paramount challenge in mobile robotics is ensuring robustness in both the software and hardware components of a system, enabling them to operate continuously without *critical failures* regardless of the context, including mission and environment [García et al. (2020)]. To address this challenge, it is essential to investigate widely accepted standards, guidelines, best practices, and methodologies for architecture, design, and operating systems of mobile robots. The question of architecture is particularly important for achieving the highest level of robot competence, termed navigation competence, which involves robust *navigation*, *data interpretation*, *localization*, and *motion control*.

A key feature of robotic middleware is its ability to bring different components together, facilitating communication and interoperability [Andreasson et al. (2023)]. Additionally, middleware provides interfaces to *sensors* and *actuators*, often requiring low-level access and operating system-specific calls. An effective robotic middleware should not only meet the requirements of a complex robotic system but also provide a well-structured *Application Programming Interface* (API) for functionality at various interaction levels. Depending on the application, users may require precise control over communication flow or a simple high-level API call.

There are a vast number of different frameworks for robotics. They are actively developed and maintained, both by the *open* source community, as well as by the *commercial* sector listed in [Elkady and Sobh (2012), Iigo-Blasco et al. (2012)]. We outline the most influential and commonly used frameworks in Table 2.1.

Middleware	Foundation Year	Open Source	Active
OROCOS [Bruyninckx (2001)]	2001	✓	✓
Player [Gerkey et al. (2001)]	2001	✓	✗
CARMEN [Montemerlo et al. (2003)]	2003	✓	✗
OpenRTM [Siekmann and Wahlster (2008)]	2008	✓	✓
ROS [Quigley et al. (2009)]	2009	✓	✓
ROS2 [Thomas D, Woodall W (2014)]	2014	✓	✓

Table 2.1 Frameworks for robotics. The table highlights the active and popular frameworks [Elkady and Sobh (2012), Iigo-Blasco et al. (2012), Andreasson et al. (2023)].

The most popular and widely-used open-source middleware in both research and industry is the *Robot Operating System* (ROS). ROS effectively satisfies the *functional requirements* of a mobile robotic system. In this work, we will consider ROS middleware, since the mobile robotic system in our company, utilizes ROS.

2.2.1 Robot Operating System (ROS)

The Robot Operating System (ROS) serves as a comprehensive operating system for service robotics, functioning as a meta-operating system that bridges the gap between an operating system and middleware [Koubaa (2016)]. It offers a suite of open-source software libraries to facilitate robotics application development. ROS streamlines the development and implementation of robotics software through its modular, *node-based* architecture, enabling the integration of complex robot behaviors, such as localization, mapping, path planning, and autonomous navigation. These capabilities are particularly relevant for mobile robots.

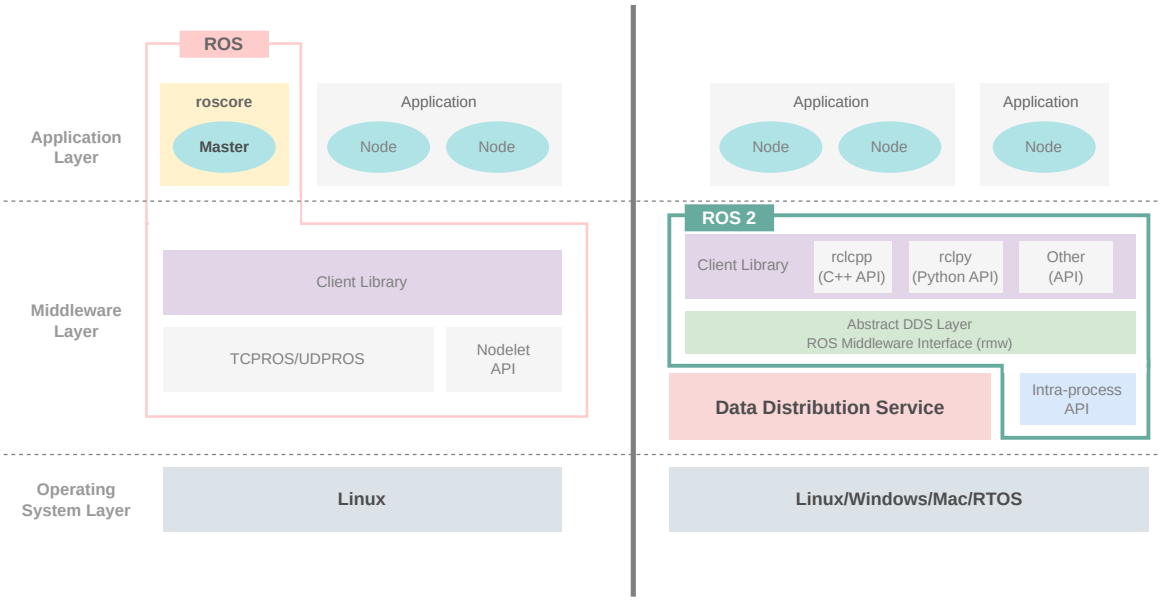


Figure 2.3 Comparison of ROS with ROS2 architecture.

ROS2[1], the successor to ROS, introduces several key improvements, including a redesigned communication layer based on the *Data Distribution Service* (DDS)[2] standard, which enhances performance, security, and Quality of Service (QoS) settings. Figure 2.3 illustrates the architecture of ROS and ROS2.

For mobile robot navigation, ROS2's advancements offer significant benefits. The improved communication layer allows for more reliable and efficient data exchange among the robot's system components, improving coordination and responsiveness during navigation tasks. ROS2's capabilities facilitate more accurate and prompt control of robot motion, essential for navigating dynamic environments safely and efficiently. Furthermore, ROS2's modularity and robustness empower developers to create scalable, *fault-tolerant* solutions for various applications and scenarios.

In the subsequent section, we will explore navigation concepts, emphasizing how ROS2's features and enhancements can be effectively leveraged to boost mobile robots' navigation capabilities. We will examine localization, mapping, and path planning techniques. We will discuss these components within the ROS2 framework to achieve advanced, reliable navigation performance.

[1]https://docs.ros.org/en/humble/index.html
[2]https://fast-dds.docs.eprosima.com/en/latest/fastdds/ros2/ros2.html

2.2.2 Mobile Robot Navigation

Mobile robot navigation is a crucial element of autonomous robot operation, enabling robots to move purposefully and efficiently within their environment. To successfully navigate, a robot must address several challenges, similar to how humans navigate from one place to another. These challenges can be broken down into four main components:

1. *Mapping*: The robot needs to create a *representation* of its *environment* to understand the context in which it operates.

2. *Localization*: The robot must determine its *position* within the *environment* to make informed navigation decisions.

3. *Path Planning*: The robot must develop an optimal *route* from its current position to its *destination*, considering any constraints or *obstacles*.

4. *Robot Control and Obstacle Avoidance*: The robot must *execute* the planned *path* while adjusting its motion to *avoid obstacles* and ensure smooth navigation.

Building these components from scratch can be complex and time-consuming. ROS2, provides pre-built package Nav2 [Macenski et al. (2020)] that simplifies the development process and facilitates efficient navigation. These components work in harmony to ensure optimal performance of mobile robots during navigation tasks, which is further facilitated by the *lifecycle* manager of ROS2. The lifecycle manager provides a structured approach to managing the various components and their states, ensuring a consistent and well-organized execution of the navigation process. By effectively handling the initialization, configuration, activation, and deactivation of each component, the ROS2 lifecycle manager contributes to the overall reliability, stability, and efficiency of the mobile robot's navigation system.

An architectural overview of ROS reveals that it does not function as a traditional operating system responsible for *process management* and *scheduling*. Instead, it operates on top of the host operating system of the computing system. As shown in Figure 2.3, ROS2 can run on a variety of operating systems, including Linux, Windows, Mac, and RTOS. In this work, we will employ the Linux operating system due to its compatibility with the industrial framework. Intrinsically, Linux does not display real-time characteristics such as deterministic execution. As a result, understanding the concept of a real-time system is essential for adapting the software architecture of the robotic system to satisfy both the Functional Requirements (FR) and Non-Functional Requirements (NFR).

2.3 Real-time Systems: Deterministic Performance

The term "*real-time*" is frequently employed in various application fields and is subject to different interpretations, not always accurate. Often, a control system is considered to operate in real-time if it can quickly respond to external events. According to this interpretation, a system is deemed real-time if it is fast. However, the term "*fast*" is relative and fails to encapsulate the primary properties that define these systems. Instead of being merely fast, a real-time computing system should prioritize predictability. Stankovic (1988) provides a comprehensive definition of a real-time system, stating that:

> *"A real-time system is a system in which the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced".* [Stankovic (1988)]

Temporal determinism in a system is the primary requirement to ensure time-predictable task execution. Although *throughput* and *low latencies* are desirable, they are considered secondary attributes. Consequently, "real-time" does not imply computing at the *fastest possible speed* but rather computing *as fast as required*.

Determinism in real-time systems is defined by the system's consistent and predictable behavior, especially concerning task *execution* and *response* times to events. In deterministic real-time systems, tasks are structured to consistently meet *deadlines* by adhering to predefined time constraints.

Timing characteristics hold significant importance in real-time computing systems, particularly for specific applications such as mobile robot operation. These systems must complete computations within a predefined time interval, which is characterized by a release time and a deadline. These critical parameters dictate the system's behavior, and missed deadlines can lead to severe consequences, including degraded performance, system failure, or even catastrophic failure.

When developing real-time computing systems, it is crucial to carefully assess timing requirements and ensure that the system will operate within the defined constraints. In subsequent sub-sections, we will explore real-time concepts in greater details, highlighting their importance in the design and implementation of real-time principles in robotics applications, ultimately contributing to more accurate and reliable performance.

2.3.1 Fundamental Concepts of Real-Time Systems

Real-time systems can be analyzed and optimized based on three primary aspects that need to be considered in order to ensure that the system meets its requirements and behaves as expected.

- **Real-time Workload**: This aspect pertains to the *computational workload* of the system, encompassing *task* parameters such as release time, execution time, and deadline. Additionally, the workload model describes the classification, types, and dependencies of real-time tasks.
- **Processing Platform**: This aspect pertains to the combination of *hardware* and *software* elements that make up the platform where the workload tasks are performed. Essential resources include processor(s) (CPUs), memory, cache, and the connections among them. The platform's architecture is of great importance, as its performance has a direct effect on the system's timing characteristics.
- **Scheduling Algorithm**: This aspect pertains to the method employed in a system with multiple tasks and varying deadlines, where numerous jobs might be ready for execution simultaneously and require completion as soon as possible to meet their deadlines. In such situations, a *scheduling algorithm* is necessary to determine which jobs are executed on the processing platform at any given time. The selection of a scheduling policy is a crucial factor that influences the temporal behavior of the system.

Optimizing real-time systems involves analyzing and refining these three aspects to ensure that the system can meet its performance objectives. This process requires a thorough understanding of the system's workload model, processing platform, and scheduling algorithm, as well as the interactions between them. By optimizing these three aspects, it is possible to improve the predictability, reliability, and performance, which contribute to the Non-Functional Requirements (NFR) of the system.

2.3.2 Real-time Workload

The Workload of a real-time system is a formal representation of the computational workload of the system, including the *tasks* that must be executed, their timing characteristics, and their dependencies. This model is a critical aspect of real-time system design because it provides insight into the characteristics of the workload and

enables the system to be analyzed and optimized for performance, reliability, and safety.

Tasks in a real-time system are the individual units of work that the computing system is designed to perform. Most commonly they are referred to as *threads* in the operating system. They are typically composed of a set of instructions that are executed by the system's processor. They can involve input/output operations, data processing, and communication with other systems. Examples are obtaining the obstacle sensor value and sending speed data to control the motors.

Generic Real-Time Task

Applications that implement real-time tasks require to specify these tasks through distinct parameters. Generally, a *real-time task*, denoted as τ_i, may generate an unbounded sequence of *jobs* throughout its lifespan. The j^{th} job associated with task τ_i can be denoted as $\tau_{i,j}$. To effectively characterize the task and its jobs, multiple parameters are utilized:

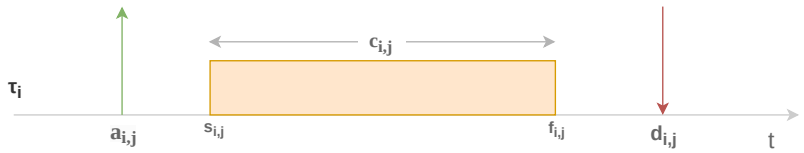


Figure 2.4 Typical parameters of a Real-Time Task from Buttazzo (2011).

- *Arrival/Release time ($a_{i,j}$)*: This is the moment when a job becomes ready for execution (triggered by some event or condition).
- *Computation time ($c_{i,j}$)*: This represents the time duration required by the processor to execute the job completely without interruption. The actual computation time of a job is by definition always less than or equal to the WCET (Worst-Case Execution Time) of the task which gives an upper bound for the execution time of any job of the same task.
- *Start time ($s_{i,j}$)*: This is the time at which a job begins its execution.
- *Finishing time ($f_{i,j}$)*: This is the time at which a job completes its execution.
- *Relative Deadline (D_i)*: The time interval within which the job execution should be completed in relation to its release time and absolute deadline. Typically, a

single value is assigned for every job of a certain task, denoted as D_i, as it refers to task τ_i.

- *Absolute Deadline ($d_{i,j}$)*: The specific instant in time by which a job should be completed to maintain the timeliness properties of the system. It is calculated based on the relative deadline and the release time:: $d_{i,j} = a_{i,j} + D_{i,j}$.
- *Laxity or Slack time ($L_{i,j}$)*: $L_{i,j} = a_{i,j} + D_{i,j} - C_{i,j}$ refers to the maximum time a job of a task can be delayed upon activation while still completing within its deadline.
- *Response time ($R_{i,j}$)*: The difference between the finishing time and the release time: $R_{i,j} = f_{i,j} - a_{i,j}$.

Some of the above parameters are illustrated in Figure 2.4. Typically, upward arrows represent new arrivals, and downward arrows indicate absolute deadlines.

Classification of Real-Time Tasks

The primary distinction between *real-time* and *non-real-time tasks* at the process level is the presence of a (relative) deadline for completing the execution of the tasks. The *deadline signifies the maximum allowable time for task completion.* Data resulting from the execution of tasks are crucial for controlling the robot. When a task misses its deadline, the *usefulness* or *utility* of the output result changes. This variable utility value can be represented by a function that varies along time after a deadline miss. Figure 2.5 illustrates the utility function $u(t)$ for different task classifications, including non-real-time tasks i.e., tasks with no deadline constraints.

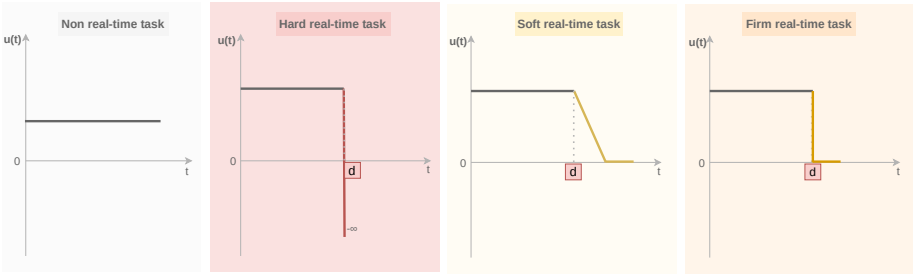


Figure 2.5 Classification of task in terms of utility function adapted from Buttazzo (1997). The real-time tasks have *deadline* constraint d.

1. *Hard real-time task*: A task is considered *hard* if delivering results after its deadline could lead to catastrophic consequences for the controlled system. A hard task's value is only realized if completed within its deadline; otherwise, the value may be considered $-\infty$ in many situations, as missing the deadline could *jeopardize* the entire system.

2. *Soft real-time task*: A task is deemed *soft* if producing results after its deadline still offers some utility for the system, although the value may decrease over time, resulting in performance degradation.

3. *Firm real-time task*: A task is classified as *firm* if producing results after its deadline is useless for the system and does not cause any damage. These tasks do not jeopardize the system but offer zero value if completed after their deadline.

A mobile robotic system may comprise tasks from all these categories. The real-time computing system must be designed to manage them using various strategies. For instance, tasks associated with actuator control and sensor data reading should be treated as hard real-time tasks, primarily due to their safety-critical nature. By treating these tasks as such, the system can ensure proper functioning and maintain the required level of safety and reliability.

Real-time tasks can be categorized based on the consistency of their activation, specifically as *periodic*, *aperiodic*, or *sporadic* tasks. Figure 2.6 showcases the three types and differentiates their activation patterns. Periodic tasks are composed of an infinite series of identical actions, known as *instances* or *jobs*, which are activated at a consistent rate. In practical situations, a periodic process can be described by its *offset* Φ_i (first periodic instance), *worst-case execution time* C_i, *period* T_i, and *relative deadline* D_i. Conversely, aperiodic tasks involve an infinite series of *identical jobs* that have irregularly spaced activation's. A sporadic task, a type of aperiodic task, is characterized by *consecutive jobs* separated by a *minimum inter-arrival time* (T_i).

In this dissertation, we will focus exclusively on *periodic* tasks. We will delve into the detailed characterization of periodic tasks, which will be consistently utilized throughout this dissertation.

Periodic Task Model

We will employ the periodic task model widely adopted in real-time computing research, as presented in the seminal work by Liu and Layland (1973). We consider a set of tasks denoted by τ, comprised of n tasks, where $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. The j^{th} job of a

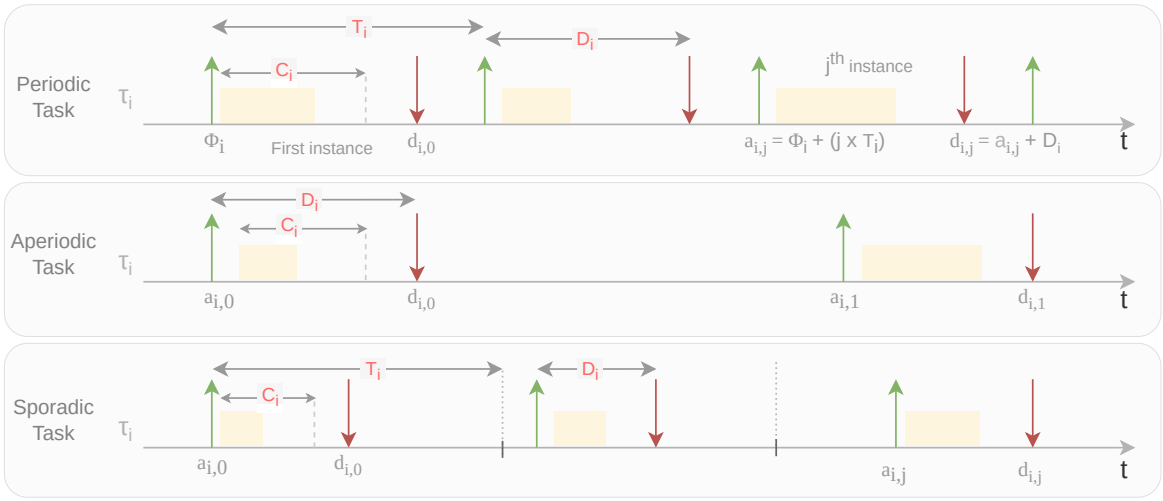


Figure 2.6 Classification of real-time tasks based on activation adapted from Buttazzo (2011). Sequence of instance for a *periodic* task, an *aperiodic* task, and a *sporadic* task.

periodic task τ_i will be denoted as $\tau_{i,j}$. The activation time of the first job ($\tau_{i,0}$) of the periodic task is represented by the offset Φ_i. If all tasks have the same offset, they are called *synchronous* tasks; otherwise, if they have different activation times, they are *asynchronous* tasks. The *release time* of the j^{th} job of the task is given by $a_{i,j} = \Phi_i + (j.T_i)$, where *period* ($T_i$) is the time interval between two consecutive releases of the task. In most real-time system research, the computation time of the task τ_i is considered as *worst-case execution time* (WCET), representing the maximum amount of time for which each job generated by τ_i may need to execute, denoted by C_i. The *relative deadline*, denoted by D_i, refers to the length of the time interval in which the task must complete execution. The *absolute deadline* of a job $\tau_{i,j}$ is denoted $d_{i,j} = a_{i,j} + D_i$, respectively. A job's absolute deadline is the *date* by which the job should complete execution. If $D_i = T_i$ (resp., $D_i < T_i$) holds, then τ_i and its jobs are said to have *implicit deadlines* (resp., *constrained deadlines*). A task system in which $D_i = T_i$ (resp., $D_i \leq T_i$) holds for every task is said to be an implicit-deadline system (resp., constrained deadline), and one in which $D_i > T_i$ holds for one or more tasks is said to be an *arbitrary-deadline* system. The ratio of the WCET to the period of a task is referred to as its *utilization* and corresponds to $u_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$. The *total processor utilization* of the periodic task set is the sum of all the task utilizations, which is $U_p \stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i$.

In this work, a periodic task τ_i will be represented by $\tau_i(\Phi_i, C_i, D_i, T_i)$ with $(D_i = T_i)$.

Having discussed the timing constraints of tasks, we will now examine dependency constraints.

Dependency Constraints

In typical real-time systems, tasks need to cooperate so as to complete their execution. They are said to be *dependent* tasks. To the contrary, tasks that do not require cooperation are referred to as *independent* tasks. Many research studies consider only independent tasks, for simplicity. However, it is essential to consider dependencies in modelization since numerous tasks in mobile robots execute collaboratively and have to synchronize their computations. Consider, for instance, a task responsible for reading sensor values from an obstacle avoidance sensor. The data retrieved by this task is crucial for a control task, which communicates via a shared variable that requires protection. Consequently, the tasks are interdependent - the output of one forms the input for the other. Tasks interact in explicit and/or implicit ways resulting in the so-called *precedence* contraints and *resource* constraints:

- *Precedence Constraints*: These constraints represent the need for computational activities to adhere to a specific order, as established during the design phase, rather than allowing arbitrary execution sequences. Precedence relations are typically illustrated by *directed acyclic graphs* (DAGs), with tasks as nodes and precedence relations as arrows.

- *Resource Constraints*: In the context of process perspectives, resource constraints refer to limitations associated with data structures, sets of variables, main memory areas, or registers of peripheral devices. A resource dedicated to a specific process is classified as *private*, while a *shared resource* is one that can be used by multiple tasks. Resource constraints, specifically those arising from the *mutual exclusion* of access to critical resources, play a significant role in real-time systems. Mutual exclusion is a mechanism that ensures exclusive access to shared resources, preventing simultaneous execution of tasks that require the same resource. In the context of task execution, resources required by a task may not always be available when requested, necessitating proper management and coordination to guarantee that tasks are executed without conflicts or interference.

Mutual Exclusion Problems

The notion of *priority* signifies the relative importance or urgency assigned to tasks concerning their execution sequence. Priority not only plays a crucial role in managing task execution order but also directly influences the resolution of synchronization challenges such as priority inversion. *Priority Inversion* is a widely recognized synchronization

challenge that affects real-time systems [Lampson and Redell (1980)]. This problem occurs when a task with a lower priority unintentionally executes while a task with a higher priority is waiting for execution. To further clarify this concept, τ_l, τ_m, τ_h, respectively, with low, medium, and high priority, as depicted in Figure 2.7.

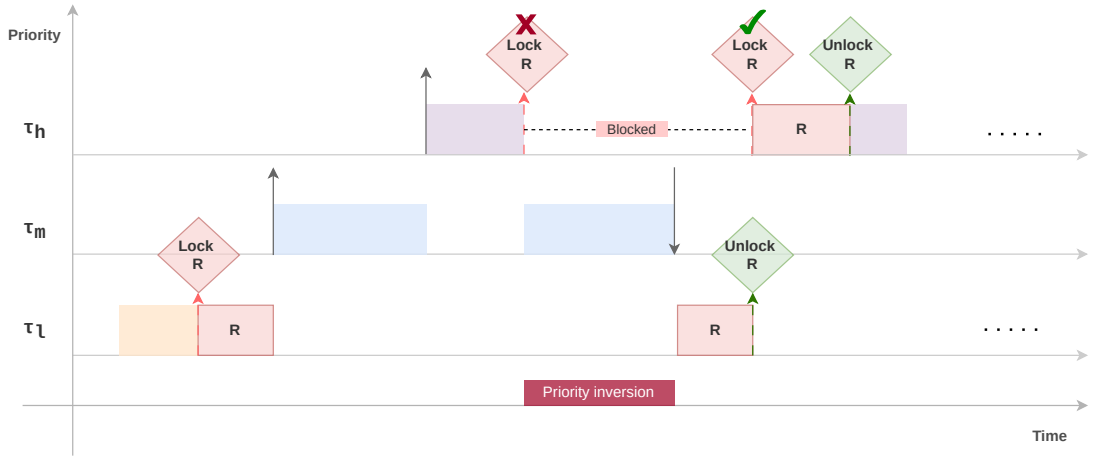


Figure 2.7 Priority inversion scenario. Task τ_l and Task τ_h share same resource R. In this situation medium priority task τ_m blocks the execution of higher priority task τ_h.

Initially, τ_l is running and locks a shared resource R. Subsequently, task τ_m arrives and preempts τ_l execution. When task τ_h becomes ready, it preempts τ_m, and attempts to acquire shared resource R. However, since τ_l has already locked by R, τ_h is unable to execute and is therefore *blocked*. Consequently, task τ_m is resumed and executed. In this scenario, the medium-priority task τ_m indirectly preempts the high-priority task τ_h, by preventing τ_l from executing and releasing the resource R. This leads to priority inversion, which prevents the high-priority task from executing.

Numerous solutions have been proposed to tackle this issue. One prominent approach is known as *priority inheritance* [Sha et al. (1990)]. The fundamental idea of this solution involves temporarily raising the priority of the low-priority tasks (τ_l) that have locked resources needed by higher-priority tasks (such as τ_h in this example). In this special case, priority of lower-priority tasks would be temporarily increased to match the priority of higher-priority task. In this research, we will resolve the priority inversion problem by utilizing *resource access protocols* [Sha et al. (1990)].

2.3.3 Processing Platform

The platform represents the hardware and software architecture of a computing system, and numerous computing systems exist to explore. This study will primarily concentrate

on the computational platforms utilized in mobile robots. As demonstrated in Figure 2.1, architecture reveals that two computational systems are present in autonomous mobile robots, which exchange data to operate the robots locomotion and other activities. High-level processing units, like onboard multi-core processors such as x86 or arm64 devices running *General-Purpose Operating System* (GPOSs), manage middleware and robotic applications. Conversely, low-level processing units, including microcontrollers that operate using *real-time operating systems* (RTOS) like MbedOS[3], FreeRTOS[4], NuttX[5], or automation PCs that run the Windows operating system, handles actuation and sensing systems.

From a deployment perspective, real-time programs can either operate without an operating system (referred to as bare metal) or on an RTOS. The latter is more appropriate when addressing the simultaneous execution of multiple tasks within the system. The system memory can be arranged either as a single shared address space among tasks or as multiple virtual address spaces to ensure task isolation. In the second scenario, applications often need operating system services through *system calls*. This necessitates a transition from *user space* to *kernel space*, which entails an execution mode change that permits software to run privileged instructions and access the entire memory space without constraints. As a result, RTOS-based real-time applications with time-predictable execution must also maintain temporal determinism at the kernel-space level. In general, the objectives of RTOSs and GPOSs diverge, with the former focusing on setting a maximum limit for execution time and the latter aiming to enhance average performance.

This research will emphasize the high-level onboard processing units in mobile robotic systems utilizing GPOS Linux[6]. Linux has become an attractive option for the embedded world due to its numerous advantages. However, the kernel's real-time performance doesn't fully meet the requirements for all real-time classes, as it was primarily developed for general-purpose systems. In response, the Linux Foundation initiated the *Real-Time Linux collaborative project*[7] in 2016, focusing on coordinating kernel development specifically for real-time environments, with an emphasis on the PREEMPT_RT patch. Developed in 2005, this patch sought to improve determinism and minimize latencies in the Linux kernel. Additionally, the *Open-Source Automation Development Lab* (OSADL)[8] is a significant contributor to real-time Linux, serving as

[3]https://os.mbed.com/mbed-os/
[4]https://www.freertos.org/
[5]https://nuttx.apache.org/
[6]https://www.kernel.org/
[7]https://wiki.linuxfoundation.org/realtime/start
[8]https://www.osadl.org/

an organization committed to advocating for and facilitating the use of open-source software in embedded environments.

In Linux environments, it's important to note that a *process* consists of one or more *threads* sharing the same address space, and a *thread* is a sequence of instructions. A *job* refers to a single unit of work performed by a single thread, and a *task* is synonymous with a *thread*.

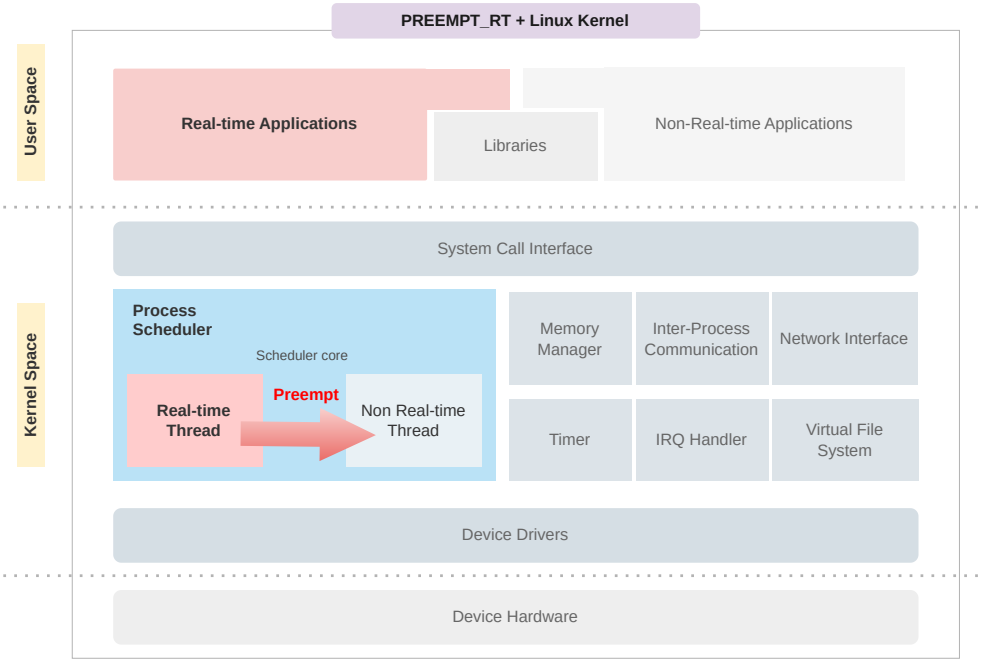
Co-kernel or *dual kernel-based* approaches, such as *Xenomai*[9], provide an alternative to the PREEMPT_RT patch by utilizing an additional OS for managing real-time threads. Xenomai's latest version (version 4) with EVL core[10] improves Linux's real-time capabilities by embedding a companion core, making it SMP-scalable, easier to integrate and maintain. Linux-based dual kernel systems typically require an interface layer like the *I-pipe*[11] to couple the secondary real-time core with the kernel's logic. However, maintaining the I-pipe has been proven challenging due to conflicts with mainline kernel changes. *Dovetail*[12], I-pipe's successor, aims to introduce a high-priority execution stage for time-critical operations, provide a straightforward interface for autonomous cores, enables a common Linux programming model for ultra-low latency applications, and facilitates Dovetail and EVL core maintenance with common kernel development knowledge, all while integrating the interrupt pipeline logic directly into the generic *interrupt handling* (IRQ) layer. Figure 2.8 illustrates the architecture of both PREEMPT_RT Linux Kernel Figure 2.8a and Xenomai-4 patched Linux kernel Figure 2.8b.

Selecting the appropriate platform model and real-time operating system is crucial for the successful implementation of real-time systems in robotic applications. Understanding the interplay between platform models, operating systems, and real-time extensions is essential for developing efficient and reliable real-time systems. This understanding is highly required to satisfy the non-functional properties in correlation with the functional properties of robotic systems.

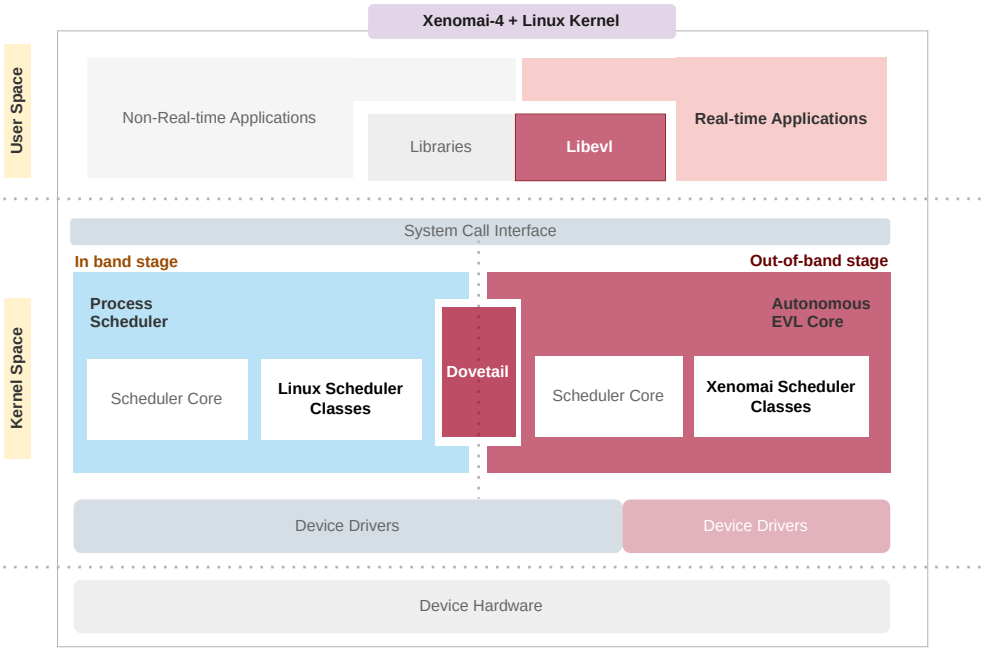
2.3.4 Scheduling Algorithm

Tasks generally execute concurrently on the same processing unit such as a microcontroller. Consequently, the operating system should be equiped with a specific software in charge of deciding which task to execute at every time instant. In other terms,

[9] https://source.denx.de/Xenomai
[10] https://evlproject.org/overview/
[11] https://lwn.net/Articles/140374/
[12] https://evlproject.org/dovetail/



(a) PREEMPT_RT enabled Linux Kernel



(b) Xenomai patched Linux Kernel

Figure 2.8 Architecture of PREEMPT_RT Linux Kernel and Xenomai-4 (EVL core) patched Linux Kernel.

the question is to decide how to assign priorities to the task so that their timing requirements be satisfied, whenever possible. The *scheduler* serves this purpose by

implementing a *scheduling algorithm*. It determines the sequence of task execution and allocates the necessary resources. Real-time scheduling has been an active topic of research from the beginning of the 70's, leading to important results that will be described in the following.

Real-Time Task State Transitions

Most of the time, the number of tasks to be executed is lower than the number of processing units available for embedded applications due to various reasons such as cost, weight, etc. Consequently, the processor becomes the most critical resource that needs to be managed. In any multi-tasking system, a task can be, either not executing since it is waiting for the event that will release it, waiting for the processor which is occupied by another task, or actively executing on the processor. This leads us to define the states of a real-time task as follows:

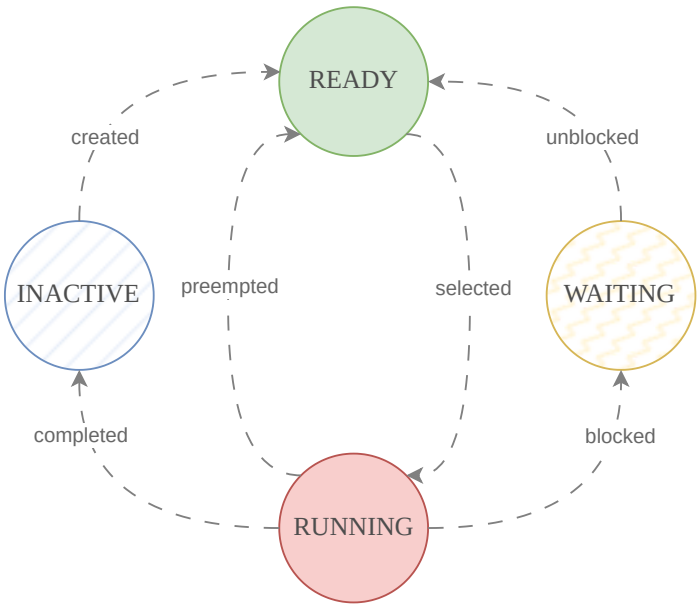


Figure 2.9 Real-time Tasks life cycle.

- *Inactive*: In this state, the task has not yet been created or initialized. It is not eligible for execution and is not running.
- *Ready*: When a task is created and initialized, it enters the ready state. In this state, the task is eligible for execution but is waiting for the scheduler to allocate processor time to it.

- *Running*: When the scheduler selects a task for execution, it enters the running state. In this state, the task is actively executing its code and performing the required operations.
- *Waiting*: If a task is interrupted or blocked during its execution, it enters the waiting state. In this state, the task is not running, but is waiting for some event, such as the release of a shared resource or the completion of an I/O operation, to start/resume execution.

The life cycle of a real-time task is described using a state diagram in Figure 2.9, which shows the transitions between these different states. These states can be used to describe the current state of the task, and the scheduler can use this information to decide which task to execute next. Each state of the task is important for the scheduler to understand the task's status and for the developer to understand how the system behaves. The scheduler uses this information to determine the order in which tasks are executed and to allocate the necessary resources to the tasks.

Classification of Scheduling Algorithms

A scheduling algorithm is responsible for assigning processor resources to tasks, meaning it establishes the execution time intervals and processors for each job while considering any limitations, such as concurrency restrictions. In real-time systems, the primary goal of processor allocation methods is to satisfy timing constraints. Numerous scheduling algorithms have been proposed for managing real-time tasks. For this research, we will focus on several key classes [Buttazzo (2011)]:

- **Preemptive vs. Non-preemptive**
  - *Preemptive* scheduling algorithms allow task interruptions to allocate the processor to a different active task based on a predefined scheduling policy. This approach enables the system to quickly respond to environmental changes and meet timing constraints.
  - *Non-preemptive* scheduling algorithms require a task to execute without interruption until completion. Although this approach may simplify implementation with lower overhead (due to the absence of context switching caused by preemptions) in comparison to preemptive scheduling, it results in lower-priority tasks obstructing higher-priority ones (priority inversion).

- **Static vs. Dynamic**
  - *Static* scheduling algorithms utilize fixed parameters assigned to tasks before activation for scheduling decisions.
  - *Dynamic* scheduling algorithms are those in which scheduling decisions are based on dynamic parameters that may change during system runtime.
- **Off-line vs. Online.**
  - *Off-line* scheduling algorithms generate schedules for the entire task set before the system initialization. These schedules are stored in a table and later executed by a dispatcher, making them suitable for systems with fixed workloads or predictable behavior.
  - *Online* scheduling algorithms make decisions at runtime while the system is in operation. These algorithms adapt quickly to environmental changes and are well-suited for systems with fluctuating workloads or unpredictable behavior.
- **Idling vs. Non-idling**
  - An *idling* scheduling algorithm allows for the possibility of inserting idle times in the processor's operation. In such cases, a task may be selected for execution or be delayed for a specified duration, even if the processor is idle and available for use.
  - A *non-idling* or work-conservative algorithm ensures that a processor executes the highest-priority task as soon as it is ready and cannot delay it if there is no other work, i.e., it operates without idle times.
- **Monoprocessor vs. Multiprocessor**
  - A *monoprocessor* scheduling algorithm is designed to manage tasks running on a single processor system.
  - A *multiprocessor* scheduling algorithm, on the other hand, allows tasks to be executed across multiple processors in a system.
- **Clairvoyant vs. Non Clairvoyant**
  - A *clairvoyant* scheduling algorithm has a complete knowledge of the future to take decisions, i.e., it is aware of all the characteristics of all tasks in advance.

– A *non-clairvoyant* scheduling algorithm does not need the knowledge of future to take decisions.

Real-Time Scheduling Analysis

Prior to delving into various scheduling approaches, it is essential to define terms frequently employed in characterizing the properties of real-time scheduling algorithms and comparing them to one another.

Schedulability testing means verifying off-line whether a given application represented by its periodic task set may be scheduled feasibly, i.e., respecting all the deadlines on a given processor.

Definition 1. *Valid: A valid schedule produced by a scheduling algorithm χ for a given task set τ is a schedule in which the constraints of each of the tasks are met.*

Definition 2. *Feasibility: Scheduling is feasible for a given task set if there exists at least one scheduler capable of producing a valid schedule.*

Definition 3. *Schedulability: A task set is schedulable if there exists an algorithm for which it is guaranteed schedulable.*

Definition 4. *Optimality: A scheduling algorithm is said to be optimal for a class of systems and a set of scheduling policies given certain assumptions if and only if any system schedulable by some policy in this set is guaranteed schedulable by this algorithm.*

In systems that require guaranteed performance, it is crucial to apply a schedulability test tailored to the scheduling policy of the execution platform. The schedulability test can be characterized in the following ways:

- *Sufficient*: if this property is present, then the task set is schedulable. If absent, the test cannot be used to conclude that the set of tasks is effectively schedulable;
- *Necessary*: if this property is not present, then the corresponding set of tasks is not schedulable. If not, it is nevertheless not possible to conclude that the set of tasks can be scheduled so that all time constraints are met;
- *Exact*: the test is both *necessary* and *sufficient*. The test can be used to determine prior to execution whether or not a task set is schedulable.

Exact schedulability tests are the most desirable for time-validating real-time systems. However, this kind of test does not always exist, can be too costly to evaluate

or can impose restrictive assumptions on the tasks. Real-time software designers therefore often choose to use tests that are sufficient but not necessary.

From an engineering standpoint, a schedulability test that is both tractable and demonstrates low pessimism is ideal. A tractable test refers to one that can be executed within a reasonable time frame and with manageable computational resources. Low pessimism relates to a test's propensity to yield false negatives, suggesting that the test might fail even if the system is capable of meeting its deadlines.

By conducting the schedulability test *offline* before run-time, system designers and engineers can confirm that the selected scheduling algorithm and system configuration deliver the required temporal accuracy. This procedure aids in preventing deadline misses and guarantees reliable system operation. Typically, a scheduling algorithm possesses a unique form of schedulability analysis. Some prevalent forms of schedulability analysis include *processor utilization-based* analysis, *response time analysis* (RTA), and *processor demand* (DBF). These analyses are specific to the scheduling algorithm, which will be discussed in the subsequent sections.

In this dissertation, we will employ several key terminologies which are defined as follows:

- **Processor Utilization** (U_p): As mentioned in the periodic task model, the utilization of a task τ_i, denoted by u_i, is the ratio of its Worst-Case Execution Time (WCET) to its period, $u_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$. The total processor utilization, U_p, is the sum of the utilizations of all tasks in the task set τ, represented as [Liu and Layland (1973)]:

$$U_p \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} u_i \tag{2.1}$$

- **Demand bound function** (W): The demand bound function for a recurring task τ_i, represented as $W_i(t)$, establishes a maximum constraint on the cumulative execution time demanded by jobs of τ_i, which not only arrive but also have deadlines falling inside any time interval of length L. This can be expressed as [Baruah et al. (1990)]:

$$W_i(L) = \max\left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1, 0\right) \times C_i \tag{2.2}$$

- **Worst-Case Response Times** (RT): In a system, task τ_i experiences interference from tasks with higher priority, which impacts the response time of the task.

We can represent the worst-case response time as $RT_i = C_i + I_i$, where I_i where signifies the largest delay in execution caused by higher-priority tasks during the interval $[t, t + RT_i)$. Task τ_i is interfered with by each higher-priority task, and thus I_i is calculated as $I_i = \sum_{j \in hp(i)} \left\lceil \frac{RT_i}{T_j} \right\rceil C_j$. Here, $hp(i)$ represents the set of tasks with a higher priority relative to τ_i. Consequently, the final RT_i can be determined as [Joseph et al. (1986)]:

$$RT_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{RT_i}{T_j} \right\rceil C_j \tag{2.3}$$

Nonetheless, the indeterminate RT_i term is present on both sides of the equation, which calls for an iterative method to determine its accurate value. Let RT_i^n represent the n^{th} approximation of the actual RT_i value. We can compute these approximations using the subsequent equation:

$$RT_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{RT_i^n}{T_j} \right\rceil C_j \tag{2.4}$$

The iteration begins with $RT_i^0 = 0$ and terminates once $RT_i^{n+1} = RT_i^n$.

- **Hyperperiod** (H): The hyperperiod of a task set represents the smallest time interval after which the global periodic pattern for all tasks repeats. It is typically calculated as the Least Common Multiple (LCM) of the periods of all tasks in the system.

Scheduling Problem

In real-time systems, optimal algorithms are those that can find the best possible solution to a scheduling problem, ensuring that all tasks meet their deadlines, if such a solution exists. However, designing and implementing optimal algorithms for more complex problems can be computationally expensive and may not be feasible within the time constraints of real-time systems. Many real-time scheduling problems are classified as NP-hard, which means that no known algorithm can find an optimal solution in polynomial time (i.e., time complexity of $O(n^k)$, where n is the input size and k is a constant). In such cases, the time complexity of finding an optimal solution can grow rapidly with the increase in the number of tasks, making it impractical to solve for large-scale real-time systems or systems with stringent timing requirements. Examples of such problems include scheduling tasks with shared resources, and multiprocessor

scheduling. For real-time scheduling problems classified as NP-hard, heuristic or approximation algorithms are often employed to find near-optimal solutions more efficiently. While these algorithms may not guarantee optimality, they can provide practical and effective solutions for real-time systems with strict timing constraints and limited computational resources.

In summary, optimal algorithms guarantee the best possible solution to a given problem, but they can be computationally expensive, especially for NP-hard problems. Non-optimal (Heuristic algorithms) offer an alternative approach, trading off some degree of optimality for faster, more efficient solutions that are more suitable for real-time systems with strict timing constraints.

Priority Driven Scheduling

Considering the vast body of literature available on scheduling, it is impractical for any survey paper to provide an exhaustive overview. Ramamritham and Stankovic (1994) has categorized different real-time scheduling algorithms into a set of distinct paradigms. In this dissertation, we particularly emphasize on *priority-based scheduling* approaches for *uniprocessor* systems.

Priority-based Scheduling Algorithm

It is a widely used approach to manage the execution of real-time tasks by assigning a priority to each task and then ordering them according to its priorities. In this method, the higher-priority tasks are executed before the lower-priority tasks. Assignment can be based on timing parameters such as relative deadline, period, laxity or on other user-defined metrics. The priority-driven scheduling paradigm can be classified as *Fixed priority* scheduling and *Dynamic priority* scheduling.

In **Fixed-Priority Scheduling**, prior to execution, each task is assigned a *static* or *fixed* priority, which remains constant during execution. Each job spawned by the task inherits the same priority value. Examples of fixed priority assignment algorithms are *Rate Monotonic* scheduling (RM) [Liu and Layland (1973)] for periodic tasks, where task priorities are assigned inversely proportional to their periods (i.e., shorter periods have higher priorities) and *Deadline Monotonic Scheduling* (DMS) [Leung and Whitehead (1982)] for periodic tasks, where task priorities are assigned inversely proportional to their relative deadlines (i.e., shorter deadlines have higher priorities). RM is certainly the most frequently implemented scheduler firstly because its implementation is simple

and because it was proved as the best fixed priority scheduler when applied to periodic tasks with implicit deadlines.

In **Dynamic-Priority Scheduling**, task priorities are updated during run-time based on changing system conditions or task attributes. Examples of dynamic priority assignment algorithms are *Earliest Deadline First* (EDF) [Liu and Layland (1973)], where task priorities are assigned based on their absolute deadlines, with tasks having the earliest deadlines assigned the highest priorities and *Least Laxity First* (LLF) [Mok (1978)] where task priorities are assigned based on their laxity. Tasks with the least laxity have the highest priorities.

In this dissertation, we concentrate on the Rate Monotonic fixed-priority scheduling algorithm and on the Earliest Deadline First dynamic-priority scheduling algorithm. Subsequently, we provide a detailed schedulability analysis for these two scheduling algorithms, along with illustrative schedules for a given task set.

Scheduling Algorithm	Schedulability Test	Inequality	Condition	Considering
Earliest Deadline First Scheduler	Processor Utilization Based Liu and Layland (1973)	$U_p \leq 1$	Exact	Independent tasks Implicit deadlines Synchronous tasks
	Processor Demand (DBF) Baruah et al. (1990)	$\forall L : L \geq 0 : [(\sum_{i=1}^{n} W_i(L) \leq t)]$	Exact	Independent tasks Constrained deadlines Synchronous tasks
			Sufficient	Independent tasks Constrained deadlines Asynchronous tasks
Rate Monotonic Scheduler	Processor Utilization Based Liu and Layland (1973)	$U_p \leq n\left(2^{1/n} - 1\right)$	Sufficient	Independent tasks Implicit deadlines Synchronous tasks
	Response Time Analysis Joseph et al. (1986)	$RT_i^{n+1} \leq D_i$	Exact	Independent tasks Constrained deadlines Synchronous tasks
			Sufficient	Independent tasks Implicit deadlines Asynchronous tasks

Table 2.2 Schedulability tests for the EDF and RM scheduling algorithms.

Schedulability Test

The schedulability tests for these algorithms are summarized in Table 2.2. The schedulability bound is proportional to the number of tasks and decreases as n, the number of tasks, increases. Considering that $\lim_{n\to\infty} n\left(2^{1/n} - 1\right) = \ln 2 \simeq 0.69$, RM can schedule any task set if $U_p \leq 0.69$. Baruah and Burns (2006) demonstrated the sustainability property of these schedulability tests meaning that, if a task set is juged schedulable it will remain schedulable with more favourable parameters i.e., a shorter computation time, a longer period or a longer deadline.

EDF Illustrative Schedule

Let us consider a mobile robotic application. The task parameters are presented as follows:
`task_name (offset, wcet, relative deadline, period)`:
`Motor Control (0,2,5,5)`, `Obstacle Sensor (0,1,10,10)`, `Teleop (0,3,15,15)`, `Battery (0,1,30,30)`.
In real-time systems, the scheduling algorithm's schedule is theoretically displayed using Gantt charts. We present the Gantt schedule of the aforementioned task set using the Earliest Deadline First (EDF) scheduling algorithm. The processor utilization-based schedulability test is satisfied since $U_p \leq 1$. The same sequence of the schedule will be repeated in the next hyperperiods. Figure 2.10, illustrates the Gantt chart up to the end of the hyperperiod equal to 30. The schedule demonstrates that the `Teleop task` is preempted by the higher-priority `Motor Control` task at time instant 5. Additionally, the schedule highlights the intervals during which the processor is idle i.e., has nothing to execute. It is evident that there are no missed deadlines in the schedule.

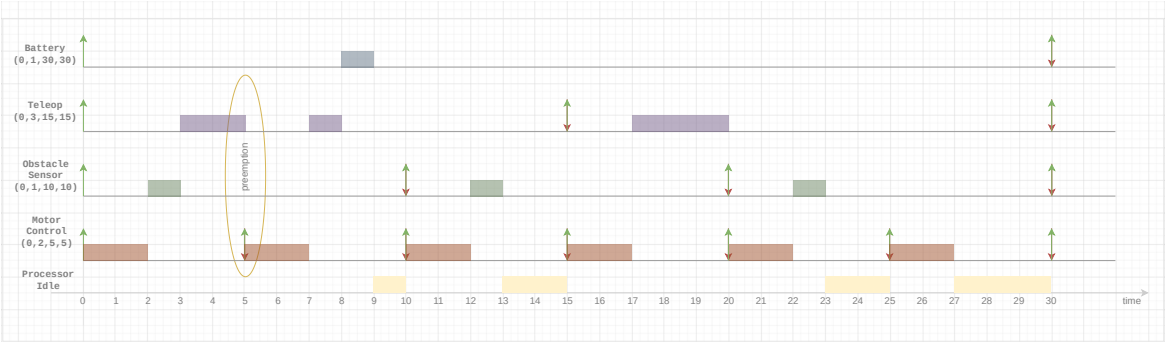


Figure 2.10 The Gantt chart represents the schedule of typical mobile robotics functional tasks using the Earliest Deadline First (EDF) scheduling algorithm.

In this example, for sake of simplicity, we have assumed the tasks to be independent. However, in reality, tasks participate to a common objective and depend on each other, exchanging data for example or simply synchronizing their execution. Our research places emphasis on characterizing and addressing these dependencies to ensure efficient and reliable scheduling in real-time systems.

2.4 Conclusion

This chapter has provided an introduction to the mobile robotics systems explored in this research. We have presented a generic architecture that displays all the

subsystems within a mobile robot, as well as the abstraction levels illustrating the hardware and software abstractions of mobile robots. We have defined the most crucial requirements, namely the functional and non-functional requirements of mobile robot software architecture. It can be observed that the functional requirements of the system are satisfied through robotic middleware frameworks, such as ROS. However, ROS is not a traditional operating system, and middleware alone cannot satisfy the *non-functional requirements* of the mobile robotic system.

Therefore, we have introduced the fundamental concepts of real-time systems, which are essential for achieving *deterministic* execution and satisfying the non-functional requirements of the system. We have discussed three main aspects: the *real-time workload*, which provides the periodic task model; the *processing platform*, which is responsible for achieving non-functional requirements in correlation with the functional requirements that we will focus on in this research; and the *scheduling algorithm*, which encompasses the properties and two classical priority-based scheduling policy (RM and EDF). One aspect we have not discussed in this chapter is the *energy* constraints, which is a primary concern of this research. This topic will be reviewed in the following chapter.

Chapter 3

Energy Management Strategies: State-of-the-Art

In this chapter, we explore the progression of energy management, from mobile robotics to real-time energy harvesting computing systems. We begin by outlining the perspectives on energy management in mobile robotic systems and real-time systems. In Section 3.1, we review the high-level energy-efficient approaches practiced in mobile robotics systems. Subsequently, we discuss the emergence of energy harvesting mobile robots and their applications in Section 3.2. Section 3.3 introduces the energy management techniques for Real-Time Energy Harvesting Systems (RTEHS) and includes definition of energy-neutrality and energy-aware scheduling in RTEHS. In Section 3.4, we describe in detail an optimal scheduling algorithm dedicated to RTEHS, namely Earliest-Deadline-Harvest (ED-H). Finally, Section 3.5 discusses the uncertainties of energy harvesting systems in real-world implementations. This includes solar energy harvesting techniques and solar energy prediction, which are discussed in Section 3.6. In Section 3.7, we address the uncertainty of energy storages that power mobile robotic systems.

3.1 Energy Management - An Overview

In this section, we provide an in-depth analysis of energy management strategies for *real-time computing systems* and *mobile robots*. We discuss the importance of energy efficiency, various techniques to minimize energy usage, challenges, and future trends. This review offers insights into enhancing energy efficiency in mobile robots, ultimately increasing *functional requirements* and contributing to *non-functional requirements* and environmental impact.

Energy efficiency is a crucial consideration in the design and operation of mobile robots. Achieving a high level of energy efficiency enables robots to perform tasks more effectively and efficiently, extending their *operational lifespan*. Several strategies can be employed to attain energy efficiency, such as using energy-efficient components, designing lightweight robots, employing high-efficiency batteries, implementing adequate power management techniques, and optimizing navigation and task performance. By

reducing power consumption, mobile robots can operate for extended periods, making them more effective and versatile in a wide range of applications.

Energy management is increasingly becoming a critical and central issue in embedded devices and mobile robotic systems. This problem can be examined at two levels: the *component level* and the *system level*. Traditionally, many components integrated into mobile robots have been designed with low-power consumption in mind. However, it is essential to recognize the critical distinction between *power/energy-aware* systems and *low-power* systems [Liu et al. (2001a)].

Power/energy-aware systems must optimize the utilization of their available power, encompassing low-power design as a special case. By adopting an energy-aware design approach, the overall utility and performance of mobile robots can be significantly enhanced. This strategic focus on energy management ensures that the robots not only consume minimal power but also make the most effective use of their energy source, enabling them to perform tasks more efficiently and autonomously [Liu et al. (2001b)].

It is essential to differentiate between the two terms, *power-aware* and *energy-aware*, which are often used interchangeably in research. Power-aware and energy-aware are related but distinct concepts in the context of designing systems powered by limited energy sources.

- *Power-aware*: refers to the design of systems and algorithms that optimize power consumption by making the best use of available power source. Power is the rate at which energy is consumed or produced, usually measured in *watts* (W). Power-aware systems concentrate on managing power consumption, considering various power sources and adapting the system's operation to optimize power usage. Examples include adjusting the *voltage and frequency of processors*, managing *power states of components*, and using *power-efficient scheduling techniques*.

- *Energy-aware*: involves creating systems and algorithms that optimize total energy consumption over a given period. Energy is the total amount of power consumed over time, typically measured in *joules* (J) or *watt-hours* (Wh). Energy-aware systems aim to reduce overall energy consumption while meeting specific performance, reliability, or functionality requirements. Examples include *energy harvesting techniques*, *energy-efficient task scheduling*, and optimizing the use of batteries and other energy storage devices [Mei et al. (2005), Farooq et al. (2023)].

To sum up, power-aware systems focus on optimizing the rate of energy consumption, while energy-aware systems concentrate on optimizing the total energy consumed over time.

3.1.1 Energy-efficient Real-time Computing System

In the early days of computing, processors were primarily designed to maximize performance without much consideration for energy efficiency. As technology advanced and computing systems became more pervasive, concerns about energy consumption, heat generation, and environmental impact came to the forefront. This led to the development of energy-efficient techniques such as *Dynamic Power Management (DPM)* [Benini et al. (2000)], and *Dynamic Voltage and Frequency Scaling (DVFS)* [Burd et al. (2000), Pillai and Shin (2001), Schmitz and Al-Hashimi (2013)]. These techniques have become crucial for modern computing systems with CPUs and GPUs, especially in real-time embedded applications with autonomy requirements.

DPM dynamically adjusts its *power states* based on the system's workload and performance requirements. In a processing system, DPM [Benini et al. (2000)] may involve turning off or placing unused components into low-power states when they are not needed, ensuring that only the necessary resources are active. For example, this may include turning off some CPU or GPU cores, network interfaces, or bluetooth connections.

DVFS adjusts processor's supply voltage based on computational demands. Lowering the voltage and in consequence frequency too, reduces power consumption. But at the same time, it impacts negatively the performance in terms of response time for the tasks, necessitating a balance between energy efficiency and real-time computing system requirements [Burd et al. (2000)].

The work proposed by Amarnath et al. (2022) focuses on HetSched, a Quality-of-Mission aware scheduler for autonomous vehicles on heterogeneous domain-specific systems-on-chips (DSSoCs). They have integrated the DVFS technique within the scheduler to optimize energy usage while considering task deadlines and a fraction of available slack. By incorporating DVFS, HetSched can further improve energy efficiency and maintain real-time computing system performance in autonomous vehicle applications.

3.1.2 Energy-efficient Mobile Robots

Several high-level techniques have emerged to optimize energy usage while maintaining operational goals. This concerns *motion planning*, *path planning*, *joint speed control and power scheduling (JSP)*.

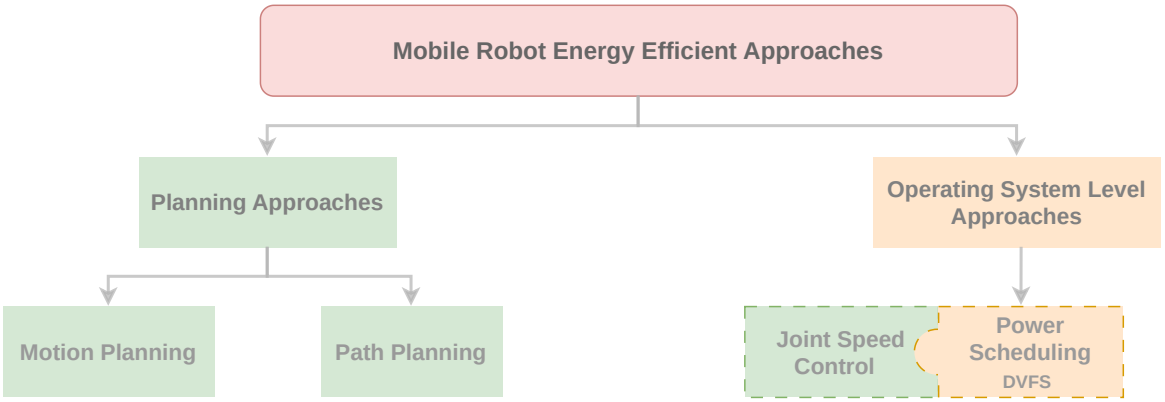


Figure 3.1 Energy-efficient approaches practiced for mobile robots.

- *Energy-efficient motion planning (EMP)*: algorithms aim to find the *optimal trajectory* that minimizes energy consumption through optimized velocity profiles. They use a calibration procedure to identify the *best velocity profile* to reduce energy consumption. For example, an energy-efficient motion planner may consider factors such as the robot's velocity, acceleration, or terrain properties to generate a trajectory that consumes the least amount of energy. [Mei et al. (2004), Tokekar et al. (2011), Xie et al. (2018), Jaramillo-Morales et al. (2020)]

- *Energy-efficient path planning (EPP)*: techniques determine the most energy-saving route by considering factors such as distance, terrain, and robot dynamics. For instance, a path planning algorithm could generate a route that avoids steep inclines or rough terrain, which would require more energy for the robot to traverse. Another example is the use of an A* algorithm with a cost function that incorporates energy consumption, allowing the robot to find the most energy-efficient path. [Mei et al. (2006), Yang et al. (2010), Liu and Sun (2011), Datouo et al. (2018), Maidana et al. (2020), Kyaw et al. (2022)]

- *Joint Speed Control and Power Scheduling (JSP)*: problem aims to optimize both the joint speed profiles of a mobile robot and the power allocation across various components and subsystems to minimize energy consumption. The JSP

problem seeks a balance between optimal joint speed control, which determines the velocity profiles for the robot's joints, and power scheduling, which adjusts the processor frequency to meet the computational demands of the tasks and optimize energy usage. By effectively managing the processor frequency through DVFS and optimizing the joint speed control, the JSP problem aims to minimize the overall energy consumption of the mobile robot. [Brateman et al. (2006), Zhang and Hu (2007), Zhang et al. (2009), Mohamed et al. (2021)]

Each of the energy-efficient techniques mentioned here has its own drawbacks and limitations. Motion and path planning techniques face challenges such as high computational complexity, sensitivity to model inaccuracies, and scalability issues. Meanwhile, the JSP approach has drawbacks such as overhead and latency, potentially impacting the real-time performance of the system. Additionally, it is highly complex to precisely tune the processor frequency and achieve optimal speed.

In Table 3.1, we provide a concise review of research works that focus on energy-efficient approaches for mobile robots. These studies primarily concentrate on two types of robots: *differential drive* and *omnidirectional drive* robots, which are widely developed for research purposes. It can be observed that most of the proposed works are based on *simulation* experiments, with very few ones that provide valid results through real-world experiments. The main objective of this review is to identify the scope and nature of these approaches. Recall that the primary goal of energy-efficient strategies for mobile robots is to prolong their operational time.

From our analysis, we found that all previous researches on energy efficiency in mobile robots address this issue through only one criterion: *energy saving*. Energy saving refers to the reduction of energy consumption by using energy-efficient approaches without altering the system's overall functionality, thereby extending the operation time. However, this approach still *relies on conventional energy sources, which may lead to instances where the energy in the storage unit is completely drained, then requires replenishment of the storage unit.*

Our dissertation focuses on an emerging research trend in mobile robotics that seeks to harness renewable energy sources. This approach employs *energy-neutral* strategies, utilizing energy-aware scheduling to meet both the timing and energy constraints of the processing system in mobile robots. These strategies often involve adopting intelligent, conservative modes that balance the demand for energy from traditional sources with harvested energy. As a result, this approach prevents the system from depleting its energy reserve, ensuring that both functional and non-functional requirements of mobile robotic systems are met.

Research Works	Energy-Efficient Approaches	Robot Type	Experiments	Energy Saving	Energy Neutral
Mei et al. (2004)	EMP	OD	Simulation	✓	✗
Mei et al. (2006)	EPP	DD	Simulation	✓	✗
Brateman et al. (2006)	JSP	DD	Simulation	✓	✗
Zhang and Hu (2007)	JSP	-	Simulation	✓	✗
Wang et al. (2008)	DVFS	-	Simulation	✓	✗
Zhang et al. (2009)	JSP	-	Simulation	✓	✗
Yang et al. (2010)	EPP	DD(4W)	Simulation	✓	✗
Liu and Sun (2011) Liu and Sun (2012) Liu and Sun (2014)	EPP	DD	Real-World	✓	✗
Tokekar et al. (2011)	EMP	DD(4W)	Real-World	✓	✗
Henkel et al. (2016)	EPP	OD	Real-World	✓	✗
Yacoub et al. (2016)	EMP	DD	Simulation Real-World	✓	✗
Xie et al. (2018)	EMP,EPP	OD	Real-World	✓	✗
Datouo et al. (2018)	EPP	OD	Simulation	✓	✗
Valero et al. (2019)	EPP	DD(4W)	Real-World	✓	✗
Ramos (2019)	EMP,EPP	DD	Simulation	✓	✗
Jaramillo-Morales et al. (2020)	EMP	DD	Real-World	✓	✗
Maidana et al. (2020)	EPP	DD	Simulation	✓	✗
Mohamed et al. (2021)	JSP	DD(4W)	Real-World	✓	✗
Kyaw et al. (2022)	EPP	DD	Real-World	✓	✗

Table 3.1 **Research work summary of energy-efficient mobile robotic approaches**. Abbreviations: EMP - Energy-efficient Motion Planning, EPP - Energy-efficient Path Planning, JSP - Joint Speed and Power Scheduling, DVFS - Dynamic Voltage and Frequency Scaling, OD - Omnidirectional Drive, DD - Differential Drive, DD(4W)- Differential Drive (2 actuated wheel, 2 steering wheel).

To the best of our knowledge, none of the examined works have tackled the primary requirements of industrial standardization for functional requirements (FR) and non-functional requirements (NFR) of system or software architecture in mobile robotic systems. As a result, our dissertation concentrates on the integration of energy-neutral techniques in mobile robots, aiming to investigate this area more thoroughly and contribute to a clearer understanding of energy efficiency in such systems. By emphasizing the use of renewable energy sources and encouraging more sustainable operation, our research aspires to improve the performance of mobile robots to meet both functional and non-functional requirements.

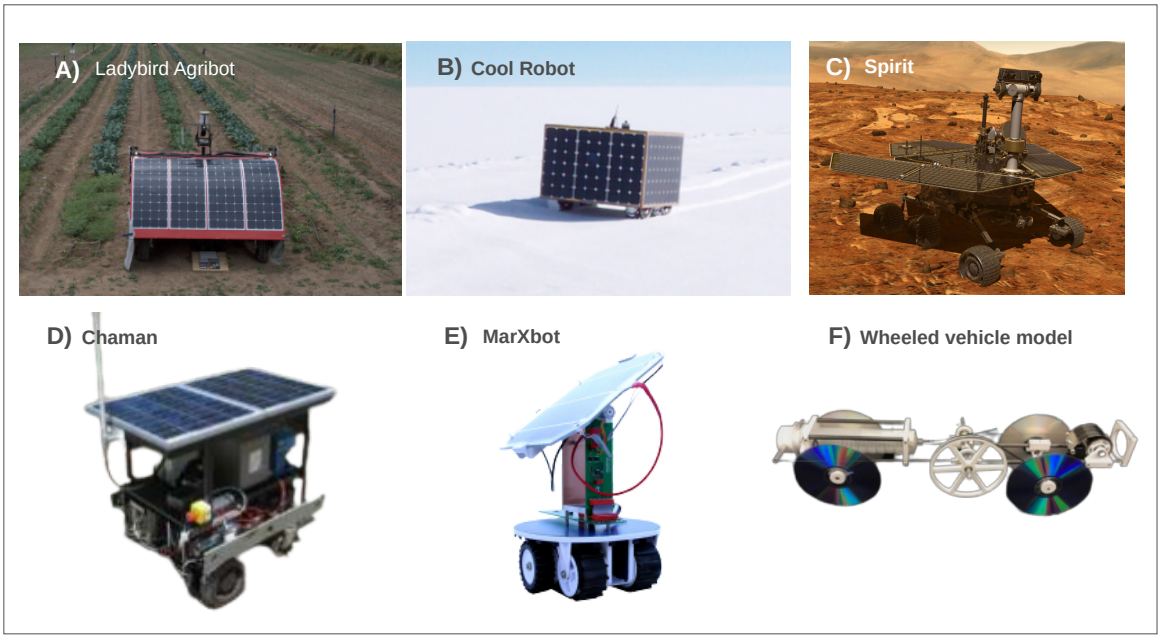


Figure 3.2 Energy-Harvesting Mobile Robotic Systems. A) Ladybird Agribot [Bender et al. (2020)], B) Cool Robot Antarctic Rover [Lever et al. (2006)], C) Spirit Mars Rover [1], D) Chaman Monitoring Robot [Guerrero-González et al. (2010)], E) MarXbot Indoor exploration [Vaussard et al. (2013)], F) Wheeled vehicle powered by temporal temperature gradients harvested using butane and iso-butane [Xiao et al. (2019)].

3.2 Emergence of Energy-Harvesting Mobile Robots

In recent years, researchers have developed mobile robots that utilize renewable energy sources along with the primary power supply. Figure 3.2 shows some of these robots, highlighting their distinct features and applications. For instance, Ladybird [Bender et al. (2020)] is a robot designed for proximal crop sensing in agriculture. It harvests solar energy and recharges its battery source, though the evaluation of the solar power system and energy management of the robot remains unaddressed. Similarly, Lever et al. (2006) proposed a robotic system primarily developed for exploration in the Antarctic region, where traditional power sources are not feasible. However, these robots were limited to summer operations due to the availability of sunlight. Guerrero-González et al. (2010) designed a wheeled robot named "Chaman" for surveillance and monitoring, which also utilizes solar power harvested via photovoltaic cells. Vaussard et al. (2013) introduced MarXbot, one of the first mobile robots that consider indoor photovoltaic harvesting. Furthermore, Xiao et al. (2019) proposed a novel idea for car-like wheeled

[1] https://solarsystem.nasa.gov/missions/spirit/in-depth/

robots, harvesting energy through an actuation system with temperature gradients using butane and iso-butane gases.

Existing studies do not address how to effectively utilize the harvested energy or manage the energy within the system to meet critical constraints. Notably, many space rovers[2] make use of solar harvesting technologies as their primary energy source, and these systems are designed to make mission-critical decisions based on energy availability. However, the approaches used in these systems are not universally applicable, and the unavailability of relevant documents prevents their adaptation to other systems.

Consequently, the pursuit of energy neutrality for mobile robotic systems remains an open research area. In our dissertation, we aim to rigorously examine this topic, emphasizing the integration of energy management strategies. This approach will enable mobile robots to efficiently utilize harvested energy and achieve energy neutrality, ultimately contributing to a more sustainable and effective operation.

3.3 Real-Time Energy Harvesting System

Energy management in systems with finite energy source can be significantly enhanced through energy neutral techniques. These techniques have been extensively explored and addressed in the context of *real-time computing* systems mainly wireless sensor networks that power connected Internet of Things (IoT) devices. These system often utilize energy harvested from renewable sources through energy harvesting units, forming *Real-Time Energy Harvesting Systems* (RTEHS).

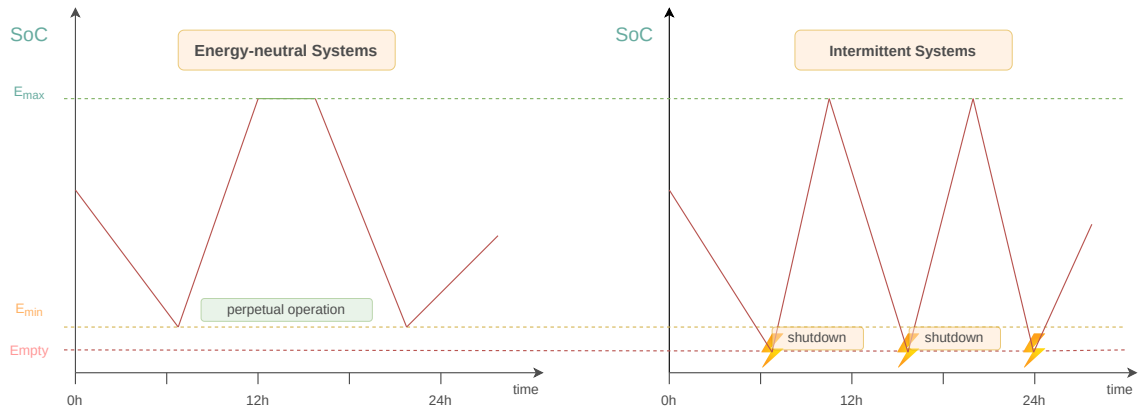


Figure 3.3 Energy Management Techniques for RTEHS adapted from Hanschke (2022).

Figure 3.3 illustrates two energy management techniques for RTEHS:

[2]https://www.jpl.nasa.gov/missions/mars-pathfinder-sojourner-rover

1. *Energy Neutral Operation* (ENO) systems are designed to achieve perpetual operation, which is characterized by the system's ability to continuously function without ever running out of energy. This is accomplished by carefully managing the energy harvested from renewable sources and balancing it with the system's energy consumption. The perpetual operation of ENO systems ensures that these systems remain functional and reliable, meeting their intended application goals while minimizing the need for external energy sources or frequent battery replenishment.

2. *Intermittent Computing* (IC) devices, where the system operates with high consumption until the energy is fully depleted, causing the sensor to shut down. When the energy level is sufficient again, operation returns to normal, and the process repeats. IC devices, by design, operate on very small capacitors and low-power harvesting methods, allowing for tiny-sized sensors. As a result, they are suitable for wearables and other applications where seamless integration into the surroundings is crucial.

The concept of ENO, proposed by Kansal et al. (2007), serves as a fundamental condition that an energy harvesting system must meet. Over time, two distinct classes of energy management techniques for ENO systems have emerged, as identified by Hanschke (2022): *planning* approaches and *real-time* approaches.

- *Planning*: approaches assume that a sensor node's activities can be predetermined. Typically, application requirements, such as a specific sampling rate, are given and incorporated into the energy management strategy. The primary goal is to meet or exceed the application requirements while maintaining energy neutrality.

- *Real-time*: approaches consider node activities as spontaneous, occurring in small segments (i.e., tasks), often described by statistical models of arrival rates. These tasks must be scheduled promptly to avoid missing deadlines while still considering energy consumption.

Our research will concentrate on real-time approach, as it offers numerous advantages for more effective energy management across various applications and environments. Real-time approach is particularly suitable for systems that require adaptability, responsiveness, robustness, scalability, and less dependence on forecasting to maintain energy neutrality and achieve optimal performance. By focusing on this approach, we aim to explore and develop efficient energy management strategy for mobile robotic systems that can be applied to a wide range of scenarios and applications. These

techniques globally satisfy the *non-functional requirements* (ref: Section 2.1.2) of the mobile robotic system and contribute to the *functional requirements* accordingly.

3.3.1 Energy-neutral Operation

Energy-neutral Operation (ENO), as explored by [Kansal et al. (2007)], focuses on the relationship between the power harvested from an energy source ($P_h(t)$) at a given time t and the power consumed by the system ($P_c(t)$) at the same time. The authors proposed three cases to model the energy behavior of the system and establish conditions for energy conservation:

1. *Harvesting system with no energy storage*: In this case, the energy extracted from the environment is directly used by the system without being stored.
2. *Harvesting system with an ideal energy buffer*: This scenario assumes an ideal mechanism for storing any amount of harvested energy.
3. *Harvesting system with a non-ideal energy buffer*: This is a more practical case, where the energy capacity is limited, charging efficiency is less than perfect, and some energy is lost through leakage.

Our focus is on the third case, as it represents a more realistic scenario. The energy conservation conditions for this case are given by:

$$E_b(0) + \rho \int_0^T [P_h(t) - P_c(t)dt]^+ dt - \int_0^T [P_c(t)dt - P_h(t)dt]^+ - \int_0^T P_l(t)dt \geq 0 \quad \forall T \in [0, \infty) \tag{3.1}$$

Here, $E_b(0)$ is the initial energy stored in the energy storage device, ρ represents the charging efficiency, $[x]^+$ defines the rectifier function[3], and P_l is the leakage power of the storage unit. However, this equation does not consider the capacity of the energy storage unit (E_b).

$$E_b(0) + \rho \int_0^T [P_h(t) - P_c(t)dt]^+ dt - \int_0^T [P_c(t)dt - P_h(t)dt]^+ - \int_0^T P_l(t)dt \leq E_b \quad \forall T \in [0, \infty) \tag{3.2}$$

[3] $[\mathrm{x}]^+ = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$

It is important to note that while condition 3.1 is both *necessary and sufficient*[4] for all allowable $P_h(t)$ and $P_c(t)$, condition 3.2 is only *sufficient* but not *necessary*. Some functions that do not satisfy this condition may still be allowable, as excess energy that is not used or stored in the buffer can be dissipated as heat from the system. In such cases, the left-hand side of (3.1) will be strictly greater than zero, by the amount of energy wasted. The condition (3.2) becomes *necessary* if wasting energy is not allowed.

The real-time computing community has tackled the challenges of ENO through numerous scheduling strategies, detailed in the following subsection. These strategies aim to ensure energy neutrality while satisfying the timing constraints and energy demands of tasks in RTEHS.

3.3.2 Energy-aware Scheduling in RTEHS

In recent years, the real-time computing domain has seen a significant increase in research focusing on scheduling strategies for real-time tasks on computing systems. These strategies aim to satisfy both timing and energy constraints. Unlike *DVFS*, which aims to minimize global energy consumption, energy-aware scheduling dynamically manages available energy sources, such as harvested environmental energy and stored battery energy. This approach determines suitable periods for executing real-time tasks and replenishing energy storage to ensure continuous operation, typically assuming a monoprocessor with two modes: *active* and *standby*. Our research seeks to develop versatile energy-aware scheduling approaches for mobile robotic systems that maintain perpetual operation while accommodating diverse system requirements and constraints. We review algorithms from the literature for scheduling tasks in real-time energy harvesting systems. These systems often involve a processing unit that executes tasks with deadlines, consuming energy stored in a reservoir after production by an environmental source, as depicted in Figure 3.4. In this context, a scheduling algorithm is considered *optimal* if it produces a feasible schedule each time another scheduler does so under the same conditions, including identical energy harvester and storage unit characteristics.

Allavena and Mosse (2001) presents a study on offline algorithms for scheduling periodic tasks with a common deadline, known as "frames," where the task execution order does not impact schedulability. The power scavenged by the energy source is assumed to be constant, and tasks consume energy at a constant rate. Tasks are categorized into recharging and discharging groups. The scheduler executes tasks from

[4]Refer to Section 2.3.4

the same category successively, selecting discharging tasks to decrease the energy level in the storage unit and recharging tasks to increase it, maintaining the energy level between minimum and maximum limits. Preemption occurs when the energy level reaches one of these limits. While this work is among the first to focus on rechargeable systems with hard real-time constraints, it has limitations, including a restrictive model (frame based systems), and an offline scheduler that lacks flexibility for new-generation real-time applications.

An early and significant work in the realm of real-time scheduling with energy harvesting considerations is the Lazy Scheduling algorithm (LSA) [Moser et al. (2006)]. LSA serves as an idling version of the EDF scheduling algorithm, enabling the processor to idle based on energy availability. With the goal of maintaining high energy storage levels, LSA executes tasks only when specific conditions are fulfilled. The algorithm introduces the concept of the energy variability characterization curve (EVCC) to assess the schedulability of a task set. While LSA is proven to be optimal under certain conditions [Moser et al. (2007)], it has limitations, such as assuming energy consumption is proportional to execution time and necessitating continuous adjustment of consumption power to the source power.

Abdedda et al. (2014) introduced a fixed-priority real-time scheduling approach for energy-harvesting systems, which serves as an idling variant of the well-known non-idling FP (Fixed Priority) scheduler. The algorithm schedules tasks according to a fixed priority and executes them as soon as there is sufficient energy in the storage unit for at least one time unit of a task. When energy is insufficient, the algorithm replenishes the energy storage, but only to the extent needed to execute one time unit of the job with the highest priority. This approach aims to balance task execution with energy consumption, ensuring tasks are processed as soon as enough energy is available, thus minimizing idle time. Despite these developments, the question of whether an optimal algorithm exists for a general energy-harvesting system model remains unanswered.

A potential solution could involve combining virtual deadlines with lookahead computation, but the complexity of such an algorithm is exponential. The existence of an optimal algorithm for fixed-priority energy-harvesting systems and the problem's potential NP-hardness are still open questions that warrant further investigation [Chetto and Queudet (2014a)].

From Table 3.2, it is evident that scheduling algorithms for RTEHS predominantly focus on independent real-time tasks. This presents a significant limitation, as these scheduling algorithms cannot be easily integrated into more realistic real-time systems

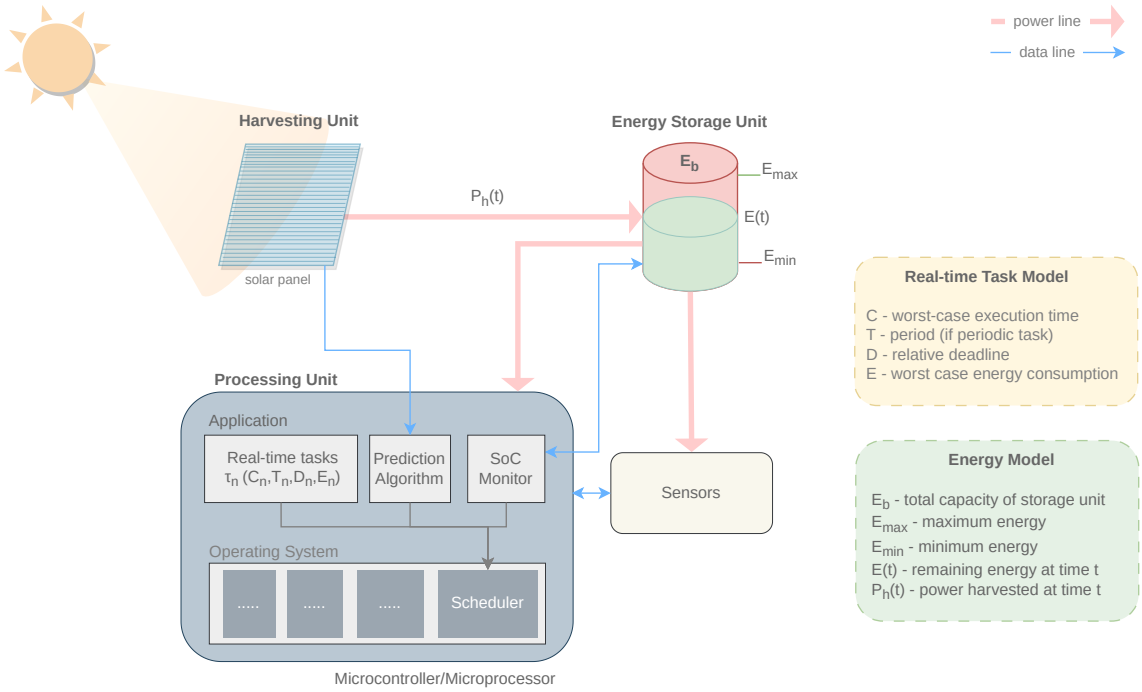


Figure 3.4 Model of Real-time Energy Harvesting System.

Research Works	Scheduling Algorithm	Task Dependency
Allavena and Mosse (2001)	DVS	Independent
Moser et al. (2007)	LSA	Independent
Abdedda et al. (2014)	PFPasap	Independent
Chetto (2014)	ED-H	Independent

Table 3.2 Existing literature of real-time scheduling strategies for RTEHS.

where processes interact to meet system-wide requirements. The nature of these interactions is diverse, including synchronization for *shared resources*, as outlined in Section 2.3.2. In mobile robots processing platform, tasks are not independent and frequently share resources, rendering the existing scheduling algorithms unsuitable for such applications.

In this dissertation, our primary focus is on the scheduling algorithm proposed by Chetto (2014), known as the *Earliest-Deadline Harvest* (ED-H). While the ED-H scheduling algorithm only considers *independent* tasks, we contribute to this research by taking into account *dependent* tasks and providing an enhanced schedulability test. Throughout the course of this research, we implement the algorithm as a novel scheduling class within the Xenomai-patched Linux kernel. The limitations of the implementation and methods to overcome these limitations are detailed in

the contribution chapters of this dissertation. With the ED-H scheduling strategy, we aim to maximize the utility for mobile robotic systems by successfully fulfilling both functional and non-functional requirements in a harmonized manner, ensuring a well-balanced and efficient system performance. In the following section, we provide an in-depth explanation of the ED-H scheduling strategy, which will be frequently referred to throughout this dissertation.

3.4 Earliest-Deadline Harvest Scheduling

We provide an in-depth overview of the dynamic priority scheduler ED-H, proposed by Chetto (2014). To better understand the concepts of ED-H, we will first define the task model, energy model, and unique terminologies related to RTEHS. Our real-time energy harvesting system model, illustrated in Figure 3.4, comprises three primary components: a *processing unit*, an *energy storage unit*, and a *harvesting unit*.

The processing unit, which may be a microcontroller or microprocessor, utilizes energy to function. The energy storage unit can be a battery or a supercapacitor. The selection relies on aspects such as system dynamics, dimensions, and budgetary considerations. The harvesting unit is responsible for capturing energy from external sources like solar, wind, vibrations, kinetic, or chemical energy. The nature of the harvesting unit depends on the ambient energy type and the amount of energy required. We will now discuss the ED-H scheduler and its application in real-time energy harvesting systems.

3.4.1 Task Model

In this section, we examine a set of real-time jobs executed on a uniprocessor processing unit with a single clock rate. We adopt the periodic real-time task model $\tau = \{\tau_i(\Phi_i, C_i, D_i, T_i) | 1 \leq i \leq n\}$ from earlier (ref; Section 2.3.2) and introduce a new parameter, E_i, representing the worst-case energy consumption (WCEC). E_i is the maximum energy a task consumes while executing on the processor, measured in energy units. The real-time task model now uses a five-tuple representation: $\tau_i(\Phi_i, C_i, D_i, T_i, E_i)$.

For the sake of simplicity, we utilize a job set, comprised of task set jobs, in place of a task set to outline RTEHS concepts. We represent the set of n preemptible and *independent* jobs as $J = \{J_i(a_i, d_i, C_i, E_i) | 1 \leq i \leq k\}$. Each job has an activation time (a_i), a absolute deadline (d_i), a worst-case execution time (C_i), and a worst-case energy consumption (E_i). Notably, E_i is not necessarily a function of C_i, meaning a job's effective energy consumption does not linearly depend on its effective execution time

[Jayaseelan et al. (2006)]. For each time unit, an upper bound (e_{ub} units of energy) is known for any job's energy consumption. However, the exact amount of effectively consumed energy in a time unit is unknown beforehand.

We use $d_{max} = \max_{0 \leq i \leq k} d_i$ and $D_{max} = \max_{0 \leq i \leq n} D_i$ to represent the largest absolute deadline and the largest relative deadline for the jobs in task set τ, respectively. The energy consumption by jobs within the time interval $[t_1, t_2)$ is indicated by $E_c(t_1, t_2)$. Similar to the processor utilization of a real-time task set (i.e., $U_p \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{C_i}{T_i}$), we define the energy utilization of the task set τ as $U_e \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{E_i}{T_i}$, which characterizes the average energy consumption of τ per time unit.

3.4.2 Energy Model

In addressing the energy considerations of the system, we formulate an energy model that encompasses both *energy storage* and *power harvesting* components. At any given moment t, the power harvester (e.g., solar panel) captures ambient energy and converts it into electrical power through an instantaneous charging rate $P_h(t)$, taking into account all losses. The energy harvested during the time interval $[t_1, t2)$ is denoted as $E_h(t_1, t_2)$:

$$E_h(t_1, t_2) = \int_{t_1}^{t_2} P_h(t) dt \tag{3.3}$$

We assume that energy production and consumption can occur simultaneously. The energy consumption during any unit time slot is no less than the energy generated within the same time slot. Consequently, the remaining capacity of the energy storage does not increase when a job is being executed. Instead, the processor must draw any additional energy required for a job from the storage.

The energy output from the source is variable and not necessarily constant. Despite this, it can be precisely forecasted in the near future with minimal processing and energy expenses. We consider an energy storage unit (e.g., a rechargeable battery) that maintains functionality even in the absence of energy to harvest. The maximum amount of energy storable at any given moment is represented by the *nominal capacity*, E_b.

The energy storage receives power from the harvester and delivers it to the processor. The energy stored at a specific time t is symbolized by $E(t)$. The energy storage retains energy without any loss over time. If the storage remains fully charged at time t while charging continues, energy is squandered. In contrast, if the storage is entirely depleted at time t (energy exhaustion), no job can be performed.

We consider energy to be wasted if the storage unit remains fully charged but is still being charged. Conversely, the storage unit is deemed fully depleted at time t if $0 \leq E(t) < E_{max}$, indicated by $E(t) \approx 0$. The system begins with the energy storage unit at full capacity (i.e., $E(0) = E_b$). The energy stored within the unit can be utilized at any later point without experiencing energy loss over time.

Types of Starvation

In the scope of the RTEH model, a job may fail to meet its deadline due to one of the following two circumstances:

- *Time Starvation*: This occurs when a job of a task reaches its deadline at time t with an incomplete execution, as the necessary processing time for finishing the job before the deadline is insufficient. The energy storage unit has remaining energy when the deadline violation occurs (i.e. $E(t) > 0$).
- *Energy Starvation*: This situation arises when a job of a task reaches its deadline at time t with an incomplete execution, due to the unavailability of the required energy for finishing the job before the deadline. The energy storage is depleted when the deadline violation takes place. The energy in the storage unit is exhausted when the deadline violation occurs (i.e. $E(t) \approx 0$).

RTEHS-specific Definitions

We now present new definitions specifically tailored to scheduling in RTEH systems.

Definition 5. *Time-validity: A schedule* Γ *for jobs of task set* τ *is said to be time-valid if the deadlines of all jobs of* τ *are met in* Γ*, considering that* $\forall i \in \{1, ..., n\}, E_i = 0$.

Definition 6. *Time-feasibility: The job set* J *is said to be time-feasible if there exists a time-valid schedule for* J.

Definition 7. *Energy-validity: A schedule* Γ *for the task set* τ *is said to be energy-valid if the deadlines of all jobs of* τ *are met in* Γ*, considering that* $\forall i \in \{1, ..., n\}, C_i = 0$.

Definition 8. *Energy-feasibility: The job set* J *is said to be energy-feasible if there exists a energy-valid schedule for* J.

Definition 9. *Energy-clairvoyance: A scheduling algorithm* χ *is considered energy-clairvoyant if it necessitates knowledge of future energy production for making runtime decisions.*

3.4.3 Description of ED-H

ED-H is a variation of the *EDF* scheduling algorithm. The traditional EDF is a greedy scheduler as it executes jobs *as early as possible*, utilizing the energy stored in the storage unit without accounting for future energy needs. We discuss the types of starvation that might arise with EDF, underlying the explanation of the ED-H algorithm. Assuming a set of jobs that EDF can schedule as time-feasible, energy starvation for a job J_j can only occur due to the execution of a job J_i that runs before J_j arrives, with $d_i > d_j$. The energy starvation of J_j resulting from J_i with $d_i \leq d_j$ is unavoidable by any scheduler. Clearly, *clairvoyance* regarding job arrivals and energy production would allow EDF to predict energy starvation and deadline violations. As a result, the primary idea behind ED-H is to permit job execution solely when starvation is not a risk. Clairvoyance in both job arrivals and energy production enables ED-H to anticipate potential energy starvation.

The decision-making process in ED-H is based on the online computation of two crucial data: *slack time* and *preemption slack energy*. These terms are essential for understanding the algorithm and are defined as follows:

Slack Time

Definition 10. *The slack time (ST) represents the maximum continuous processor time that can be made available from time t while still guaranteeing the deadlines of all the jobs in job set J.*

The slack time of a real-time job set J at the current time t is given by

$$ST_J(t) = \min_{d_i > t} ST_{J_i}(t) \tag{3.4}$$

Here $ST_{J_i}(t)$ is the *slack time* of the job J_i at the current time t defined as:

$$ST_{J_i}(t) = d_i - t - h(t, d_i) - AT_i \tag{3.5}$$

In this equation, $h(t, d_i)$ refers to the total processing *time demand* of the uncompleted jobs at time t with deadline at or before d_i. Here, AT_i denotes the total remaining execution time of uncompleted jobs that are currently ready at t with absolute deadline at or before d_i.

$$h(t_1, t_2) = \sum_{i}^{n} W_i(t_1, t_2).C_i \tag{3.6}$$

$W_i(t_1,t_2)$ provides the maximum number of job instances of a task τ_i in the time interval $[t_1,t_2)$, which is given by:

$$W_i(t_1,t_2) = \max\left(0, \left\lfloor \frac{t_2 - a_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - a_i}{T_i} \right\rceil + 1\right) \tag{3.7}$$

Preemption Slack Energy

Definition 11. *The preemption slack energy (PSE) is the maximum energy that can be consumed by the currently active job at time t while still guaranteeing energy feasibility for jobs that may preempt it.*

Let d be the absolute deadline of the active job. The preemption slack energy of the job set J at the current time t is given by:

$$PSE_J(t) = \min_{t<d_i<d} SE_{J_i}(t) \tag{3.8}$$

Here, $SE_{J_i}(t)$ represents the *slack energy* of job J_i at current time t. It is the maximum surplus energy that the system can consume within the time interval $[t,di)$ while guaranteeing enough energy for jobs released at or after t and with a deadline at or before d_i.

$$SE_{J_i}(t) = E(t) + E_h(t - d_i) - g(t,d_i) \tag{3.9}$$

In this equation, $g(t,d_i)$ is the total *energy demand* of the uncompleted jobs at time t with deadline at or before d_i.

$$g(t_1,t_2) = \sum_{i}^{n} W_i(t_1,t_2).E_i \tag{3.10}$$

These definitions and equations form the basis for understanding the computation of slack time and preemption slack energy in the ED-H scheduling algorithm. By taking these factors into account, the algorithm can efficiently schedule tasks in energy-constrained environments while guaranteeing the deadlines and energy feasibility of all jobs.

3.4.4 ED-H algorithm Specification

The ED-H scheduler incorporates more than just a single rule for selecting a job that is ready for execution. It also features a rule for dynamically managing processor activity

by designating intervals when the processor should remain *idle* (standby) and those when it should *execute* a job. Let $\mathcal{L}(t)$ represent the list of ready jobs at the current time t.

Rule1: Use EDF-assigned priorities to choose the next job from the ready list $\mathcal{L}(t)$

Rule2: The processor remains *idle* during $[t, t+1)$ if $\mathcal{L}(t) = \emptyset$

Rule3: The processor remains *idle* during $[t, t+1)$ if $\mathcal{L}(t) \neq \emptyset$ and one of the following condition is true:

(a) $E(t) \approx 0$

(b) $PSE_J(t) \approx 0$

Rule4: The processor is *busy* during $[t, t+1)$ if $\mathcal{L}(t) \neq \emptyset$ and one of the following condition is true:

(a) $E(t) \approx E_b$

(b) $ST_J(t) = 0$

Rule5: The processor can be either *busy* or on *idle* if $\mathcal{L}(t) \neq \emptyset$, $0 < E(t) < E_b$, $ST_\tau(t) > 0$ and $PSE_\tau(t) > 0$.

The ED-H scheduler is renowned for its adaptability and energy efficiency, ensuring that energy is conserved and alerting the system promptly about any negative slack time or preemption slack energy situations. The implementation involves several rules, *Rule 1* selects the job with the earliest deadline form the ready list. *Rule 2* denotes the processor state as *idle* when there are no jobs in the list. *Rule 3*, prevents job execution if the storage unit is almost empty or running the job would cause an unavoidable energy shortage due to insufficient preemption slack energy. *Rule 4* dictates that the processor should be *busy* if the storage unit is full or remaining idle would lead to a missed deadline due to zero slack time. *Rule 5* allows the processor to be either on *idle* or *busy* without jeopardizing the subsequent schedule's validity under specific conditions, such as the storage unit being neither empty nor full and the system having, both non-zero slack time and preemption slack energy. Unique cases derived from *Rule 5* include *ASAP* and *ALAP* approaches, which involve executing jobs either *as early as possible* when there is sufficient energy or *as late as possible* without exceeding the storage unit's maximum capacity. The ED-H variant depends on the rule chosen for initiating and concluding the storage charging phase, with one example being to

execute jobs when the energy level reaches a specific threshold and transition the processor to *idle* mode to recharge the storage unit when the charge level falls below another predefined threshold.

3.4.5 Schedulability Analysis

We recall that the schedulability test serves as an essential mechanism for validating the feasibility of a job set processed through the scheduling algorithm. Under ED-H scheduling this examination determines whether every job can meet its timing and energy constraints. The schedulability test for ED-H encompasses both static (offline) and dynamic (online) analysis. In particular, dynamic analysis proves to be indispensable for applications where job arrivals are not known a *priori*. By evaluating the computed *slack time* and *preemption slack energy*, dynamic analysis enables the scheduler to make informed decisions during runtime. Thus, it guarantees that jobs can meet their timing and energy constraints.

Static Analysis

The static analysis method utilizes an approach equivalent to the processor demand method of EDF to calculate *static slack time* and *static slack energy*. The Static Slack Time (SST) between $[t_1, t_2)$, denoted as $SST_J(t_1, t_2)$, represents the longest interval within $[t_1, t_2)$ during which the processor can stay idle while still ensuring the execution of jobs of set J with activation times at or after t_1 and deadlines at or before t_2:

$$SST_J(t_1, t_2) = t_2 - t_1 - h(t_1, t_2) \tag{3.11}$$

From this, we can deduce the static slack time of the set τ:

$$SST_J = \min_{0 \le t_1 < t_2 \le d_{max}} SST_J(t_1, t_2) \tag{3.12}$$

Demonstrating that $SST_J \geq 0$ proves that the job set J is schedulable by ED-H in the absence of energy constraints.

The Static Slack Energy (SSE) between t_1 and t_2, denoted as $SSE_J(t_1, t_2)$ is the maximum amount of energy available during the interval $[t1, t2)$ while still ensuring the execution of the jobs of J with activation times after t_1 and deadlines at or before t_2:

$$SSE_J(t_1, t_2) = E_b - E_h(t_1, t_2) - g(t_1, t_2) \tag{3.13}$$

We can then define the static slack energy of J as follows:

$$SSE_J = \min_{0 \leq t_1 < t_2 \leq d_{max}} SSE_J(t_1, t_2) \tag{3.14}$$

The static slack energy of J represents the energy surplus that can be consumed at any time while still guaranteeing that the energy requirements of the jobs of J will be met.

Schedulability Condition

To assess the feasibility of an application, it must be schedulable under the chosen RTEHS model. The feasibility test considers the properties of its components, particularly the storage unit's capacity and the harvesting unit's power production. Theorem 1 demonstrates that time and energy constraints can be evaluated separately, dividing the feasibility test into time and energy feasibility tests. Thus, an application is feasible if and only if it is both time-feasible and energy-feasible [Chetto (2014)].

Theorem 1. *[Chetto (2014)] A job set J conforming to the RTEHS model is feasible if and only if*

$$SST_J \geq 0 \text{ and } SSE_J \geq 0 \tag{3.15}$$

The time feasibility test has a complexity of $O(n^2)$, as the static slack time calculation requires n^2 different intervals. If ambient energy can be predictively estimated for each time interval using a finite number of values, the energy feasibility test will also have a computational complexity of $O(n^2)$.

3.4.6 Illustrative example of ED-H

In order to facilitate the understanding of how ED-H operates, we will examine a periodic task schedule. We will consider a task set with attributes defined as follows: `task_name (offset, wcet, relative deadline, period, wcec)`.
τ_1 `(0,1,6,6,13)`, τ_2 `(0,3,10,10,33)`, τ_3 `(0,2,15,15,18)`.

For simplicity and clarity of the ED-H concept, we adopt a number of assumptions. Firstly, the power harvested during one time instance, denoted as P_h, is always a constant value equal to 7, with storage unit begin recharged at the beginning of every time unit. Secondly, power is consumed from the storage unit solely during the execution of the jobs; which implies there is no power consumption when the processor is standby. Lastly, during job executions, the power consumed per unit is determined

by the ratio of the jobs wcec to its wcet. The storage unit is at its maximum capacity $E_b = 30$ energy units at time $t = 0$.

We apply Theorem 1 to test the feasibility of the jobs. As SST_J and SSE_J is equal to 0, this proves that there exists a time-valid and energy-valid schedule for the job set, thereby inferring that the task set is guaranteed to be schedulable by ED-H.

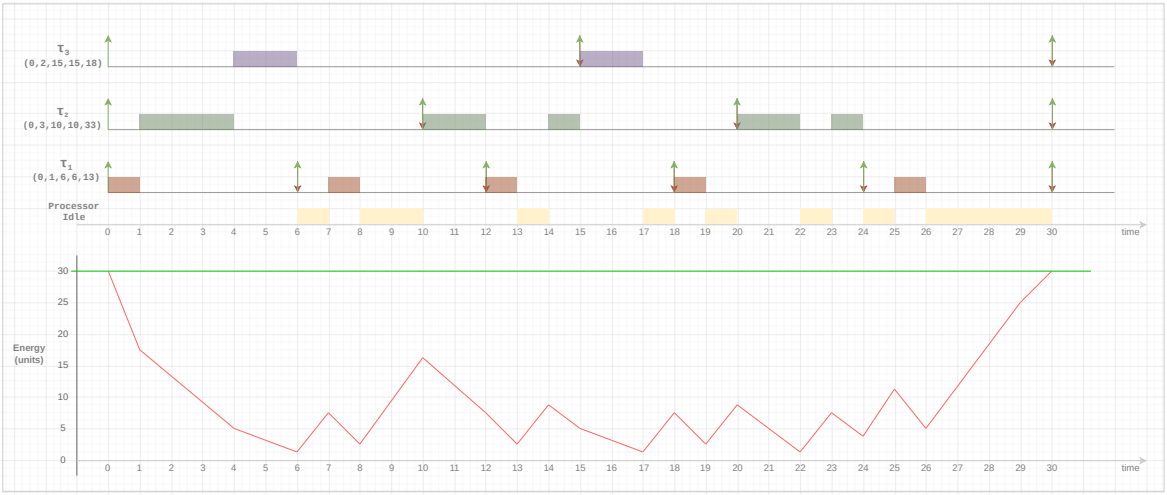


Figure 3.5 The ED-H schedule for the three tasks is presented, with the energy level of the storage unit depicted by a red line. The maximum capacity of the storage unit, marked by a green line, is 30 energy units.

We will now construct the ED-H schedule. At time $t = 0$, three jobs are in the ready list (i.e., $\mathcal{L}(0) = 3$). The scheduler first applies Rule 1 to select the earliest deadline job, $\tau_{1,0}$, from the ready list. Then performs the check by calculating ST_J with Equation 3.4 and PSE_J with Equation 3.8. Rule 5 is satisfied and we adopt the ASAP approach, which chooses the processor to execute the job $\tau_{1,0}$, followed by the next job $\tau_{2,0}$ and then $\tau_{3,0}$.

At $t = 6$, job $\tau_{1,1}$ is released and ST and PSE are calculated as before. Since there is no sufficient energy to execute the job at that time, implying Rule 3, the processor is put on standby to recharge the storage unit. At $t = 7$ there is sufficient energy to complete the execution of job, thus Rule 5 is applied. In the same way, ST and PSE are calculated whenever a new instance of a job is present in the ready list, and the respective Rules are applied.

In this schedule, the energy unit is at its maximum capacity at the start which is 30 energy unit. When the schedule completes the first hyperperiod, denoted as $H = 30$, the energy unit is fully replenished. Figure 3.5 illustrates the schedule of the task set and depicts the energy level of the storage unit. It is evident that no job misses its

deadline and the energy level in the storage unit never depletes below the minimum energy level E_{min}, which in this case 1 energy unit.

3.4.7 Properties of ED-H

Optimal Analysis

Formally proven *optimal* [Chetto (2014)], the ED-H scheduling algorithm outperforms any other scheduler under the same hardware and energy conditions when constructing a valid schedule for a set of jobs J.

Theorem 2. *[Chetto (2014)] The ED-H scheduling algorithm is optimal for the RTEHS model.*

Optimality is established by recognizing that job deadlines are missed due to either time or energy shortages. In essence, ED-H creates a valid schedule if there isn't a time interval where both the processor demand surpasses the interval size and the energy demand exceeds the total available energy.

Clairvoyance

The following theorem gives an important restriction on ED-H. It precisely gives the length of the time interval in future where prediction is required.

Theorem 3. *[Chetto (2014)] ED-H scheduling algorithm operates as an online lookahead-D.*

As shown by Chetto and Queudet (2014a), no online scheduling algorithm can achieve optimality without at least D units of time clairvoyance. To make a decision at any time t, ED-H needs to know both the job arrival process and the energy production process during the subsequent D units of time. Regarding clairvoyance, ED-H's *lookahead-D* performance remains unmatched by any other scheduler.

A notable shortcoming of the ED-H scheduling algorithm is its inability to account for dependent tasks. This consideration is crucial, as many processing applications in mobile robotics systems involve shared resources, leading to task dependencies. Consequently, the need to incorporate and analyze tasks that share resources serves as a key driving force behind this research. Additionally, the complexity of implementing the ED-H scheduling algorithm within a real-time operating system (RTOS) remains an open research area. Addressing this challenge is vital for further exploration and practical application of the algorithm in real-world systems.

In the following sections, we will explore the uncertainties of energy harvesting systems that integrate works from various domains. This exploration will provide a deeper understanding of the challenges associated with energy management in real-world scenarios and help us develop more effective solutions for mobile robotic systems that incorporate energy harvesting technologies.

3.5 Energy-neutral Operations uncertainties

Energy management for ENO systems has been extensively studied through theoretical contributions. However, achieving the ENO condition in practice can be quite challenging due to several factors, including energy consumption uncertainty, energy harvesting uncertainty, and energy storage uncertainty. These uncertainties are briefly explained below:

1. **Energy consumption uncertainty**: The power consumption of a computing system can be influenced by various factors, making it difficult to accurately predict. Importantly, the scheduling schemes often assume that parameters such as task execution time and energy consumption are known a priori using static analysis, which is difficult to estimate or predict in practice. Separate studies exist for analyzing Worst-case Execution Time (WCET) [Puschner and Schedl (1997), Wenzel et al. (2008), Wilhelm et al. (2008)] and Worst-case Energy Consumption (WCEC) [Jayaseelan et al. (2006), Wagemann et al. (2015), Sieh et al. (2017), Wägemann et al. (2018), Eichler et al. (2019)], but they are complex and challenging to adapt for different operating systems.

2. **Energy harvesting uncertainty**: Harvested energy from renewable sources is inherently unpredictable. While patterns in the energy source can be learned over time, the exact amount of energy that will be available at any given moment is difficult to determine. This uncertainty adds complexity to the energy management process, as it is necessary to adapt to changing energy availability.

3. **Energy storage uncertainty**: The characteristics of energy storage systems, such as charging and discharging rates, can also introduce uncertainty. Furthermore, determining the actual energy capacity of a storage system, as opposed to its nominal capacity, can be a complex task. Energy storage devices, like batteries or capacitors, may also experience degradation over time, which affects their performance and capacity.

These uncertainties must be carefully managed in order to achieve an accurate estimation of energy neutrality and optimize the performance of ENO systems. In real-time systems, scheduling strategies often depend on worst-case analyses of tasks running on the system. However, these worst-case analyses can be challenging to explore. As a result, we will further investigate approaches that specifically address these uncertainties, particularly in the context of mobile robotic systems. By examining these approaches, we aim to better understand how to manage the uncertainties inherent in energy consumption, harvesting, and storage for such systems.

3.6 Energy Harvesting Systems

Energy harvesting, also known as *energy scavenging or power harvesting, involves capturing and converting ambient energy from the environment into electrical energy.* This harvested energy can subsequently be utilized to power or recharge the systems, effectively extending their operational capabilities and reducing dependence on conventional energy sources.

Numerous energy harvesting technologies have been developed to capture energy from various sources, including solar, wind, vibration, temperature, and radio frequency. This dissertation focuses on the identification and evaluation of suitable harvesting technologies for mobile robotic systems. In this section, we present an overview of different technologies and highlight the most appropriate harvesting technology that is widely adopted for wireless sensor networks, specifically solar harvesting. Additionally, we discuss its suitability for mobile robot systems, as well as the limitations and complexities of harvesting technology for industrial mobile robots.

3.6.1 Harvesting Technologies

Mobile robot systems can be equipped with various energy harvesting technologies, depending on the application and environmental conditions. In the following, we discuss some suitable technologies that can be implemented individually or as a hybrid, depending on the energy demand of the robotic system [Liang et al. (2022)]:

1. *Solar energy harvesting*: Solar energy harvesting captures energy from sunlight using photovoltaic (PV) cells. It is a widely practiced and well-established technology, suitable for outdoor robots and applications where sunlight is readily available. Solar energy harvesting is clean, renewable, and offers a virtually unlimited energy source during daylight hours.

2. *Piezoelectric energy harvesting*: Piezoelectric energy harvesting converts mechanical strain or vibrations into electrical energy using piezoelectric materials. This technology is particularly useful in applications where robots are subject to constant movement or vibrations.

3. *Triboelectric nanogenerators* [Barkas et al. (2019), Yang et al. (2023)]: Triboelectric nanogenerators harvest energy from the contact or separation of two different materials, generating an electric charge due to the triboelectric effect. This technology can be applied in scenarios where robots experience frequent contact with surfaces or objects, such as robotic grippers or wheeled robots.

4. *Regenerative braking* [Canfield et al. (2019), Ko et al. (2019)]: Regenerative braking captures the kinetic energy lost during braking or deceleration and converts it into electrical energy. This technology is especially relevant for mobile robots with frequent stop-and-go movements, allowing them to recover energy otherwise wasted as heat during braking.

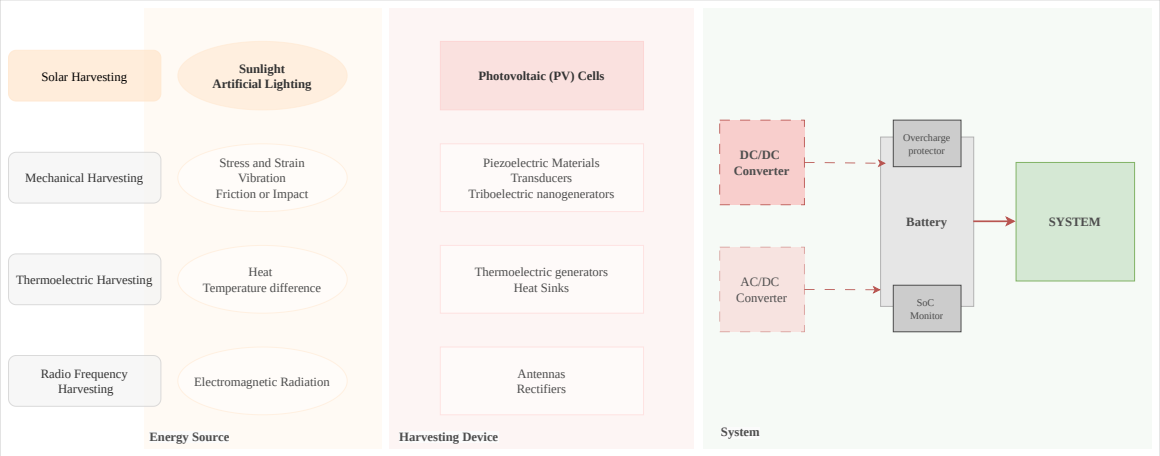


Figure 3.6 Highlighting the various energy-harvesting technologies that can be adapted for mobile robotic system.

Table 3.3 shows the comparison of power densities provided from various literature sources for different harvesting technologies. This helps to identify the suitable harvesting technology for the scalability of energy demand of the system. Among the various energy harvesting technologies, solar energy harvesting remains efficient and is widely practiced and extensively explored due to its abundant availability, renewability, and eco-friendliness. The advancement of photovoltaic materials has increased interest in exploring the harvesting of energy from indoor ambient lights. We further discuss

Harvesting Technology	Power density	Condition
Solar - Outdoor [Hamadani et al. (2021)]	36 $\frac{mW}{cm^2}$	Mono-crystalline Silicon PV cell under Air Mass 1.5 Global (the standard for 1000 W/m2 irradiance with the sun's spectrum)
Light - Indoor [Shore et al. (2021)]	0.2 $\frac{mW}{cm^2}$	Mono-crystalline Silicon PV cell under white LED with a CCT of 3000K 1000 lx illuminance
Vibrations [Los et al. (2004)]	0.3 $\frac{mW}{cm^3}$	
Thermal [Los et al. (2004)]	0.017 $\frac{mW}{cm^3}$	
Radio Frequency [Vaussard (2015)]	0.003 mW	
TENG [Seung et al. (2020)]	0.5 mW	Textile-based tire cord TENG running at 1000rpm

Table 3.3 Comparison of harvested power density from various harvesting technologies under different conditions.

the advancement and limitations of solar energy harvesting for outdoor and indoor applications, which will be the primary concern in this dissertation.

3.6.2 Solar Energy Harvesting

Solar energy harvesting has emerged as a powerful and sustainable approach to meeting the energy needs of various applications, including mobile robotic systems. It involves capturing sunlight and converting it into electrical energy using photovoltaic (PV) cells, which are semiconductor devices that generate electricity when exposed to sunlight.

PV cells, typically composed of semiconductor materials like silicon or gallium arsenide (GaAs), form the backbone of solar energy harvesting systems. When exposed to sunlight, these semiconductor materials release electrons, which are then collected by metal contacts to generate an electric current. The performance of a solar cell is influenced by several factors, including the intensity of sunlight, the angle of incidence, and the ambient temperature. The qualitative characteristics of solar cells are crucial in understanding their performance. These characteristics include:

1. *short-circuit Current* (I_{sc}): It is the maximum current generated by the solar cell when the voltage across the cell is zero. This occurs when the cell is under illumination but not connected to any external load.

2. *open-circuit Voltage* (V_{oc}): It is the maximum voltage generated by the solar cell when there is no current flowing through it. This occurs when the cell is under illumination but not connected to any external load.

3. *Fill Factor* (FF): It is the ratio of the maximum power output of the solar cell to the product of I_{sc} and V_{oc}. It represents the efficiency of the solar cell in converting sunlight into electricity.

Another significant aspect of solar energy harvesting is the azimuth angle, which is the horizontal angle between the sun's position and the solar cell's surface. This angle affects the incident angle of sunlight on the solar cell and, consequently, its energy conversion efficiency. To optimize energy harvesting, the solar cell should be oriented to minimize the azimuth angle.

Solar energy harvesting is also affected by solar irradiance i.e., the amount of sunlight energy received per unit area per unit time, usually expressed in watts per square meter (W/m^2). Solar irradiance varies with time of day, weather conditions, and geographical location. Lux, a unit of measurement for illuminance, indicates the intensity of light falling on a surface. Table 3.4 demonstrates the lux intensity in various environments, which affects the power harvested.

Environment	Typical Lux
Direct Sunlight	32K to 100K
Ambient Daylight	10K to 25K
Factory, Workshops	1K
Office, Laboratories	500
Warehouse Aisles	200

Table 3.4 Light Intensity in different environments [5]

The energy conversion efficiency of a solar cell depends on factors such as the semiconductor material, cell quality, and operating conditions. Illumination levels are influenced by controllable and uncontrollable factors. Uncontrollable factors include intensity and weather conditions, while controllable factors include deployment site and solar panel orientation. The maximum power that can be harvested depends on the solar harvesting unit's surface area. Typically, solar panels are manufactured in specific sizes, but for a prototype system, it is essential to build a panel by connecting mini cells to cover the maximum possible area the system can accommodate, meeting the power requirements of the storage units. Arranging PV cells in series (increasing voltage) and parallel (increasing current) increases the power harvested with the increase in area.

Solar panel efficiency is influenced by manufacturing parameters, such as materials used, cell layer thickness, and coating, as well as environmental parameters like snow, dust, temperature, and aging. Environmental parameters, an active research topic [Farahmand et al. (2021)], are challenging to predict and often neglected in simple solar cell models. Solar cell load characteristics can be further understood through I-V (current-voltage) curves and P-V (power-voltage) curves (Figure 3.7). The curves are graphical representations of the relationship between current, voltage, and power

[4] https://greenbusinesslight.com/

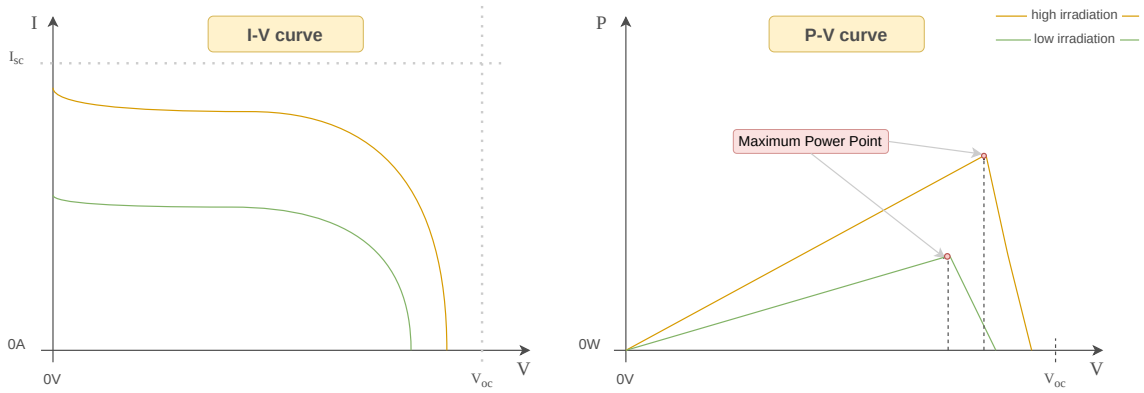


Figure 3.7 Load characteristics of a PV cell.

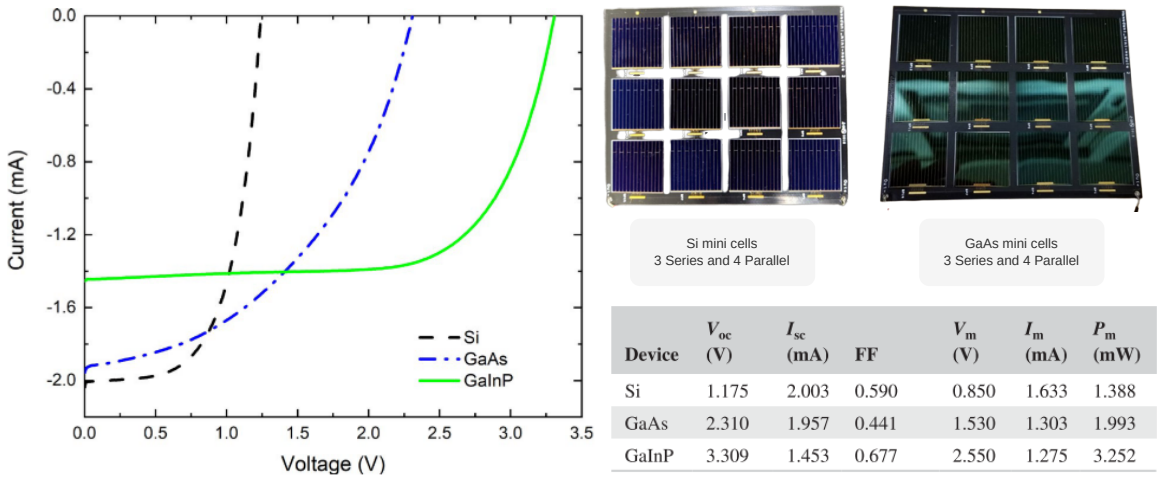
under varying illumination conditions. The Maximum Power Point (MPP) on these curves ensures optimal energy conversion efficiency. According to [Jeong and Culler (2012)], the MPP mainly depends on the illumination level and is challenging to configure statically. Input load matching techniques are a trade-off between efficiency and harvested power [Khosro Pour et al. (2013), Ram et al. (2017)].

Efficiency of Indoor Solar Energy Harvesting

In recent years, there has been a growing interest in indoor photovoltaic (PV) energy harvesting as an alternative energy source for powering indoor mobile robots, particularly those used in industrial settings. Since most industrial mobile robots operate indoors, harvesting energy from indoor lighting can significantly contribute to their energy efficiency, extending their operational capabilities, and reducing reliance on conventional energy sources.

Indoor PV cells are specifically designed to efficiently capture energy from artificial light sources, such as fluorescent, incandescent, and LED lights commonly found in indoor environments. These cells differ from outdoor solar panels in terms of material composition, energy conversion efficiency, and spectral response to better suit indoor lighting conditions. Operating at lower light intensities compared to outdoor solar panels, indoor PV cells exhibit higher energy conversion efficiency under such conditions, making them well-suited for applications where robots need to function in areas with limited natural sunlight.

Recent research by Shore et al. (2021) and Hamadani et al. (2021) has conducted experiments to evaluate the efficiency of different PV cell materials under indoor environment conditions. Their results, as shown in Figure 3.8, indicate that different



Device	V_{oc} (V)	I_{sc} (mA)	FF	V_m (V)	I_m (mA)	P_m (mW)
Si	1.175	2.003	0.590	0.850	1.633	1.388
GaAs	2.310	1.957	0.441	1.530	1.303	1.993
GaInP	3.309	1.453	0.677	2.550	1.275	3.252

Figure 3.8 I-V curve measurements for the three solar cells Si, GaAs, and GaInP under 1000 lux [Shore et al. (2021)].

materials have improved harvested power. Interestingly, these types of PV cells can be utilized both indoors and outdoors.

To our knowledge, there are no existing studies exploring the integration of photovoltaic energy harvesting systems into industrial mobile robots. This presents an exciting and challenging research opportunity, as harnessing energy from both outdoor and indoor lighting can significantly improve the energy efficiency and performance of these robots. However, one critical and challenging aspect that needs to be addressed is harvest profiling.

Harvest profiling involves predicting the future energy availability, which is essential for real-time energy-neutral scheduling strategies that rely on the timing window of prediction techniques. This aspect will be discussed in the following section, providing valuable insights for the development and implementation of energy harvesting systems in industrial mobile robots, thereby advancing their energy efficiency and overall performance.

3.6.3 Solar energy harvest prediction methods - A closer look

Solar irradiation fluctuates throughout the year and day due to the Earth's path around the Sun and its rotation, leading to significant variations in power harvested from solar panels. As a result, energy-harvesting systems in mobile robotics must monitor the incoming energy and estimate future intake to efficiently manage the decision making system.

In the literature, a common approach to represent the observed harvest is to use discrete time intervals or time slots (T_s). The length of the time slots (T_l) depends on the profile horizon (T_h), which can range from short-term systems that track the diurnal cycle using a T_h of 24 hours to long-term systems that account for yearly fluctuations. When predicting energy, it is essential to determine a representative value for the energy harvested in each time slot. One established approach is to use the average value, calculated by taking the mean of collected samples in each time slot. However, using the average value presents challenges, and literature [Renner and Turau (2012)] suggests that time slot lengths below 30 minutes do not significantly increase energy-harvesting system benefits. For instance, the average value does not account for imbalances in energy harvesting within the time slot. This issue becomes more critical when the storage level is near the minimum. Balancing the need for more precise energy predictions against computational complexity and memory requirements is crucial when selecting an appropriate method for energy prediction in mobile robotic systems operating on such short timescales.

Energy prediction techniques are essential for effectively managing harvested energy. Multiple algorithms have been developed that use past observations to estimate future harvest conditions, as listed in Figure 3.9. The Exponentially-weighted Moving Average (EWMA) [Kansal et al. (2007)] filter is a widely used approach, providing a memory-efficient, low-complexity method for estimating future harvest in the same slot on the next day. Other approaches, such as the Weather-conditioned Moving Average (WCMA) [Piorno et al. (2009)] and Profile Energy Prediction Model (Pro-Energy) [Fong (2017)], attempt to improve the accuracy of short-term estimation by incorporating weather information and typical daily harvest representations, respectively. However, these methods may not be suitable for longer-term predictions or may introduce additional complexity. For indoor environments, Stricker and Thiele (2022) employ a random forest approach to predict energy harvesting. Yamin and Bhat (2021) propose a novel hierarchical machine learning model that takes into account recent history and daily variations to accurately predict future energy availability. Additionally, they suggest using online learning to adapt the model to seasonal and spatial variations in harvested energy.

For applications with missions characterized in seconds, it is essential to carefully select a prediction method that balances accuracy, computational complexity, and memory requirements to ensure efficient and reliable robot operation. This consideration is vital for the successful integration of energy prediction techniques into mobile robotic

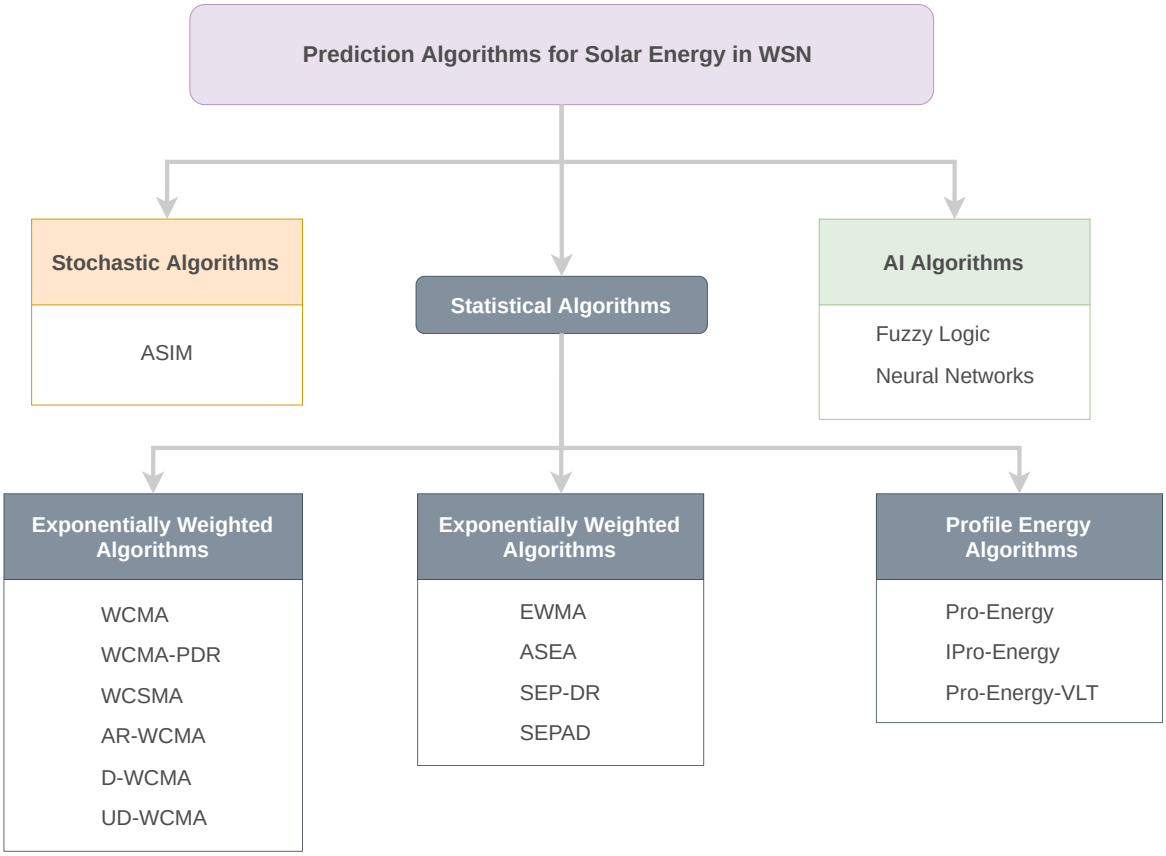


Figure 3.9 Overview of Solar Energy Prediction Algorithms for Energy Harvesting Systems.

systems, ensuring that the chosen method aligns with the systems specific operational requirements and constraints.

3.7 Storage Systems - Powering mobile robots

Industrial robots, apart from Autonomous Mobile Robots (AMRs), are predominantly stationary and rely on a direct and continuous energy supply. In contrast, the mobility of AMRs necessitates an onboard energy source with limited capacity. The selection of an energy source for such systems is a design-level decision and depends on the robot's requirements. Addressing energy supply and operational longevity issues could lead to more sustainable and commercially viable robots for new applications, thereby driving further industrial expansion. Rigorous research is essential for improving existing

solutions and exploring new options for powering robots [Gurguze and Turkoglu (2017), Farooq et al. (2023)].

In this context, environment and task-specific solutions may be more desirable and practical, as each task and environment imposes different constraints on the selection of powering technology. Offering multiple solutions with varying costs, operational times, and hardware setups for similar tasks and working environments could facilitate the commercialization of a broader variety of robots catering to diverse customer demographics. Currently, batteries, internal combustion engines, and fuel cells are the most commonly used power supply methods for AMRs Farooq et al. (2023). Industrial AMRs predominantly utilize batteries, with a wide range of options available on the market. Thorough research is crucial for selecting the appropriate battery based on system requirements.

Parameter	Lead Acid	Li-ion polymer	Super capacitors	Li-ion
Energy Density (Wh/Kg)	30-50	100-130	1-10	110-130
Recharge Cycle (upto 80% of initial capacity)	200-300	300-500	10^6	500-10000
Fast Charging Time	8-16h	2-4h	1-10s	2-4h
Self-discharge	weeks	months	days	long months
Nominal Voltage (V)	2V	3.6V	1-3V	3.6V
Charge rate (C)	5-0.2	2-1	-	2-1
Operating Temperature (°C)	-20 to 60	0 to 60	-40 to 85	-20 to 80

Table 3.5 Comparison of energy storage technologies Buchmann (2017).

Table 3.5 presents a comparison of various battery technologies. While supercapacitors are widely used and adapted for Wireless Sensor systems, AMRs require high energy density, which is satisfied by Li-ion batteries. Consequently, Li-ion batteries are the most popular battery technology for industrial mobile robots. There is a growing interest in exploring new battery technologies using sodium-ion, with sustainability being a significant advantage in a world transitioning away from carbon-based energy sources [Abraham (2020)]. Researchers are increasingly focused on providing solutions and materials for fast battery charging.

Due to the complexity and limitations of charging technologies, there is considerable interest in utilizing ambient energy sources. Although numerous harvesting technologies could address charging limitations, charging battery efficiency through harvesting technologies remains a challenge. Gibson and Kelly (2010) demonstrated that high system efficiency was achieved by directly charging the battery from a photovoltaic (PV) system without intervening electronics and matching the PV maximum power point voltage to the battery charging voltage at the desired maximum state of charge.

Accurately measuring the State of Charge (SoC), which identifies a battery's energy level, is another challenge in battery storage systems. Several methods exist for obtaining Lithium-Ion State of Charge (SoC) measurements or Depth of Discharge (DoD) for a lithium battery. Some methods, such as impedance spectroscopy or hydrometer gauge for lead-acid batteries, are complex and require specialized equipment. Partovibakhsh and Liu (2015) proposed a method for estimating Li-ion battery SoC using an unscented Kalman filtering method. The two most common and straightforward methods for estimating a battery's state of charge are the voltage method or Open Circuit Voltage (OCV) and the coulomb counting method. These methods are widely practiced, and components for estimating SoC using these methodologies are readily available for integration with robotic systems.

3.8 Conclusion

In this chapter, we have explored the energy management strategies practiced for mobile robotics. The approaches reviewed primarily focus on energy saving, resulting in an increased operation time for mobile robots. However, these strategies do not fully address the functional and non-functional requirements (FR and NFR) of the system/software architecture in mobile robotics, which encompass both performance and resource utilization (time and energy).

To address these limitations, we introduce the energy-neutral concept, which is novel for the mobile robotics community. An energy neutral system is defined as one that adapts its energy consumption regarding the energy it may receive from an external source. While some robots utilize an energy harvester to recharge its energy supply, they lack the intelligence needed to avoid exhausting the energy storage. The energy-neutral concept has mainly been extensively explored in computer engineering to effectively operate small embedded computing devices such as sensor nodes.

In this chapter, we have presented a brief state of the art related to the scheduling issue for the RTEHS model that comprises tasks with deadlines and energy harvesting considerations. In most research studies, the objective is to minimize the total energy consumed by software to maximize the application's lifetime or the duration between battery recharges. In contrast, scheduling under energy harvesting settings aims to guarantee energy neutrality, ensuring that the system never consumes more energy than harvested while satisfying real-time constraints expressed by deadline success.

This scheduling issue is more complex than in real-time systems with no energy limitations due to variations in the energy produced by the source, which may not

align with the timing requirements of real-time tasks. Consequently, any scheduling algorithm must include a power management procedure capable of deciding when to make the processor busy and for how long, and when to let the processor idle, allowing for energy storage unit recharging.

We have described the optimal dynamic scheduler ED-H which is dedicated to independent tasks. One shortcoming we identified is that its implementation cannot be integrated into a realistic context like mobile robots, where the tasks interact through shared resources. Extending the ED-H scheduling algorithm to consider resource access and implementing the algorithm under a real-time operating system remains an open area of research.

Additionally, we discussed the uncertainties that must be addressed to implement a complete real-time energy harvesting solution for mobile robotics systems. These include energy harvesting uncertainty, which encompasses solar power harvesting technology and prediction algorithms for estimating future harvested energy, and energy estimation for the remaining energy in the storage unit. Both quantities are required to guarantee the performance of the energy-aware scheduling algorithm ED-H.

Chapter 4

Methodology: Designing Real-time Deterministic Industrial Mobile Robots

In this chapter, we tackle the unique challenges associated with industrial mobile robot (IMR) architectures. Unlike other software systems, these robots need to interact in real-time with the unpredictable and often rapidly changing environments. This necessitates architectures that can handle real-time responses, manage sensors and actuators, process tasks concurrently, and seamlessly integrate high-level planning with low-level control. Despite these needs, the existing architectures of IMR fall short, causing difficulties in programming, configuration, and adherence to real-time requirements. This issue is discussed extensively in Section 4.1. Therefore, this chapter is dedicated to re-imagining and rebuilding such an architecture to not only meet industrial standards but also to be user-friendly and easily maintainable.

We aim to create a robust robot system that is predictable at all times during operation. We discuss this in Section 4.2, detailing necessary hardware and software modifications to enhance the system's functional predictability, maintainability, and usability. Next, in Section 4.3, we explore the significant role of a real-time Linux kernel and a dual kernel architecture in guaranteeing deterministic software performance. Following this, we justify our choice of the Jetson AGX processing platform for our redesign in Section 4.4. This platform aligns perfectly with the needs of an IMR. Finally, we detail our approach to enhance the real-time performance and predictability of the robot using the ROS 2 architecture in Section 4.5. Here, we focus on adjusting the scheduling abstraction to increase the functional behavior and reliability of the ROS2 applications. By the end of this chapter, our aim is to provide developers with a thorough guide for achieving the highest degree of performance and integrity of IMR to meet the ISO standard.

4.1 Problem Formulation

We analyze the challenges encountered by a generic IMR architecture. This architecture is generally divided into two primary categories: the hardware architecture which comprises computing units and various peripheral components interfaced for the mobile robot's functioning, and the software architecture which includes entities that control the operational behavior of the peripheral components. Collectively, these aspects pose challenges that hinder the mobile robot from fulfilling industrial demands effectively while maintaining product quality and user satisfaction.

Considering the generic architecture of the mobile robot illustrated in Figure 2.1, the system includes two computing units: high-level and low-level processing units. A microcontroller unit is selected as the low-level processing unit to control peripheral components such as actuators and sensors. An x86 processor is chosen to manage the mobile robot autonomously, utilizing robot middleware such as ROS. Figure 4.1 shows the two processing units, and highlights how the velocity computed from x86 machine is transferred to the microcontroller unit. This one converts and communicates the speed values to motor drivers using CANopen [Farsi and Ratcliff (1997)] library.

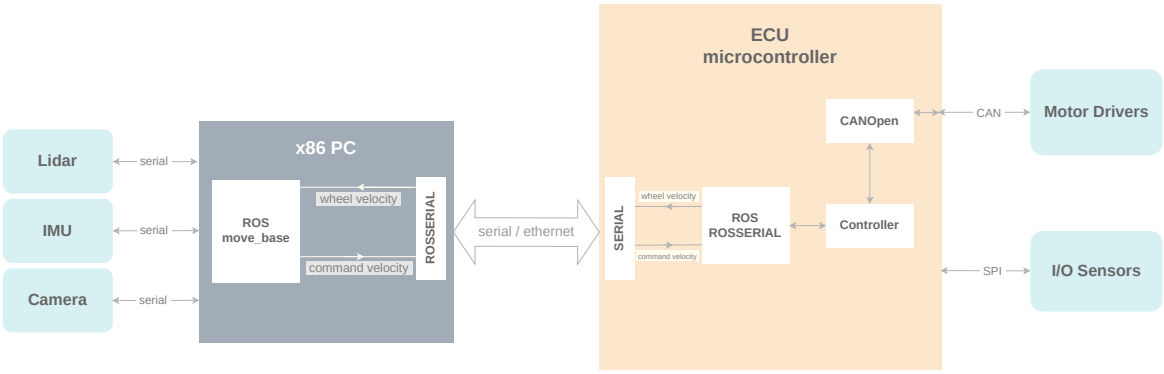


Figure 4.1 Processing platforms in mobile robot highlighting the navigation system.

Several difficulties emerge with this architecture in industrial settings, beginning with challenges for integrators or developers. Debugging low-level firmware that controls peripheral components can be arduous, mainly due to limited accessibility when the robot is in motion. This issue, which we term as the difficulty to *program*, *reprogram*, or *reuse* the software for user-specific requirements, leads to complications.

The next obstacle lies in *reconfigurability*. In this context, reconfigurability refers to adjusting the mobile robot according to user specifications. At times, users might need to alter the robot's maximum speed or interface different peripheral components. This process necessitates reprogramming, which is often a complex task. Overall,

reconfiguration demands physical access to the mobile robot, which is time-consuming and often difficult, affecting the product's usability.

The second major issue concerns the *overload* management of applications within the high-level processing units. If not chosen wisely, the processing platform could lead to the product's performance degradation. For instance, a processing platform with four cores could potentially be overloaded due to improper thread and resource management, leading to decreased performance and the product becomes unpredictable.

The third challenge involves the *inter-process communication reliability* between the high-level processing unit and the low-level processing unit. Consider a case as shown in Figure 4.1, where the ROS robot middleware, which controls navigation and operates on the high-level processing unit, delivers a command velocity. This command, defining the robot's speed needed to reach the goal, is processed by the low-level unit to transfer the necessary instructions to the motor drivers. Additionally, the low-level processing unit reads the wheel velocity from encoders and transmits the information to the high-level processing unit. The information exchange occurs via ROS topics using specific libraries such as *rosserial*[1], typically through a serial or ethernet communication link between the two processing units. However, this setup can lead to delays and potential data loss, negatively impacting the autonomous control of the robot in dynamic real-world environments. This is primarily because *rosserial* works only as a minimal experimental support [Macenski et al. (2022)].

The final issue concerns *energy management*. All the electronic components interfaced in the mobile robot consume power. Therefore, it is vital to design an architecture and select components that optimize low power consumption. Addressing these issues is paramount to deliver a simple architecture that ensures *product quality* and *usability*. The following sections propose solutions to overcome these challenges, emphasizing the need for an architecture that harmoniously balances performance, user satisfaction, and energy efficiency.

4.2 Product Architecture and Design Principles

Our primary objective is to benchmark an architecture that effectively addresses the above issues, thereby offering a robust product suitable for industrial applications and fulfilling user requirements. For instance, employing an Industrial Automation PC, such as Programmable Logic Controllers (PLC) based PC, as a low-level processing

[1]http://wiki.ros.org/rosserial

unit could mitigate the difficulties associated with reprogramming and reconfiguration [Kim and Shin (2017), Xie et al. (2018)].

However, this architecture does come with its own set of challenges. The primary issue is that these PCs operate on the Windows operating system, and some features are not open-source. This limitation hinders developers when they need to incorporate new functionalities. A key drawback persists in the form of inter-process communication latency between high-level processing units. Moreover, this architecture would lead to greater energy consumption, as industrial PCs consume more power than microcontrollers. Additionally, the cost associated with automation PCs is considerably high. Therefore, although the use of industrial automation PCs can enhance the product's usability to some extent, it still presents disadvantages that preclude meeting industrial requirements and ensuring product quality. Even if we were to continue using microcontrollers as low-level processing units, we could achieve improvements in thread management and reduce communication latency through efficient library usage. This could be done by employing commercially supported libraries like *micro-ROS* [Belsare et al. (2023)]. However, such an approach still falls short in terms of product usability, specifically in programming and reconfiguration.

Given these factors, it is evident that designing an architecture that satisfies all requirements is a challenging task. In response to these challenges, we propose a robust architecture designed with a two-fold emphasis: hardware considerations and software considerations. This approach strives to balance performance, energy efficiency, and user satisfaction, all of which are critical for the successful deployment of mobile robots in industrial settings.

4.2.1 Hardware Consideration

The hardware of a mobile robot pertains to the physical components integrated within its system, such as motors, industrial sensors, and processing units. These elements communicate using established industrial protocols like the CAN (Control Area Network) and I/O (Input-Output) links.

Our approach concentrates on the processing platform. It is essential for this unit to comply with all the necessary requirements and successfully mitigate the challenges we have previously identified. Primarily, the processing platform must be capable of integrating industrial sensors through suitable peripheral interfaces. Secondly, the platform should be designed for easy programmability and reconfigurability, with no requirement for physical access. An ideal approach would involve a single processing platform that fulfills these prerequisites. This would negate the need for a separate

low-level processing unit and, in turn, eliminate inter-process communication issues between the high and low-level processing units. After careful consideration, we have identified the Jetson PCs, specifically the Jetson AGX Xavier or Jetson AGX Orin models, as the most suitable options to meet all peripheral requirements to interface the components for autonomous mobile robots.

Figure 4.2 Hardware consideration for IMRs.

Figure 4.2 shows the simplest hardware consideration for mobile robot. This unified processing platform enables programming and reconfiguring through network access, thereby eliminating the need for physical contact. Furthermore, it supports all necessary industrial peripherals, facilitating seamless interaction with industrial sensors. Additional I/O link devices can be implemented to augment sensor interface capabilities. The Jetson platform can function across different power modes, providing an added advantage of reconfigurability. Furthermore, its support for machine learning functionalities enhances the operational capabilities of mobile robots in industrial settings[2] [Schmid et al. (2020)].

By taking into account this hardware consideration, we not only enhance the product's usability but also identify a platform that supports functional accuracy, performance, and reliability. However, a thorough understanding of these qualities requires an in-depth examination of the software architecture of the processing platform.

4.2.2 Software Consideration

Mobile robots rely significantly on their software, which determines their functional and non-functional properties. The complexity of robot software systems arises largely from the need to manage various sensors and actuators in real time, despite the presence of considerable uncertainty and interference. Furthermore, the robot must be capable of

[2]https://github.com/dusty-nv/jetson-reinforcement

accomplishing specific tasks while simultaneously detecting and responding to unanticipated circumstances. Performing all these activities together and asynchronously escalates the system's complexity.

However, implementing a robust architecture, complemented with fitting programming tools, can assist in handling this complexity, as suggested by Kortenkamp et al. (2016). It is important to remember that no one architecture suits all needs. Each architecture comes with its own set of strengths and weaknesses. Ahmad and Babar (2016), undertook a detailed exploration of issues concerning software architectures for robotic systems. They mapped key trends from 1990 to 2015, namely object-oriented, component-based, and service-driven robotics. However, these architectural approaches often lacked the validation of quality-specific or architecturally crucial requirements. In response to these challenges, the Robot Operating System (ROS) was developed [Quigley et al. (2009)]. This flexible framework was designed for developers to build upon, enabling the creation of robot software systems that can cater to a multitude of hardware platforms, research contexts, and runtime demands. Despite its versatility, ROS lacked some essential production-grade features and algorithms. Its limitations in security and performance made ROS1 unsuitable for safety-critical applications [Macenski et al. (2020)].

To address these limitations, ROS2 was introduced, offering industrial-grade features such as enhanced security, reliability, and real-time capabilities, vital for the next generation of robots. A critical part of this upgrade was the incorporation of the Data Distribution Service (DDS) [Pardo-Castellote (2003)]. DDS, with its flexible transport configurations and scalability, proved suitable for real-time distributed embedded systems. The adoption of ROS2 significantly contributed to the rapid, reliable deployment of real robot systems in a range of challenging settings. García et al. (2020) conducted an extensive empirical study to understand the practical aspects of robotics software engineering. They identified several requirements for robotic software. Many of these requirements, including *quality assurance*, *reuse*, and *interoperability*, are satisfactorily addressed by ROS2. However, the aspect of *deterministic execution* still represents an unresolved challenge.

Figure 4.3 provides a complete depiction of the proposed software architecture. It distinctly demarcates the user space modules and kernel space modules. User space modules are those which users can directly access and manipulate. Conversely, kernel space modules operate at the system level and can undergo modifications during the installation of the processing platform. In our architecture, we employ a Linux[3]

[3]https://www.kernel.org/

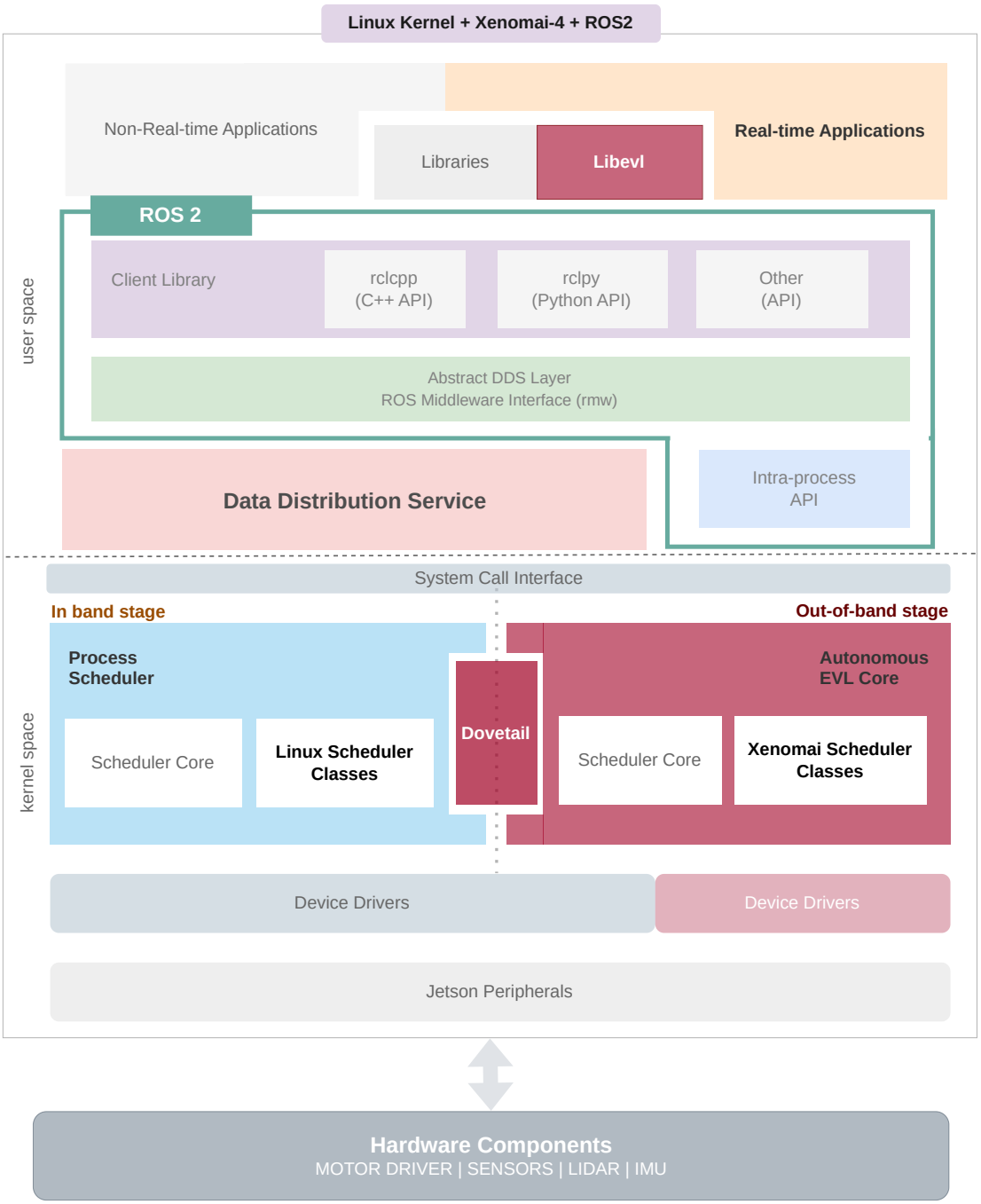


Figure 4.3 Software abstraction for industrial mobile robot. Highlighting the user space and kernel space modules.

operating system that is patched with Xenomai[4] kernel running on Ubuntu[5] distro. The goal of this combination is to establish a unified system that can execute high-

[4]https://evlproject.org/
[5]https://ubuntu.com/

level operations while also managing the control of external components (hardware components of IMR). More specifically, the Xenomai kernel is designated to handle control threads of external components, particularly those with stringent timing requirements. In mobile robot software, it is essential for practitioners to distinguish between non real-time and real-time applications. In the following sections, we will explore real-time applications in more depth and provide reasons for our choice of a Xenomai kernel-enhanced Linux operating system for robotic applications. We will also delve into the deterministic characteristics inherent in ROS2.

4.3 Real-time Linux

Before exploring the details of real-time Linux, it is important to recall a few aspects previously discussed in Section 2.3.3. Firstly, it is crucial to understand that having a Linux kernel alone does not necessarily transform an operating system into a real-time one. A prevalent misconception is that real-time equates to optimized performance. This is generally a misinterpretation stemming from systems, often labeled as real-time, that have efficient enough performance to prevent any human-perceivable deadline failures [Lelli (2014)]. However, a real-time Linux kernel does not necessarily enhance performance optimization [Madden (2019)]. Instead, it aims to produce a deterministic response to an external event, with the objective being to minimize response latency rather than optimizing throughput. According to a widely accepted definition, in a real-time system, the correctness of a computation relies not only on the logical accuracy of the result but also on the time it takes to produce it. If a system fails to meet its timing constraints, it is deemed to have failed. This definition aligns with the POSIX Standard 1003.1[6], which describes real-time responsiveness as an operating system's ability to deliver a specified level of service within a defined response time [Burns (2009)].

In essence, real-time systems are best suited for scenarios with extreme latency dependencies, where a missed deadline results in system failure, rather than mere performance degradation. Historically, industrial automation society have preferred to use PLCs for real-time performance. Recently, however, Linux-based industrial PLCs[7] have emerged. Despite their benefits, these PLCs lack versatility for all developers, leading to increased focus on industrial PCs for solutions with high computing demand. While it is imperative for these industrial PCs[8] to offer real-time performance, this

[6]https://standards.ieee.org/ieee/1003.1/7700/
[7]https://www.wago.com/global/automation-technology/discover-plcs/pfc100
[8]https://premioinc.com/blogs/blog/the-differences-between-industrial-pcs-and-plcs

can be achieved through specific kernel configurations of the operating system, and further enhanced with dual-kernel approaches. Therefore, developers must follow best practices in developing applications that deliver real-time performance.

4.3.1 PREEMPT_RT Linux

Developers face a challenging trade-off when seeking to reduce latency and achieve real-time computing capabilities: they must make the Linux kernel preemptible. The Linux kernel Scheduler[9] has several preemption models available:

- `CONFIG_PREEMPT_NONE`: No Forced Preemption (Server)
- `CONFIG_PREEMPT_VOLUNTARY`: Voluntary Kernel Preemption (Desktop)
- `CONFIG_PREEMPT`: Preemptible Kernel (Low-Latency Desktop)
- `CONFIG_PREEMPT_RT`: Fully Preemptible Kerenl (Real-Time)

The `PREEMPT_NONE` model is the default behavior in standard kernels, where forced preemption is not allowed. This is optimized for overall throughput, particularly in high-computation server workloads, but it doesn't guarantee low latency as kernel code is never preempted. The `PREEMPT_VOLUNTARY` mode allows for quicker application responses to user inputs, ideal for desktop use. This mode enables a low-priority process to voluntarily preempt itself even during a system call in kernel code. While this may slightly reduce throughput, it lowers the maximum rescheduling latency and improves the perceived smoothness of applications under load. The `PREEMPT` model, like `PREEMPT_VOLUNTARY`, enables voluntary preemption points in the Linux kernel, but it also allows kernel code to be involuntarily preempted outside critical sections. This can provide reduced kernel latencies suitable for desktop or embedded systems with latency requirements in the millisecond range, however at the cost of slightly lower throughput and increased runtime overhead to kernel code. `PREEMPT_RT`, a patch set hosted by the Linux Foundation, implements a priority scheduler and other real-time mechanisms. A significant portion of the `PREEMPT_RT` locking code was merged into recent Linux Kernel[10]. `PREEMPT_VOLUNTARY` is the default preemption model selected for desktop systems. Current Ubuntu release Ubuntu 22.04 have enabled `PREEMPT_DYNAMIC`[11], allowing the preemption model to be chosen at boot time.

[9] https://bootlin.com/doc/training/preempt-rt/preempt-rt-slides.pdf
[10] https://www.kernel.org/
[11] https://lore.kernel.org/lkml/87v90kcf7v.mognet@arm.com/T/

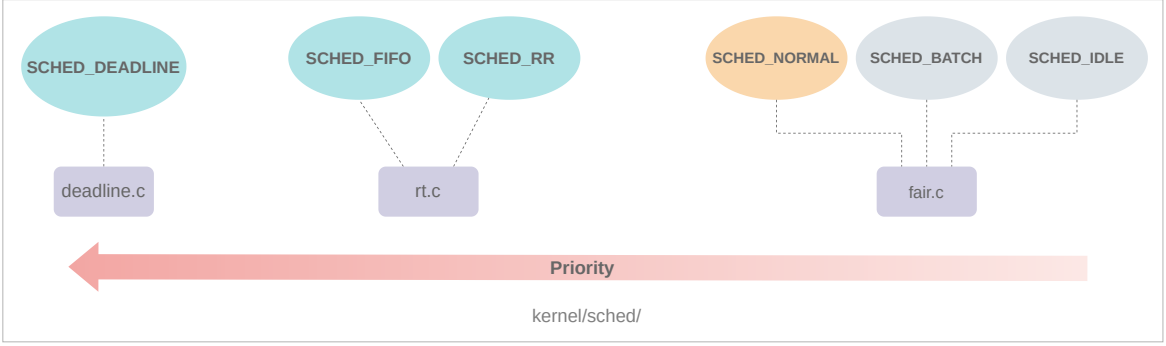


Figure 4.4 Linux Kernel scheduling class policies. Highlighting the real-time scheduling policies: `SCHED_DEADLINE`, `SCHED_FIFO`, `SCHED_RR` and non real-time "default" policy `SCHED_NORMAL`.

The Linux kernel scheduler plays a vital role in enabling real-time behavior as it determines which runnable thread should be executed. This extends to both user space threads and kernel threads. Each thread is assigned a scheduling class or policy, which defines the algorithm used for its execution. Thread with different scheduling classes coexist on the system. In Linux kernel, scheduling policies[12] are categorized into real-time and normal, with their priorities depicted in Figure 4.4. Real-time policies are `SCHED_FIFO`, `SCHED_RR`, and `SCHED_DEADLINE`. `SCHED_FIFO` and `SCHED_RR` have priority levels from 0 to 98 and take precedence over normal threads. Priority 99 is reserved for critical housekeeping threads [Madden (2019)]. `SCHED_FIFO` allows a thread to run uninterrupted unless a higher priority thread preempts it, blocks, yields, or terminates. `SCHED_DEADLINE`, introduced in kernel 3.14, manages periodic threads based on a specified deadline. A detailed explanation of `SCHED_DEADLINE` policy can be found in Lelli (2014). `SCHED_NORMAL`, `SCHED_BATCH`, and `SCHED_IDLE`, handle regular threads. `SCHED_IDLE` is employed when the processor is idle, and `SCHED_BATCH` allows threads to run longer with less frequent interruptions, which is suitable for batch jobs. `SCHED_NORMAL` is the *default* policy used for regular threads and is employed when a user application is created without defining a scheduling policy. It utilizes the *Completely Fair Scheduler* (CFS)[13] which aims to emulate an "ideal" multitasking CPU, sharing equal CPU power among running threads. Given the constraint of one thread execution at a time, CFS uses a "virtual runtime" concept, where the thread's actual runtime is normalized to the number of running threads.

[12]https://manpages.ubuntu.com/manpages/focal/man7/sched.7.html
[13]https://docs.kernel.org/scheduler/sched-design-CFS.html

Real-time Application

When designing a real-time application, adherence to a set of best practices is recommended, primarily because not all POSIX APIs were constructed with real-time functionality in mind. A crucial step in developing a real-time application is the configuration stage, usually performed within the application's initialization section. This involves setting up the parameters of the application, allocating memory, initializing locks, and starting threads. It also includes defining scheduling attributes such as priority levels and deadlines, along with specifying CPU affinity. Utilizing the standard C library is sufficient as it encompasses the POSIX real-time API. Among various C libraries, `glibc`[14] is highly recommended due to its mature support for certain real-time features. In UNIX, the creation of a process, generally involving the `fork()`[15] function, initiates an address space containing program code, data, stack, shared libraries, among others, and a thread that executes the `main()` function. While a process starts with one thread, additional threads can be incorporated using `pthread_create()`. These additional threads operate within the same address space as the initial thread and execute a function passed to `pthread_create()`.

The function `pthread_attr_setschedpolicy()` is used to set the scheduling policy of a thread. The existing system calls `sched_setscheduler()` and `sched_getscheduler()` have not been extended due to potential binary compatibility issues that could arise from modifying the `sched_param` data structure in existing applications. To circumvent this problem, two new system calls, `sched_setattr()` and `sched_getattr()`, have been introduced [Lelli (2014)]. These system calls are also compatible with the other existing scheduling policies. The interpretation of the arguments passed to these system calls depends on the selected policy. These calls are intended to supersede the previous system calls, which will be retained to avoid breaking existing applications. The prototype of these new system calls is shown in Listing 4.1. In Appendix B.1, we present the structure of thread creation, focusing on the setting of different scheduling policies, particularly `SCHED_FIFO` and `SCHED_DEADLINE`, as well as the attributes associated with these policies.

```
#include <sched.h>

struct sched_attr {
    u32 size;
    u32 sched_policy;
    u64 sched_flags;
```

[14] https://www.gnu.org/software/libc/
[15] https://man7.org/linux/man-pages/man2/fork.2.html

```
    /* SCHED_OTHER , SCHED_BATCH */
    s32 sched_nice;
    /* SCHED_FIFO , SCHED_RR */
    u32 sched_priority;
    /* SCHED_DEADLINE */
    u64 sched_runtime;
    u64 sched_deadline;
    u64 sched_period;
};

int sched_setattr(pid_t pid, const struct sched_attr *attr);
int sched_getattr(pid_t pid, const struct sched_attr *attr, unsigned
   int size);
```

Listing 4.1 `SCHED_DEADLINE` API.

The Linux kernel, even with the application of a real-time patch, does not provide guaranteed deadlines [Madden (2019)]. The kernel offers what is known as soft real-time behavior through its real-time scheduling policies. This soft real-time approach means that while the kernel strives to schedule threads within their timing deadlines, it does not assure that these deadlines will always be met. In contrast, hard real-time systems provide a guarantee that all scheduling requirements will be met within certain boundaries. This level of assurance is not offered by Linux, as it does not make any promises about the scheduling of real-time threads. However, the Linux scheduling policy does ensure that real-time threads are operational whenever they are runnable. Despite lacking a design to guarantee hard real-time behavior, the `PREEMPT_RT` locking code - which forms the majority of the remaining real-time patches - was integrated into Linux 5.15[16]. Yet, there are still more enhancements to be introduced upstream[17]. Currently, `PREEMPT_RT` is the accepted standard implementation of real-time Linux. However, users that demand very low latency and very high reliability of deadlines may be better served using a proprietary real-time kernel patch, such as Xenomai[18].

4.3.2 Xenomai patched Linux Kernel

Xenomai brings improved real-time abilities to Linux. It does this by adding a dedicated core into the kernel that handles tasks needing a fast and stable response time. This method is called a dual-kernel architecture. It allows some tasks to have strict real-time

[16] https://www.kernel.org/
[17] https://ubuntu.com/blog/what-is-real-time-linux-part-iii
[18] https://source.denx.de/Xenomai/xenomai

promises while others can have extensive operating system services. Here, the standard kernel and the real-time core work almost separately, each taking care of different tasks. The real-time tasks are always prioritized. We are mainly looking at Xenomai-4, the most recent version which uses the EVL core[19]. The EVL core was adapted from the Xenomai 3 Cobalt core[20] and redesigned to be more scalable and easy to maintain. However, both versions follow the same idea of dual-kernel technique. The introduction of the EVL project has simplified the integration of Xenomai into the Linux kernel. This is achieved by co-existing the EVL code with the Linux kernel. The activation of the `CONFIG_EVL` configuration results in the Linux kernel being equipped with EVL modules.

A valid question arises *"Why choosing Xenomai when we can run POSIX-based applications on a `PREEMPT_RT` kernel?"* The answer is simple. If the application already follows POSIX standards and meets performance needs, then Xenomai may not be needed. But Xenomai might be useful in certain cases:

- If the application requires bounded response times and very limited jitter, consistently.
- If the application needs reliable performance, meaning no kernel or user code should interfere with strict real-time deadlines.
- If the application's design needs the real-time execution to be separated from the general-purpose activities. This way, they will not share important subsystems like the common scheduler.
- If it is not possible or effective to use CPU isolation to reduce the impact of the non-real-time workload on the real-time side.

Certainly, Xenomai may not operate effectively with low-end hardware, particularly single-core CPUs, as the kernel needs at least one non-isolated CPU for system maintenance tasks. This can lead to a situation where the system might freeze. For instance, in a device with a single-core CPU, Xenomai may face challenges as the single processor must manage both system maintenance tasks and regular operations, which could potentially cause operational congestion, leading to system hangs. A key reason for choosing EVL for our work is its ability to reduce the difficulty associated with developing additional scheduling policies, a process that is facilitated by a well-documented guide of its internal functions.

[19] https://evlproject.org/overview/
[20] https://source.denx.de/Xenomai/xenomai/-/wikis/Introducing_Xenomai_3

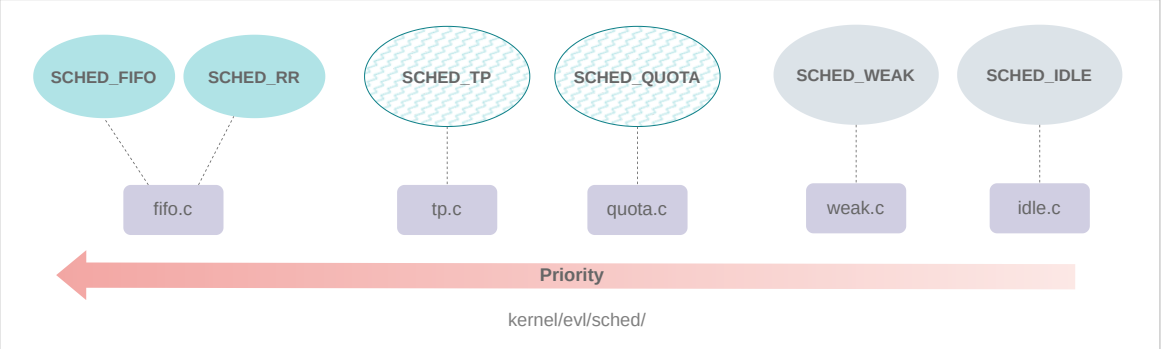


Figure 4.5 Xenomai kernel scheduling policies. `SCHED_FIFO` real-time policy, `SCHED_TP` and `SCHED_QUOTA` policies are supported optionally by the EVL core. To enable either `CONFIG_EVL_SCHED_QUOTA` or `CONFIG_EVL_SCHED_TP` must be set in the kernel configuration.

In the context of *out-of-band*[21] thread scheduling, the EVL core introduces its unique hierarchy of scheduling policies as depicted in Figure 4.5. These are checked in a particular sequence every time the system needs to determine the next eligible thread for execution on the current CPU.

- `SCHED_FIFO`: This stands for the well-known First-In, First-Out real-time policy[22]. Additionally, it also manages the Round-Robin policy, known as `SCHED_RR`, internally.
- `SCHED_TP`: This policy aims to manage temporal partitioning among multiple thread sets. Its function is to prevent any overlap of these thread sets on the CPU on which this policy is executed.
- `SCHED_QUOTA`: This policy serves to enforce a limit on CPU utilization by threads during a fixed time frame.
- `SCHED_WEAK`: This is a non-real-time policy that allows the threads to run primarily in-band but still permits them to request EVL services.
- `SCHED_IDLE`: This is the default fallback policy that the EVL core uses when there are no runnable tasks on the CPU for other policies.

[21]Within the EVL documentation, the term "out-of-band" is used to denote the EVL core, whereas "in-band" refers to the Linux core. This distinction helps differentiate the threads each core manages in the dual-kernel architecture.

[22]https://pubs.opengroup.org/onlinepubs/009695399/functions/xsh_chap02_08.html

It is essential to note that the `SCHED_QUOTA` and `SCHED_TP` policies are supported optionally by the EVL core. Out of all these policies, only `SCHED_FIFO`, `SCHED_QUOTA`, and `SCHED_TP` are real-time scheduling policies[23].

The EVL core needs to respond promptly to interrupts, irrespective of ongoing kernel operations. To circumvent this, Dovetail[24] uses an interrupt pipelining mechanism[25], allowing immediate handling of interrupts, no matter what activities the main kernel is performing. As a result, the EVL core can swiftly react to interrupts, while main kernel operations proceed mostly unaffected, except for minor pauses needed for out-of-band operations.

EVL Real-time application

```
#include <sys/types.h>
#include <stdlib.h>
#include <error.h>
#include <errno.h>
#include <evl/evl.h>
#include <evl/clock.h>
#include <evl/thread.h>
#include <pthread.h>

#define SCHED_PRIORITY 50  /* Priority */
#define FIFO_PERIOD_NS 1000000  /* 1ms */

void *threadfunc(void *parm)
{
    int ret = 0;
    struct evl_sched_attrs attrs;   /* evl scheduling attributes */
    struct evl_thread_stats stats;  /* thread statistics */

    attrs.sched_policy = SCHED_FIFO;
    attrs.sched_priority = SCHED_PRIORITY;
    attrs.fifo.period = FIFO_PERIOD_NS;

    ret = evl_attach_self("/threadfunc:%d", getpid());
    if (ret < 0)
         error(1, -ret, "evl_attach_self() failed");
```

[23] https://evlproject.org/core/user-api/scheduling/
[24] https://evlproject.org/dovetail/
[25] https://evlproject.org/dovetail/pipeline/

```
    ret = evl_set_schedattr(evl_get_self(), &attrs);
    if (ret < 0)
         error(1, -ret, "evl_set_schedattr() failed");

    /* Make the thread periodic which is handled within the kernel */
    evl_set_thread_periodic();
    while (1) {
         /* Wait for the thread period */
         evl_thread_wait_period(efd, &stats);

        /* Your code here */
    }
    return NULL;
}

int main(int argc, char *const argv[])
{
    pthread_t thread;
    struct sched_param param;
    cpu_set_t cpu_set;
    int ret;

    CPU_ZERO(&cpu_set);
    CPU_SET(0, &cpu_set);

    ret = sched_setaffinity(0, sizeof(cpu_set), &cpu_set);
    if (ret)
     error(1, errno, "cannot set affinity to CPU0");

    ret = evl_init();
    if (ret)
     error(1, -ret, "evl_init() failed");

    ret = pthread_create(&thread, NULL, threadfunc, NULL);
    if (ret)
        error(1, errno, "pthread_create() failed");

    pthread_join(thread, NULL);
    return 0;
}
```

Listing 4.2 EVL periodic application structure.

In the EVL environment, an application typically consists of one or more specialized EVL threads, which run alongside standard POSIX threads. EVL threads, once bound to the EVL core, use services provided by the `libevl`[26] library, which is a specialized library built to offer ultra-low latency and real-time services. This library plays a critical role in allowing EVL threads to achieve high-performance real-time operations. It is important to note that when an application uses libevl services, it should be linked against the libevl library. Initially, an EVL thread is just a regular POSIX thread. However, by calling the function '`evl_attach_self()`', the thread gains the ability to use EVL's real-time services, which are critical for meeting time-sensitive demands. Importantly, during these critical periods, the thread should avoid using common C library services (glibc), as doing so could compromise the real-time performance.

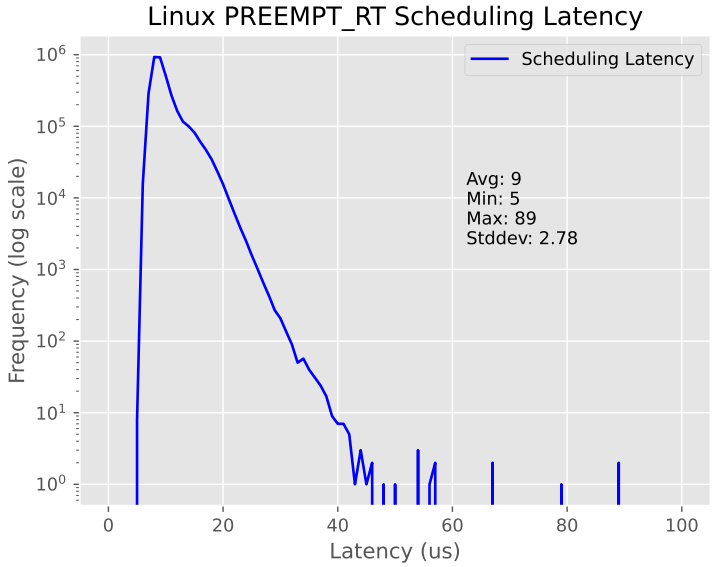
To facilitate the management of real-time periodic threads, we have incorporated two APIs into libevl: '`evl_set_thread_periodic()`' and '`evl_thread_wait_period()`'. The `evl_set_thread_periodic()` function makes a thread periodic by creating a periodic timer for it within the scheduling class, while `evl_thread_wait_period()` makes the thread wait for the timeout period and also provides any overrun status of the thread. Listing 4.2 presents a straightforward code structure that allows a conventional POSIX thread to utilize its real-time services after this thread has established an attachment to the EVL core. In a typical lifecycle of an EVL application, a thread first attaches to the EVL core, performs time-critical operations exclusively using EVL services, and then, when the critical operations are complete, the thread can use standard C library services for clean-up operations before exiting.

4.3.3 Scheduling Latency

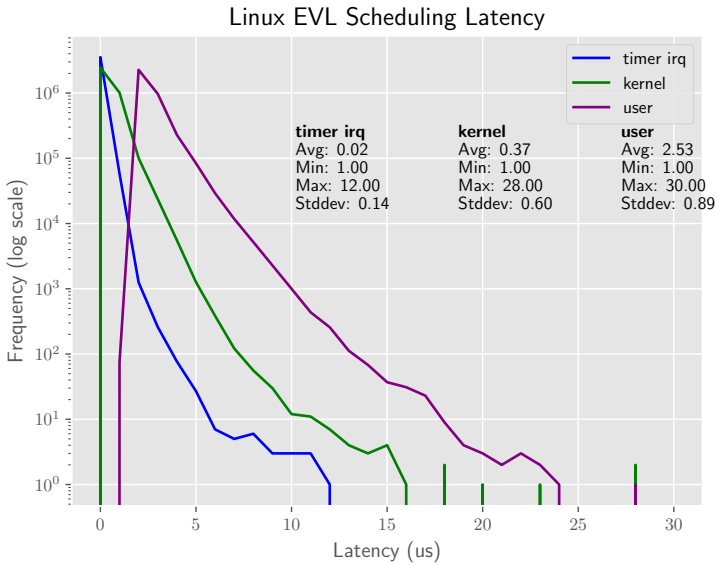
Scheduling latency is a crucial metric in the realm of real-time Linux [de Oliveira et al. (2020)]. It essentially represents the longest delay between the moment a high-priority thread becomes ready to execute its tasks, and when it actually begins executing its own code. This latency value plays a pivotal role in decision-making related to thread scheduling and allocation across processor cores. Additionally, it informs whether real-time threads should enter a sleep state or remain actively waiting after completing their present tasks.

To measure scheduling latency, we use a popular tool called `cyclictest`[27]. This tool evaluates the difference between a thread's intended wake-up time and its actual wake-up time, thereby providing critical insights about system latency performance.

[26] https://evlproject.org/core/user-api/
[27] https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start



(a) Histogram of scheduling latency using PREEMPT_RT



(b) Histogram of scheduling latency using EVL core

Figure 4.6 Scheduling Latencies of PREEMPT_RT and EVL core in a stressed environment. Scheduling latency of PREEMPT_RT compared with 'user' latency of EVL core.

There have been numerous studies indicating Xenomai's superiority over Linux in real-time applications [Brown and Martin (2010), Diana Marieska et al. (2011), Huang et al. (2015), Yang and Shinjo (2020), Jo and Choi (2022)]. However, such outcomes

largely depend on the specific setup and platform being utilized. In parallel, Linux has been progressively refining its real-time capabilities. We are not claiming `Xenomai` is inherently better than the real-time Linux variant, `PREEMPT_RT`. Although `PREEMPT_RT`, is continuously evolving, Xenomai has already demonstrated its effectiveness for applications requiring strict timing requirements. We employ Xenomai for the management of threads requiring strict timing, ensuring their operation remains reliable under worst-case conditions.

We conducted our testing on an *i5-7200U 4-core CPU running at 2.50GHz*, with a Linux kernel enabled for real-time operations, incorporating the `PREEMPT_RT` and `EVL_CORE` configurations. The tools used were `cyclictest` for Linux threads and `latmus`[28] for EVL threads. During these tests, the CPU was subjected to a 100% load using the `stress-ng`[29] tool. Our focus was on scheduling tasks on a single core, and we assigned a high priority (with the `SCHED_FIFO` policy -p 80 priority), with memory lock and real-time clock options. The duration of the test was 1 hour and results for both Linux and EVL threads is shown in Figure 4.6. Using the *latmus* tool, we measured the scheduling latency for timer irq, user thread and kernel thread. Meanwhile, cyclictest was used to create a user thread and measure its scheduling latency. The latency observed in Figure 4.6a can be compared with the user latency displayed in Figure 4.6b. However, we have chosen not to present the latency of `SCHED_OTHER` obtained by running cyclictest without the priority option. This is due to the latency being substantially high, exceeding 200us (maxing out at 8000us). The results, under a high-stress workload, showed that Xenomai had lower latency than `PREEMPT_RT`. This ensures more reliable scheduling of threads, which is crucial for maintaining the functional effectiveness of mobile robot under worst-case scenarios.

4.4 Jetson Development

Developing the proposed hardware and software architecture was an arduous task, particularly due to the scarcity of Jetson documentation at the time. We employed the Jetson AGX Xavier Developer Kit to construct the new architecture. In terms of the hardware architecture, we established communication with the motor drivers via the CAN peripherals using the CAN ports of the Jetson and CAN transceivers. The sensors of the mobile robot were interfaced using industrial I/O expandable shields[30] , facilitating the simple interaction with and manipulation of I/O sensors. Other robot

[28]https://evlproject.org/core/testing/
[29]https://wiki.ubuntu.com/Kernel/Reference/stress-ng
[30]Industrial-Shields

accessories can be connected directly to the Jetson PC's peripheral ports due to its inbuilt communication peripherals. We focused on the critical elements for industrial mobile robots, namely motor drivers and obstacle detection sensors, for interfacing with the PC. Other sensors, such as Lidar, IMU, and Camera, can be connected directly to the PC's serial USB ports.

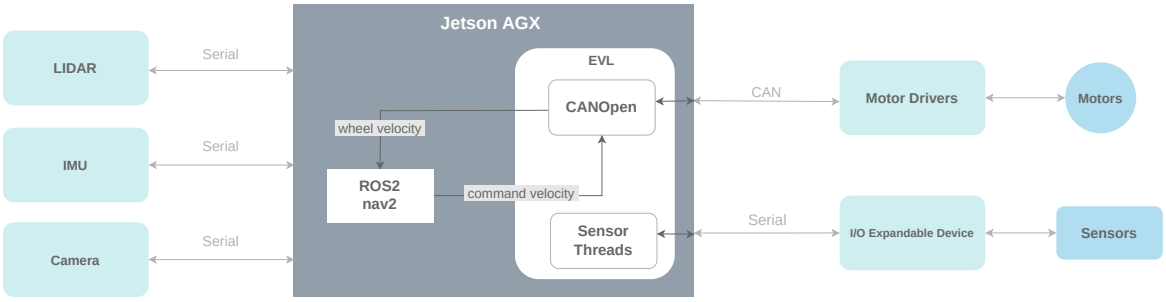


Figure 4.7 IMR hardware interface with Jetson AGX developer kit. With the software abstraction threads controlling the robot.

Building the proposed software architecture posed the next challenge. At the time of our work, the Jetson PC was released with Ubuntu 20.04 running Linux tegra 5.10. Jetson uses a specific Linux kernel for the tegra SoC, enabled with the `PREEMPT` configuration. Building and booting the Jetson with the `EVL` patched Linux kernel was an intricate task. To our knowledge, no other work has used the `EVL` patched Linux kernel on Jetson. After several attempts and a deep understanding of the Jetson device tree's components, we managed to install the `EVL` Linux kernel to boot on Jetson. However, this was a difficult process due to the ongoing development of the Jetson documentation. Upon successful installation of the kernel, we discovered that the Jetson drivers, including the CAN and others, were not functional because the `EVL` patched Linux kernel did not contain specific development for Jetson. Consequently, we worked on integrating the components of `linux-evl-v5.10`[31] into the `linux-tegra-5.10`[32] kernel, ensuring a complete kernel with `EVL` and `tegra` modules. This step allowed us to utilize the drivers of Jetson. Ultimately, we built and installed the Jetson with `linux-tegra-evl-5.10`. The guide detailing this installation process is presented in Appendix B.4. Figure 4.7 illustrates the complete architecture of the proposed industrial mobile robot. Our primary objective was to run peripheral component control threads with strict timing constraints under the `EVL` core. We utilize the CANopen library for motor communication, recognizing it as the standard library for interfacing

[31] https://source.denx.de/Xenomai/xenomai4/linux-evl.git
[32] https://github.com/OE4T/linux-tegra-5.10.git

with motor drivers. The CANopen and Sensors threads were established using POSIX threads, operating under the `SCHED_FIFO` policy, and subsequently attached to the EVL core. If enhanced reliability is required for additional component interfaces, the corresponding threads, designed to operate the specific libraries of these components, can be scheduled to run under the EVL core. However, we have not yet explored ROS2's real-time capabilities. As ROS2 manages all the robot's functional behavior, it is crucial to focus on its real-time characteristics to ensure that ROS2 also performs as expected. This focus is vital for maintaining system integrity and meeting industrial standards.

4.5 ROS2 real-time characteristics

We recall that the Robot Operating System, known as ROS, is not an actual operating system but rather a framework developed for creating robotics applications. Faced with limitations in the original ROS, particularly regarding operations in restrictive environments and maintaining real-time constraints, the developers decided to introduce an improved version: ROS2. Rather than developing a completely new middleware, they opted to utilize an existing one. After a thorough evaluation, DDS[33] emerged as the preferred choice, thanks to its long-standing history, strong support, and its capabilities in delivering real-time performance and safety certification.

4.5.1 Data Distribution Service

The DDS specification, defined by the Object Management Group (OMG)[34], forms the basis of a publish/subscribe data-distribution system. The implementation details are managed by the OMG, with various versions developed by different vendors, such as RTI[35] and eProsima[36]. DDS's versatility allows it to support a wide range of applications, from small embedded systems to large-scale infrastructure systems, including distributed real-time embedded systems. At the heart of DDS is a Data-Centric Publish-Subscribe (DCPS) model. This model creates a "global data space" that independent applications can access, facilitating efficient data distribution.

[33] https://docs.ros.org/en/humble/Installation/DDS-Implementations.html
[34] https://www.omg.org/omg-dds-portal/
[35] DDS-Implementation-Connext
[36] DDS-Implementation-Fast-DDS

In DDS, each process that publishes or subscribes to data is referred to as a participant, comparable to a node in ROS. Participants interact with the global data space using a typed interface. The DCPS model is comprised of various entities:

- *DomainParticipant*: This is the entry point to the service and a container for other entities. All applications within a *Domain* communicate with each other, promoting communication optimization and isolation.
- *Publisher*: Responsible for issuing data, a *Publisher* manages one or more *DataWriters* and sends data to one or more *Topics*.
- *Subscriber*: Responsible for receiving published data and making it available, a *Subscriber* acts on behalf of one or more *DataReaders*.
- *DataWriter*: A *DataWriter* is used by a *DomainParticipant* to publish data through a *Publisher*. It publishes data of a specific type.
- *DataReader*: Attached to a *Subscriber*, a *DataReader* allows a *DomainParticipant* to receive and access data. The type of data must match that of the *DataWriter*.
- *Topic*: Used to identify each data-object between a *DataWriter* and a *DataReader*, a *Topic* is defined by a name and a data type.

All DCPS entities come with a Quality of Service[37] (QoS) policy representing their data transport behavior. Each data transaction is configurable via various QoS policy options, like:

- *Deadline*: This policy mandates that a *DataWriter* and a *DataReader* must update their data at least once within a specific deadline period.
- *History*: This policy determines whether the data transport should deliver only the most recent value, all intermediate values, or a combination of both. The exact behavior is configurable via the depth option.
- *Reliability*: This policy dictates how data is transported. '*Best Effort*' mode ensures quick data transport but might risk data loss in less robust networks. '*Reliable*' mode guarantees data delivery through the retransmission of missed samples.

[37] QoS: The overall performance of a network. Includes factors such as bandwidth, throughput, availability, jitter, latency, and error rates.

- *Durability*: This policy makes the system attempt to save multiple samples for any potential late-joining *DataReader*. The number of samples saved is contingent on the History policy, with settings such as '*Volatile*' and '*Transient Local*'.

Real-Time Publish/Subscribe (RTPS) protocol is used for data transportation between a DataWriter and a DataReader. This DDS standard protocol enables DDS implementations from different vendors to interoperate. This protocol abstracts and optimizes transport, such as TCP/UDP/IP, and is flexible enough to take advantage of a QoS policy. With DDS, users can focus on their specific goals and determine how to meet real-time constraints easily. This is achieved by generating code as a DomainParticipant using the DDS APIs, which includes QoS policies, while the DCPS middleware handles data transport in distributed systems based on the specified QoS policy. Maruyama et al. (2016) carried out a performance evaluation of ROS 1 and ROS2, considering factors such as latency, throughput, thread count, and memory usage. Additionally, they examined the impact of changing DDS implementations and QoS policies in ROS2. Their findings suggest that DDS enhances ROS2's *fault tolerance* and *flexibility* across different platforms. However, despite ROS2's focus on improving real-time capabilities, particularly in network contexts, guaranteeing strict timing constraints remains a complex challenge.

4.5.2 ROS2 Scheduling Abstraction

From a logical perspective, ROS applications are composed of *nodes*, the smallest self-contained units of behavior. These nodes communicate using the *publish-subscribe* paradigm: *nodes* publish messages on *topics*, which broadcast the message to all nodes that are subscribed to the topic. Nodes react to incoming messages by activating *callbacks* to process each message. Since these callbacks may publish messages themselves, complex behavior can be implemented as a network of *topics* and *callbacks*. We recall the structure of ROS2, to define the important scheduling abstraction of ROS2. We find a series of layered abstractions, as depicted in Figure 4.8. It supports applications written in various languages, with official support for C++ and Python. Here is a simplified breakdown:

- *ROS Client Library* (rcl): rcl ensures uniformity of behavior across programs, regardless of the programming language used, by providing APIs.

- *ROS Middleware Library* (rmw): rmw acts as a communication bridge between rcl and the DDS. The implementation of rmw varies according to the DDS vendor.

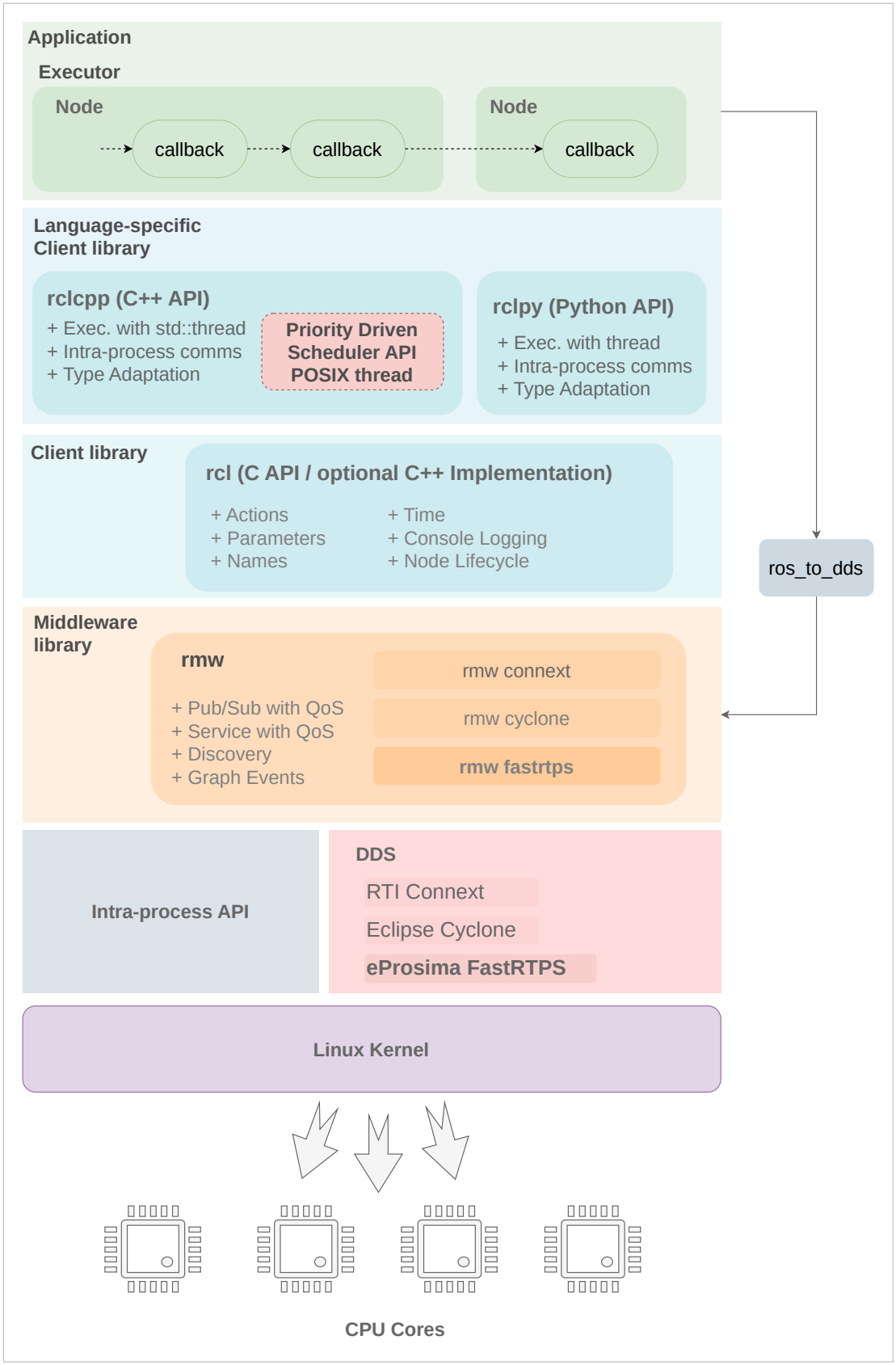


Figure 4.8 ROS2 structure with scheduling abstractions. With the highlighted priority driven scheduling package to schedule ROS2 executors with POSIX thread APIs.

Within ROS2, there are three essential abstractions related to scheduling [Choi et al. (2021)]: *callbacks*, *nodes*, and *executors*.

1. *Callbacks*: In ROS2, callbacks are the smallest units that can be scheduled. There are five types of callbacks: *timer*, *subscription*, *service*, *client*, and *waitable*. While *timer* callbacks operate based on a fixed schedule, the others are event-triggered. Callback functions in ROS2 primarily facilitate the transport of messages between publishers and subscribers.

2. *Nodes*: A node in ROS2 is a set of callback functions, arranged for feature partitioning and modularity. It is also the smallest entity that can be allocated to executors. This means that an executor is responsible for executing all callbacks within its allocated node.

3. *Executors*: Executors are responsible for managing the execution of callbacks. Essentially a thread operating on CPU cores, is responsible for executing assigned callbacks. The node abstraction is used to allocate callbacks to executors. Once nodes are assigned to an executor, it handles all callbacks from those nodes, irrespective of the callback's origins. Callback scheduling by an executor is distinct from traditional priority-based real-time task scheduling [Casini et al. (2019)]. In an executor's callback scheduling, two features stand out. Firstly, callback priorities are type-based, with timer callbacks always given the top priority. All callbacks are non-preemptively executed. Secondly, an executor updates non-timer callback readiness via communication with the middleware layer (rmw) at a polling point, leading to a round-robin-like operation of chains [Casini et al. (2019)].

When deploying a ROS2 application, it is essential to allocate individual nodes to hosts and then map them to the operating system's processes. Notably, ROS2 does not place any restrictions on this process. The implementation of the ROS2 execution model relies on running executors within the processes, which handle messages from rcl and subsequently trigger the related callbacks [Macenski et al. (2023)]. By initiating the `spin()` function of the executor instance, the current thread begins querying the *rcl* and middleware layers for incoming messages and other events. This continues until the node is deactivated. Currently, rclcpp offers three types of Executors: *Single Threaded Executor* (the most basic type), *Multi-Threaded Executor* (which allows for parallel processing by creating multiple threads), and *Static Single-Threaded Executor* (designed to optimize runtime costs when scanning a node's structure). Moreover,

ROS2 enables the arrangement of a node's callbacks into groups: Mutually Exclusive (where callbacks can not run simultaneously) and Reentrant (where callbacks can run in parallel). The current scheduling schematic of ROS2 executors is presented by Casini et al. (2019). *While the three executors provided by rclcpp are suitable for most applications, they fall short for real-time applications, which require definite execution times, predictability, and custom control over the execution sequence. This is attributed to a complex and mixed scheduling schematic, the risk of priority inversion, lack of explicit control over the callback execution sequence, and absence of built-in control for initiating specific topics.* Lastly, the overhead from the executor in terms of CPU and memory utilization is fairly significant [Macenski et al. (2023)].

From a real-time perspective, it is crucial to acknowledge that custom scheduling policies for executors. The scheduling methodology employed by the operating system for each ROS executor's threads significantly influences the overall timing behavior of the application. To achieve more predictable scheduling, executor threads can be associated with a reservation server, a strategic configuration aimed at ensuring predictability [Casini et al. (2019)]. The seminal work by Casini et al. (2019) served as the theoretical foundation for investigating the temporal behavior of ROS2. It made significant strides in predicting the end-to-end latency (or response time) of time-sensitive processing chains. They presented an intuitive model of ROS applications operating on a resource reservation scheduler such as Linux's `SCHED_DEADLINE`. Intriguingly, they validated their approach using the `move_base` package, a core component of the navigation stack for autonomous robots. Additionally, noteworthy work on scheduling schematics was performed by Choi et al. (2021). Their efforts focused on reducing end-to-end system latency through the use of the Priority-Driven Chain-Aware Scheduler (PiCAS). They established a modified rclcpp package that includes specialized APIs for assigning priorities to executors, which are then scheduled under the `SCHED_FIFO` policy of the `PREEMPT_RT` patched Linux kernel. When compared with traditional ROS2 scheduling that uses the `SCHED_OTHER` policy, and resource reservation which deploys the `SCHED_DEADLINE` policy, the improvement in end-to-end latency was substantial. This development enhances the predictability of ROS2 scheduling schematic, thus meeting the real-time application requirements within ROS2. However, there remains a gap in the research concerning the scheduling ROS2 applications under the Xenomai framework, an area with the potential to further improve system-level reliability and predictability.

4.5.3 ROS2 with Xenomai

There exists research focusing on real-time architecture for robotic control, particularly with respect to the Xenomai framework. This includes the work of Choi et al. (2009), which employed a Linux patched Xenomai 2.0 and a sink node; the research undertaken by Yang and Choi (2014) centered around trajectory planning; and the study conducted by Delgado et al. (2019), which utilized the Cross-Domain Datagram Protocol (XDDP). Furthermore, investigations such as those by Park et al. (2020) and Barut et al. (2021) have explored the scheduling latency for ROS2 under `PREEMPT_RT` Linux. However, these studies did not place particular emphasis on the scheduling schematics of ROS threads. Within the context of integrating the ROS2 and Xenomai-4 frameworks, no extant work is currently available. Our goal was to contribute by expanding the PiCAS [Choi et al. (2021)] rclcpp package's functionality, enabling it to schedule either a single executor thread or multiple threads using the EVL core. This modification provides a direct and user-friendly adaptation for developers, empowering them to develop and deploy real-time applications. By fostering system predictability through this method, we anticipate an improvement in the dependability of robotic devices.

We start by delineating the PiCAS framework, depicted in Figure 4.9. This framework adopts the workload models, namely Callback, Node, Executor, and Chain, originally conceptualized by Choi et al. (2021). Chains, which are not inherent to the ROS2 framework, are formed by system designers to encapsulate message exchanges between nodes. Developers assign priority to callbacks and executors based on their criticality to meet application-level requirements. Given the significant impact of end-to-end latency on real-time systems, PiCAS's focus is directed towards scheduling, resource allocation, and chain analysis. All proof-of-concept components, including priority assignment and end-to-end latency computation, are preserved as we primarily concentrate on scheduling enhancements within the Xenomai framework. To rectify the shortcomings of the original ROS2 framework, detailed in Section 4.5.2, PiCAS facilitates the prioritization of critical computation chains across the complex layers of ROS2, as represented in Figure 4.9. The existing ROS2 scheduling architecture is revamped with two considerations in mind: firstly, a higher-priority chain should execute before lower-priority chains; secondly, instances of the same chain assigned to the same CPU core should execute in their arrival order. This strategy aims to mitigate self-interference between instances of the same chain and consequently, prevent unwanted latency increases. Taking these factors into account, the author develops chain scheduling strategies within and across executors. To implement these strategies, PiCAS introduces a callback priority assignment scheme, which assigns

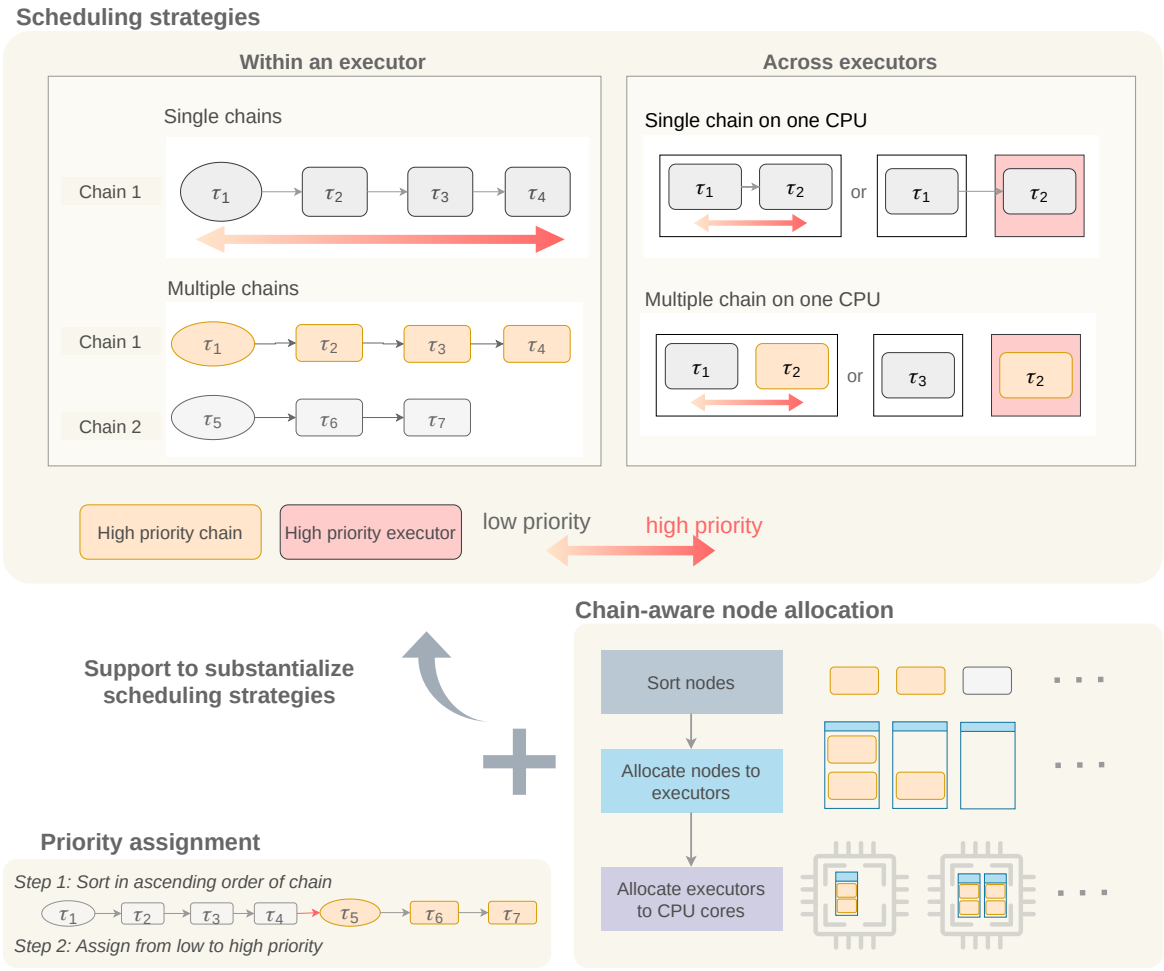


Figure 4.9 PiCAS framework [Choi et al. (2022)].

higher priority to callbacks of more critical chains. Additionally, it prioritizes callbacks at the beginning of a chain to mitigate self-interference. PiCAS also incorporates a chain-aware node allocation algorithm, which allocates nodes to executors and maps executors to available CPU cores in line with the scheduling strategies. This algorithm aims to allocate all nodes related to the same chain to a single CPU core whenever feasible, minimizing interference between different chains. Initially, PiCAS was implemented for a 'SingleThreaded' executor. However, Hoora et al. (2023) later enhanced it for 'Multithreaded' executors with both constrained and arbitrary deadlines.

We have advanced the functionality of the PiCAS framework, allowing it to operate with the EVL core. We developed an API , `attach_executor_evl_core()`, which accepts a attribute (true or false). It is essential to invoke this API to bind the executor to the EVL core, thus enabling the usage of EVL's scheduling capabilities. The PiCAS framework has been tested under three different conditions: ROS2 default, ROS2

Chain set	**Chains**
Chain 1	$\Gamma_1[\tau_i(\pi_i)] =: [\tau_1(30), \tau_2(31), \tau_3(32)]$
Chain 2	$\Gamma_2 =: [\tau_4(23), \tau_5(24), \tau_6(25), \tau_7(26), \tau_8(27), \tau_9(28), \tau_{10}(29)]$
Specifications [sec]	
Timer callback (τ_1 and τ_4)	$C_i = 0.109$, $T_i = 1$
Regular callbacks	$C_i = 0.131$

Table 4.1 Chain sets and their specifications.

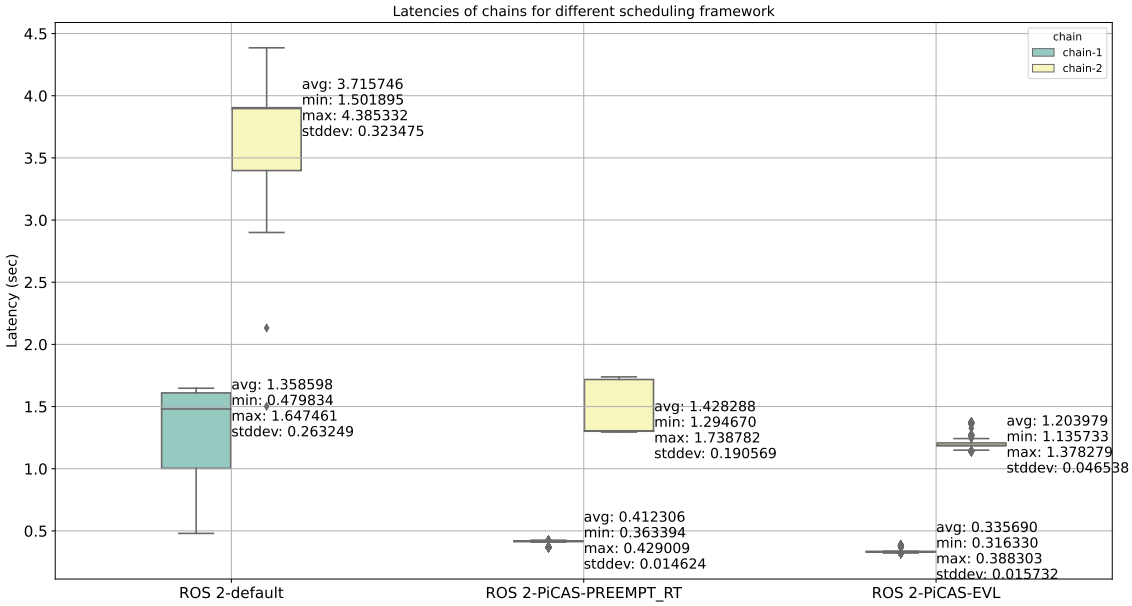


Figure 4.10 End-to-end latency results of ROS2 PiCAS running under EVL core compared with ROS2 default and PiCAS PREEMPT_RT.

PiCAS-PREEMPT_RT, and ROS2 PiCAS-EVL. The chain set utilized for this testing is identical to the one specified in Choi et al. (2021), as depicted in Table 4.1. Moreover, we have assigned priorities to the chains (π_i), using a predefined priority assignment rule. These tests were conducted on an *i5-7200U 4-core CPU, operating at 2.50GHz*, with a Linux kernel configured for real-time operations, incorporating both the `PREEMPT_RT` and `EVL_CORE` configurations. We utilized ROS2 distro Galactic[38], with the default DDS (FastRTPS). The PiCAS framework has also been implemented for the long-term distro Humble[39] and is compatible with other DDS options, such as Connext. We used `trace_picas`[40] to observe the end-to-end latency for the chains and depicted the results as a box plot in Figure 4.10. The results indicate that the PiCAS framework

[38]https://docs.ros.org/en/galactic/index.html
[39]https://docs.ros.org/en/humble/index.html
[40]trace_picas package is provided with PiCAS framework to measure the end-to-end latencies of chains

significantly reduces the chain latency in comparison to the ROS2 default scheduling scheme. It is also crucial to note that executing the executor at the system level is highly predictable and consistent with the PiCAS framework. Our further enhancement to employ the EVL core has significantly improved the end-to-end latencies. This test was performed under heavy workload conditions, and it was demonstrated that EVL scheduling further enhances the integrity of the PiCAS framework. This enhancement presents the possibility of scheduling ROS2 with Xenomai. Developers who require very low latency for their ROS2 applications can benefit from the straightforward adaptation of the EVL core.

4.6 Conclusion

In this chapter, we have provided a thorough guide that highlights the importance of real-time application and its use in IMRs. It is designed as a hands-on manual for developers who aim to implement a real-time framework to ensure predictable and robust performance for IMRs. We started by discussing the current issues related to how most systems do not focus on predictability and have limitations in maintenance and usability in an industrial setting. After a thorough evaluation of contemporary technologies and methodologies, we proposed a solid architecture that focuses on both the hardware and software aspects of the industrial robot. The hardware setup we suggest is simple and easy to maintain and integrate with, and it also simplifies programming and setup, making it more user-friendly.

We provided details on how to consider the software aspects to create and implement real-time features. We defined the system modules and scheduling classes that are used to ensure a high level of predictability. We discussed the use of both the `PREEMPT_RT` and `Xenomai` framework, which helps in bringing real-time features to Linux operating systems. We stressed the importance of the Xenomai framework for devices that require high levels of predictability and reliability. We also provided an easy-to-understand guide on how to create real-time applications using the EVL core. We provide the two new APIs that have been developed under EVL code that help define periods and schedule threads to run periodically. This is essential as the threads that control the hardware parts of an industrial robot should run periodically. We proved the importance of the Xenomai system by evaluating its scheduling latency, showing it provides a high level of predictability for threads running under the EVL core, even in worst-case conditions. We discussed the challenges faced during the development of the proposed architecture using the Jetson AGX platform. We provided details on

the kernels used and modified to run the EVL patched Linux kernel on the Jetson Developer kit. We highlighted the real-time features within ROS2, as it is where all the behavioural aspects of an IMR lies. We provided a brief overview of DDS, focusing on its real-time features using the QoS policy at the network level of the middleware. We acknowledged research that emphasizes the scheduling schematic of ROS2 executors to ensure reliability through real-time practices by using resource reservation and priority scheduling. We have extended the work of Choi et al. (2021) to run ROS2 applications under the EVL core to increase predictability in worst-case situations. This chapter provides a thorough guide for developers, covering both hardware and software aspects and providing a minimalist approach to practice. However, we did not discuss about energy issues. In the next chapters of this dissertation, we will explore the optimal energy-aware scheduler, namely ED-H, its requirements, and the implementation procedure into this software layer to meet the strict timing and energy requirements of the application.

Chapter 5

Real-time energy-aware scheduling under shared resource constraints

A lot of research studies addressed the scheduling issue of deadline constrained tasks in the RTEH (Real Time Energy Harvesting) context. In particular, in this thesis we focused on the optimal one, ED-H [Chetto (2014)]. However, optimality of this scheduler assumes that the tasks are independent i.e. they do not communicate and do not synchronize each other. The supplementary resource requirements of the tasks has not yet been thoroughly examined. As discussed in Section 2.3.2, the imperative role of shared resources in the system is clearly established. For instance, in a mobile robot, a task such as sensor fusion, where encoder readings and IMU data are fused to create odometry data, is crucial. This fusion necessitates a mutual exclusion mechanism to manage the data correctly, thereby emphasizing the importance of shared resources. Therefore, we propose an extension to the independent workload model that incorporates the critical resources. Consequently, we introduce new definitions and notations that will prove instrumental in subsequent sections. The main objective of that chapter is firstly to show how to implement jointly a resource access protocol with the ED-H scheduler and secondly to derive a new schedulability condition.

The plan of the chapter is as follows. In Section 5.1 we outline the fundamentals of shared resource and the terminologies used when a processing platform executes tasks with shared resources. Additionally, we present the resource access protocols designed to prevent priority inversion, emphasising on Dynamic Priority Ceiling Protocol (DPCP). In Section 5.2, we introduce a novel schedulability test, specifically designed for real-time tasks that share resources under the ED-H scheduling algorithm. This marks a novel contribution, being the first work to consider tasks with shared resources in the context of the state-of-the-art scheduling algorithm for RTEHS.

5.1 Shared Resources and Critical Sections

In this section, we revisit the key concepts that we first introduced in Section 2.3.2, focusing on *shared resource constraints*. *Resources* within a computer system can be broadly classified into two categories, *physical* and *virtual*. Physical resources refer

to tangible computer system components like memory, I/O, and hardware devices. On the other hand, virtual resources are abstract entities that maintain some level of internal consistency, such as shared data structures. The term '*critical sections*' is used to label the parts of a program that interact with *shared resources*.

Notions

Within the scope of a single-processor system, we present a set of resources, denoted as $\Omega = \{R_i \| 1 \leq i \leq \kappa\}$, which are typically serially reusable. These resources are allocated to jobs in a nonpreemptive manner, meaning that once a resource R_i is in use by a job, it remains inaccessible to other jobs until its release. The phenomenon known as a *race condition* occurs when the sequence in which a shared resource is accessed influences the final outcome. To mitigate this, a '*mutual exclusion*' mechanism is implemented, ensuring exclusive access to a critical section for a single job at any given time, thereby safeguarding the order and efficiency of operations. In this *mutual exclusion* mechanism, a job intending to utilize a resource R_i initiates a *lock* request, represented as $L(R_i)$. This request signals the job's requirement for the resource. The job resumes its operation once the requested resource is granted. Upon completion of its operation requiring the resource, the job releases it, performing an *unlock*, symbolized as $U(R_i)$. Each critical section, under the assumption of using only a single unit of the resource, is denoted as $[R; Rc]$. Here, Rc denotes the maximum execution time for that section.

5.1.1 Resource Conflicts and Blocking

In the context of shared resources, situations often arise where multiple jobs attempt to access these resources simultaneously during overlapping time intervals. Such circumstances are referred to as *resource conflicts*. A resource conflict occurs when two jobs require resources of the same type. Conflict escalates into *contention* when one job requests a resource that is currently allocated to another job.

As previously stated, mutual exclusion ensures that only one job accesses a critical section at any given time. If a job's lock request fails, it becomes *blocked* and is removed from the ready job queue. This job remains blocked until it is granted the required resource R_i, at which point it becomes *unblocked*, returns to the ready job queue, and resumes execution when scheduled. While '*blocking*' implies a mandatory delay imposed by other jobs due to the immediate unavailability of the resource. The principle of

mutual exclusion thus ensures orderly access to resources, preventing conflict and maintaining operational efficiency.

When multiple jobs share the same resources, certain challenges may arise. One of these complexities is *priority inversion*, which was defined in Section 2.3.2. *Deadlock* is another possible issue, occurring when jobs indefinitely wait for each other to release shared resources, resulting in a stalemate. To counteract these issues, scheduling theory offers various *resource access protocols*. These protocols establish rules determining the conditions for resource request approval, as well as the scheduling of jobs that require these resources.

5.1.2 Resource Access Protocols

We study the renowned uniprocessor resource synchronization protocols, designed to mitigate priority inversion and deadlock.

The first solution to counteract indirect priority inversion is the implementation of a strategy that disallows preemption when a job enters a critical section. One protocol, known as the *Non-Preemptive Protocol* (NPP), provides a straightforward approach to prevent priority inversion [Mok (1983)]. Under NPP, a job attempting to lock acquisition is given the highest scheduling priority, thereby safeguarding it from preemption by other jobs. This method effectively solves both priority inversion and deadlock problems. However, it creates a new issue where lower-priority jobs can obstruct unrelated higher-priority jobs, impacting their response time.

In response to this issue, Sha et al. (1990) proposed the *Priority Inheritance Protocol* (PIP). Under PIP, if a lock is free, a job locks it immediately. However, if the lock is occupied, the higher-priority job, is *blocked*. To counteract the priority inversion, higher-priority job transfers its current scheduling priority to the lock holder, low-priority job, until it releases the lock and higher-priority job can obtain it. Although PIP effectively prevents preemption by all lower-priority jobs, it has limitations. Deadlocks can occur when lock requests are nested. Additionally, PIP does not minimize the blocking time of higher-priority jobs since any locking request is granted immediately if the lock is free.

The *Priority Ceiling Protocol* (PCP) addresses both of these issues [Sha et al. (1990)]. PCP introduces *priority ceilings* and mandates that all jobs resource requests are known beforehand. The resource priority ceiling is the highest priority of all jobs that can obtain a specific resource. The system's *current priority ceiling* is the highest priority ceiling of the resources currently in use. When a job requests a resource, if the resource is not free, the job is *blocked*. Otherwise, the protocol follows certain

rules: if the job's current priority is higher than the current priority ceiling, the job immediately acquires the resource; if the task's priority is equal to the current priority ceiling and the task already holds another resource with the same resource priority ceiling, the task acquires the resource; otherwise, the task is *blocked*. Initially PCP was designed for fixed-priority scheduling algorithms, such as RM, this protocol has been extended by Chen and Lin (1990) to dynamic-priority scheduling algorithms, such as EDF. In this context, the priority ceiling is evaluated at each modification of the ready job list that is caused by activation or completion of jobs.

In Baker (1990) *Stack Resource Policy* (SRP) is a PCP variant that allows non-blocking tasks to share a single execution stack. The main difference between this protocol and PCP is that it prevents any preemption in locking scenarios that could be problematic, instead of denying lock attempts. Another PCP variant with similar analytical properties to SRP is the *Immediate Ceiling-Priority Protocol* (IPCP) [Burns (2009)]. The main idea here is to elevate a task's priority with each resource request to the resource's ceiling priority and lower the priority back when the resource is released.

In this dissertation, our primary focus is on the *Dynamic Priority Ceiling Protocol* (DPCP) as proposed by Chen and Lin (1990). In the context of energy-aware scheduling, particularly the ED-H algorithm, our intention is to scrutinize the synchronization issues when tasks operate with shared resources. More specifically, we aim to explore the impact on the schedulability test for ED-H in circumstances involving shared resource usage.

Notions

To delineate the DPCP protocol, we need to introduce the essential terminologies. Firstly, each resource within the protocol is associated with a unique parameter known as '*priority ceiling*'. For a resource R_i, its (dynamic) *priority ceiling*, denoted as $\rho(R_i)$, is equivalent to the highest priority among all jobs requiring R_i. It is important to note that the resources required by all jobs are identified in advance, before any job commences its execution.

Secondly, we must consider the system's '*current priority ceiling*'. This is a dynamic value represented as $\hat{\rho}(t)$, with t standing for the current time. If at time t, certain resources are being used, then $\hat{\rho}(t)$ matches the highest priority ceiling of the resources in use. Conversely, if all resources are free at time t, the ceiling $\hat{\rho}(t)$ defaults to a non-existent priority level π, which is theoretically lower than the lowest priority of all jobs. Additionally, the '*blocking set*' of a task is critical, representing the set of critical sections that may block jobs of the task. Specifically, it includes the critical sections

of lower-priority tasks whose priority ceilings are equal to or higher than the task's priority. This blocking set is denoted as Θ.

In systems with static priorities, the blocking set can be readily determined, given that task priorities and priority ceilings are static. However, in a system operating with a dynamic priority scheme, the blocking set's identification becomes less straightforward, as the relative priorities among tasks are subject to change from job to job.

5.1.3 Dynamic Priority Ceiling Protocol

The principles of Dynamic Priority Ceiling Protocol (DPCP) can be adapted from Priority Ceiling Protocol (PCP) in the context of Earliest Deadline First (EDF) scheduling and resource contention. The rules are thus reformulated as follows:

1. *Scheduling Rule*: EDF rule,
   (a) At the release time t, the initial priority $\pi(t)$ of every job J corresponds to its absolute deadline, with an earlier deadline indicating higher priority.
   (b) Every ready job J is preemptively scheduled in accordance with its current priority $\pi(t)$, which is based on its absolute deadline.
2. *Allocation Rule*: Whenever a job J requests a resource R at time t, one of the two scenarios arises:
   (a) If R is already held by another job, J's request fails and J becomes *blocked*.
   (b) If R is free:
      i. If J's priority $\pi(t)$ surpasses the current *priority ceiling* $\hat{\rho}(t)$, R is allocated to J.
      ii. If J's priority $\pi(t)$ is not higher than the system's current *ceiling* $\hat{\rho}(t)$, R is allocated to J only if J is the job currently holding the resource(s) with a *priority ceiling* equal to $\hat{\rho}(t)$. If not, J's request is denied, and J becomes *blocked*.
3. *Priority-Inheritance Rule*: If J becomes *blocked*, the job J_l that blocks J_h inherits the current priority $\pi(t)$ of J. J_l operates at its inherited priority until it releases all the resources with *priority ceilings* equal to or higher than $\pi(t)$; at that point, the priority of J_l reverts to its original priority $\pi_l(t')$, which it had at the time t' when it was allocated the resource(s).

These rules encapsulate the fundamentals of DPCP, where priority ceilings associated with resources are *dynamic*, varying in line with currently active jobs and their corresponding absolute deadlines. The resource is allocated contingent on both the initial and the inherited priorities (if applicable) under DPCP. The priority inheritance mechanism within DPCP assures that a job of lower priority that blocks a higher priority job will temporarily operate at the higher priority to circumvent *priority inversion* until the blocking is resolved.

For a job J_i using DPCP, the worst-case blocking length B_i is defined as the duration of the longest critical section in Θ_i. By factoring in the worst-case blocking of each task, a *sufficient* schedulability condition can be formulated, as depicted in the following theorem.

Theorem 4 (Chen and Lin (1990))**.** *A set of n periodic tasks can be scheduled by EDF using the DPCP protocol if the following inequality is met:*

$$\sum_{i=1}^{n} \frac{C_i + B_i}{T_i} \leq 1 \tag{5.1}$$

5.1.4 Example of EDF+DPCP

We illustrate the operation of EDF+DPCP through an example with a task set that comprises of three periodic tasks τ_1 , τ_2 and τ_3, and two semaphores S_1 and S_2 used to protect the access to shared resources R_1 and R_2 respectively. The tasks τ_i `=` $(C_i, T_i, D_i; [R_k; e_k])$ are described by the following parameters: τ_1 `= (2, 6, 5;` $[R_1; 1]$`)`, τ_2 `= (2, 8, 6;` $[R_2; 1]$`)` and τ_3 `= (8, 24, 20;` $[R_1; 3[R_2; 3]]$`)`.

Figure 5.1 shows how the periodic task set is scheduled by EDF using DPCP for managing shared resource accesses. For the tasks, the blocking sets are computed as follows: $\Theta_1 = \{S_1\}$ as τ_3 might lock R_1 when τ_1 intends to use it. Similarly, $\Theta_2 = \{S_2\}$ as τ_3 might lock R_2 when τ_2 intends to use it. For τ_3, no blocking is expected, and therefore $\Theta_3 = \{\}$. The worst-case blocking times that each task may encounter are determined from these blocking sets. For τ_1 and τ_2, the worst-case blocking times are $B_1 = B_2 = 3$, while $B_3 = 0$.

The execution commences with the highest priority job $J_{1,0}$ which starts its operation and accesses the free resource R_1. Subsequently, at $t = 2$, job $J_{2,0}$ embarks on its execution, accessing and completing its operation as the required resource R_2 is free. At $t = 4$, job $J_{3,0}$ being the only job remaining, initiates its execution and locks the available resource R_1. A new instance of job $J_{1,1}$ is introduced at time $t = 6$ with a higher priority ($\pi(6) = 12$) than $J_{3,0}$ preempting the latter to execute. However, at

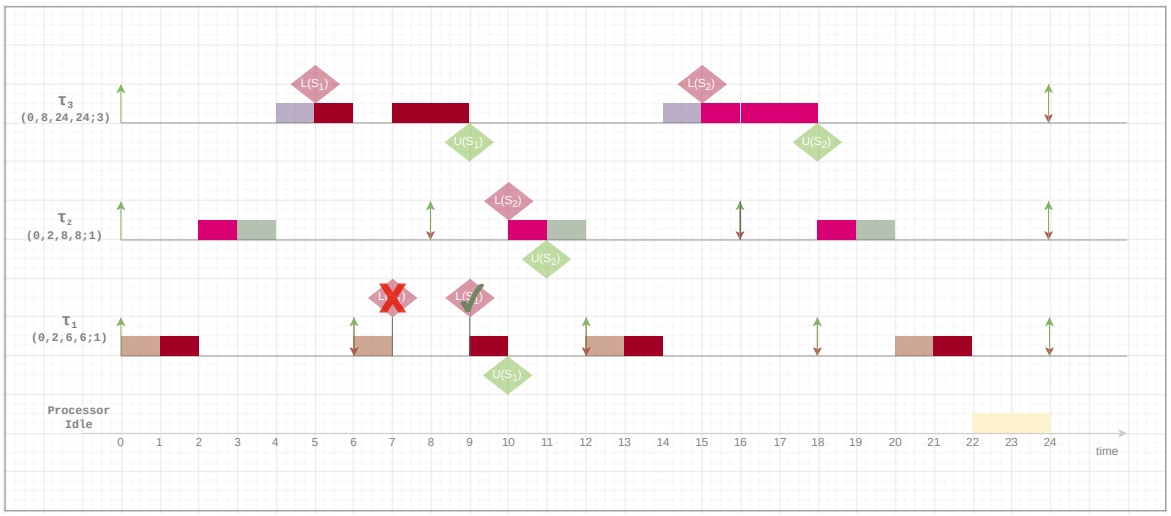


Figure 5.1 Illustrative example of EDF+DPCP schedule.

$t = 7$, when $J_{1,1}$ attempts to lock R_1, which is already secured by $J_{3,0}$ it cannot obtain the lock. This situation necessitates the verification of the priority ceiling, resulting in $J_{3,0}$ inheriting the priority of $J_{1,1}$ and resuming its execution to complete its access to R_1. Upon $J_{3,0}$ releasing R_1 at $t = 9$, $J_{1,1}$ acquires the lock on R_1 and completes its execution before the deadline. This example succinctly depicts the combined efficacy of EDF scheduling and DPCP in managing shared resources effectively and preventing deadline misses.

5.2 Energy-aware scheduling with Shared Resource constraints

Our examination is centered on a RTEHS with a set of periodic tasks, scheduled under the ED-H energy-aware scheduling algorithm. This analysis is primarily aimed at understanding the task and energy models pertinent to the RTEHS, as detailed in Section 3.4. Following this, the resource model will be discussed, merging previously established notions with their associated parameters, particularly those crucial to RTEHS.

Within the RTEHS, the process of scheduling real-time tasks that share resources under energy constraints presents an additional *blocking factor*. This typically occurs when a higher priority job requires a resource currently used by a job of lower priority. As depicted in Figure 5.2, the higher priority job encounters two forms of blocking, namely *blocking time* and *blocking energy*. Considering these blocking factors is critical as the higher priority job could face time or energy starvation while waiting for the resource. Therefore, our interest is to include these blocking aspects into the

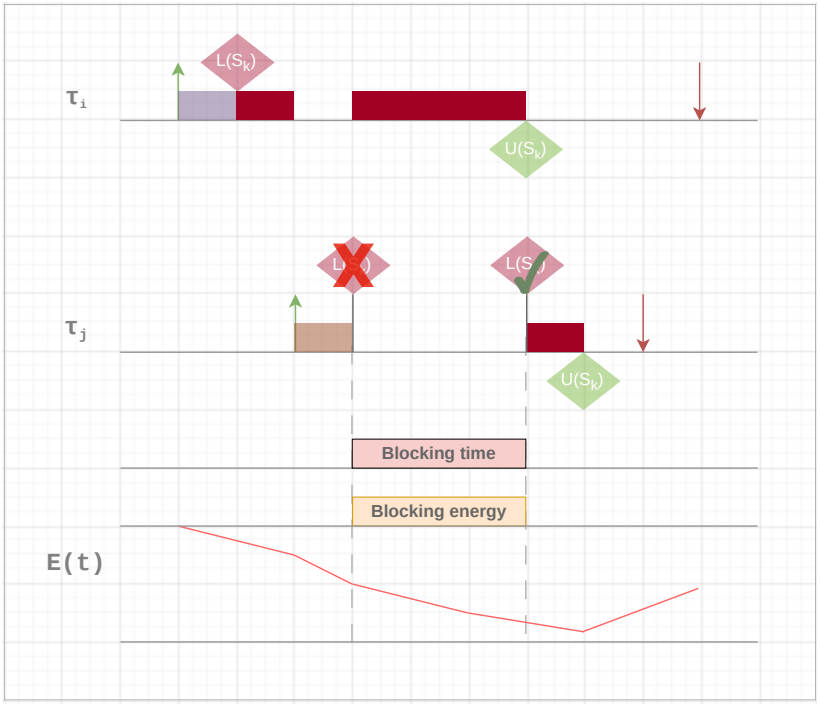


Figure 5.2 Blocking factors for tasks with shared resources under RTEHS.

schedulability analysis of ED-H. Doing so permits that a robust schedulability test for tasks with shared resources can be established.

5.2.1 Resource Model

We are considering a set of periodic tasks sharing a collection of κ resources, denoted as $\Omega = \{R_i \| 1 \leq i \leq \kappa\}$. The status of each resource is either locked or unlocked, depending on whether a job is currently accessing it or not.

Each resource R carries an associated dynamic *priority ceiling* at any given time t, notated as $\rho(R)$. This dynamic priority ceiling is defined as the highest priority among all tasks that could potentially require access to R. With the dynamic nature of job priorities in the system, these priority ceilings adjust accordingly. Within the DPCP, a job will temporarily take the priority of the highest priority job it is obstructing until the blockage is resolved. This inherited priority is denoted as $\pi_h(t)$. Conversely, if a job is not obstructing any others, its actual priority is indicated as $\pi_a(t)$.

Further parameters associated with RTEHS include $\delta_{l,h}$ and $\xi_{l,h}$. Here, $\delta_{l,h}$ represents the longest duration a lower-priority job J_l can hold a resource needed by a higher-priority job J_h. The term $\xi_{l,h}$ denotes the maximum energy consumed by the lowest-priority job J_l while holding the resource needed by the higher-priority job J_h.

5.2.2 Schedulability Analysis of ED-H + DPCP

In this section, we outline the approach to evaluate the feasibility of a set of periodic tasks in the context of shared resources. Initially, we introduce the schedulability of a periodic task set under ED-H, as described in Chetto and Queudet (2019). Theorem 5 presents the schedulability test, managing both the processor demand (h) and energy demand (g):

Theorem 5. *A set of deadline-constrained jobs J of a periodic taskset with $U_p \leq 1$ and $U_e \leq P_h$ is schedulable by ED-H iff for every interval $[t_1, t_2)$, $h(t_1, t_2) \leq t_2 - t_1$ and $g(t_1, t_2) \leq C + E_h(t_1, t_2)$*

Proof. Chetto and Queudet (2019) □

Before delving into the specifics, it is crucial to understand the necessity of incorporating the blocking factor in the schedulability test for ED-H. To illustrate this, we consider a periodic task set denoted as τ_i = $(C_i, T_i, D_i, E_i; [R_k; Rc_k, Re_k])$. The parameters are defined as follows : τ_1 = (4, 8, 8, 6; [R_1;2,3]), τ_2 = (2, 12, 12, 2; [R_2;1,1]) and τ_3 = (6, 24, 24, 18; [R_1;5,10]). Initially, the storage is considered to be at maximum capacity, $E(0) = E_b = 5$, and the instantaneous harvested power P_h, is a constant at 2.

By applying the existing schedulability test, the task set is deemed schedulable, in line with Theorem 5. We draw up a schedule for the task set, as shown in Theorem 5. At the start, the job with the earliest deadline, $J_{1,0}$ executes, followed by $J_{2,0}$. At $t = 6$, job $J_{3,0}$ begins its execution and acquires a lock on resource R_1. Subsequently, when job $J_{1,1}$ is released at $t = 8$ and preempts, it encounters the already locked resource R_1, which is in use by $J_{3,0}$. After verifying the priority ceiling, $J_{3,0}$ inherits the priority of $J_{1,1}$ and continues to use resource R_1. During its execution $J_{3,0}$ consumes energy that may be required by future jobs. However, by the time $J_{3,0}$ releases R_1, job $J_{1,1}$ lacks sufficient time and energy to complete its execution within its deadline. This results in an invalid schedule.

The schedulability tests, originally designed for independent periodic tasks under ED-H scheduling, can be broadened to include blocking terms. When blocking factors are present, the schedulability tests, which are both *necessary* and *sufficient* under preemptive scheduling, become merely *sufficient*. This is because blocking conditions, which are unique to each job, are assessed under worst-case scenarios and can never occur concurrently.

Our schedulability analysis is based on the processor/energy demand criterion. We adopt a similar strategy as proposed by Baruah (2006) to extend the demand

Figure 5.3 Non-valid ED-H schedule when considering tasks with shared resources.

criterion by introducing the concept of *blocking function* in scenarios involving blocking terms. However, in our context, we are dealing with both time and energy constraints. Consequently, it becomes essential to establish a blocking time function as well as a blocking energy function. These functions facilitate a detailed exploration of task set schedulability, considering it from a two-fold perspective – the time and energy domain.

5.2.3 Analysis in the time domain

The processor demand associated to a periodic job set J, $h(t_1,t_2)$ over the interval $[t_1,t_2)$ is the total processing time required by jobs with arrival times at or after t_1 and deadlines at or before t_2. We denote $B_T(L)$, as the function of longest blocking time for which a job with relative deadline less than or equal to L may be blocked by a job with relative deadline greater than L.

Lemma 1. *For any instants* t_1 *and* t_2,

$h(t_1,t_2) \leq \sum_{i=1}^{n} N_i(t_1,t_2).C_i + B_T(t_2-t_1)$

with $B_T(t_2-t_1) \stackrel{\text{def}}{=} \max\{\delta_{l,h} \parallel D_l > t_2-t_1 \text{ and } D_h \leq t_2-t_1\}$

and $N_i(t_1,t_2) \stackrel{\text{def}}{=} \max\left(0, \lfloor \frac{t_2-r_i-D_i}{T_i} \rfloor - \lceil \frac{t_1-r_i}{T_i} \rceil + 1\right)$

Proof. The number of jobs generated in $[t_1,t_2)$ corresponds to the number of non-negative integers k that satisfy the inequalities $t_1 \leq r_i + k\mathring{u}T_i$ and $t_2 \geq r_i + k\mathring{u}T_i + D_i$. There are exactly $\max\left(0, \lfloor \frac{t_2-r_i-D_i}{T_i} \rfloor - \lceil \frac{t_1-r_i}{T_i} \rceil + 1\right)$ such k's. Each job requires C_i processing time units coming from its normal execution with no resource access. Consequently, it has to receive at least $N_i(t_1,t_2)\mathring{u}C_i$ processing time units in $[t_1,t_2)$. However, under ED-H+DPCP, in time interval $[t_1,t_2)$ a job J_h with relative deadline less than or equal to t_2-t_1 of that wants to access a resource which is held by job J_l with relative deadline greater than t_2-t_1 may be blocked. It involves additional processing time bounded by $B_T(t_2-t_1)$. The lemma follows since the cumulative processing time required is upper bounded by $B_T(t_2-t_1) + \sum_{i=1}^{n} \max\left(0, \lfloor \frac{t_2-r_i-D_i}{T_i} \rfloor - \lceil \frac{t_1-r_i}{T_i} \rceil + 1\right).C_i$. □

5.2.4 Analysis in the energy domain

The energy demand $g(t_1,t_2)$ over $[t_1,t_2)$ is defined as cumulative energy required by jobs with arrival times at or after t_1 and deadlines at or before t_2. Let us denote $B_E(L)$ the highest quantity of energy from which a job with relative deadline less than or equal to L may be deprived of by a job with relative deadline greater than L.

Lemma 2. *For any instants t_1 and t_2,*
$g(t_1,t_2) \leq \sum_{i=1}^{n} N_i(t_1,t_2).E_i + B_E(t_2-t_1)$
with $B_E(t_2-t_1) \stackrel{\text{def}}{=} \max\{\xi_{l,h} \parallel D_l > t_2-t_1 \text{ and } D_h \leq t_2-t_1\}$

Proof. Each job requires at most E_i energy units coming from its normal execution with no resource access. Consequently, it has to receive at least $N_i(t_1,t_2)\mathring{u}E_i$ energy units in $[t_1,t_2)$. Under ED-H+DPCP, in time interval $[t_1,t_2)$ a job J_h with $D_h \leq t_2-t_1$ that wants to access a resource which is held by a job J_l with $D_l > t_2-t_1$ may suffer from lack of energy due to the largest amount of energy consumed by job J_l upper bounded by $B_E(t_2-t_1)$. The lemma follows since the cumulative energy required is upper bounded by $B_E(t_2-t_1) + \sum_{i=1}^{n} max\left(0, \lfloor \frac{t_2-r_i-D_i}{T_i} \rfloor - \lceil \frac{t_1-r_i}{T_i} \rceil + 1\right).E_i$. □

□

5.2.5 Sufficient schedulability test

We may derive the sufficient schedulability test of ED-H+DPCP for any periodic task set and any energy source profile :

Theorem 6. *A periodic task set is schedulable by ED-H+DPCP on any interval* $[t_1, t_2)$ *if:*

$$\sum_{i=1}^{n} N_i(t_1, t_2).C_i + B_T(t_2 - t_1) \leq t_2 - t_1 \tag{5.2}$$

$$\sum_{i=1}^{n} N_i(t_1, t_2).E_i + B_E(t_2 - t_1) \leq C + E_h(t_1, t_2) \tag{5.3}$$

Proof. Consider an independent task set in which each job issued from task τ_i has worst-case computation time equal to C_i and worst-case energy consumption equal to E_i. This task set will be schedulable by ED-H as long as Theorem 5 is satisfied. Under DPCP, directly from Lemma 1, the blocking time function can be treated as an additional processing time while from Lemma 2, blocking energy function can be treated as additional energy consumption of τ_i. Thus, it follows from Theorem 6 that the task set will be schedulable if equations 5.2 and 5.3 are satisfied. □

Theorem 6 above answered the question of whether a set of periodic tasks may respect all their deadlines between any two time instants, be given their computing, energy and resource requirements in one side and the characteristics of the energy storage and energy source in the other side. Theorem 6 gives a *sufficient* schedulability condition which deals with upper bounded worst-case processing time through inequality 5.2 and upper bounded worst-case energy consumption through inequality 5.3.

Illustrative example of ED-H+DPCP

We present an illustrative example that demonstrates the effectiveness of ED-H in scheduling tasks with shared resources under the DPCP protocol. We consider a periodic task set denoted by τ_i `=` $(C_i, T_i, D_i, E_i; [R_k; Rc_k, Re_k])$. The tasks parameters are defined as follows : τ_1 `= (3, 10, 10, 9; [`R_1;2,6`])`, τ_2 `= (4, 15, 15, 5; [`R_2;2,2.5`])` and τ_3 `= (9, 30, 30, 18; [`R_1;4,8[R_2;1,2]`])`. Initially, the storage is considered to be at maximum capacity, $E(0) = E_b = 6$, and the instantaneous harvested power P_h, is a constant at 2.

Initially, job $J_{1,0}$ begins execution, employing resource R_1. Subsequently, job $J_{2,0}$ uses resource R_2, which is free at the time. Job $J_{3,0}$ then starts its execution, securing a lock on resource R_1, which is free. Both the time and energy domain analyses are carried out for the time interval $[10, 20)$. Equation 5.2 is verified, with $(3+4) \leq 10$ being satisfied. Similarly, Equation 5.3 is verified, with $(9+8) \leq 26$ also being satisfied. Figure 5.4 depicts a valid schedule of the task set under the combined approach of ED-H and DPCP.

Figure 5.4 ED-H+DPCP valid schedule.

5.3 Conclusion

The schedulability test discussed previously in this dissertation did not apply when tasks share mutually exclusive resources. In this chapter, we have delved into a detailed schedulability analysis for the ED-H energy-aware scheduling algorithm that specifically focuses on tasks subject to *shared resource* constraints. Key elements such as *resource access protocols* and shared resource constraints were discussed, underscoring their role in preventing issues such as *priority inversion* and *deadlocks*, which can arise from *resource contention*. Moreover, we elaborated on an additional resource model specifically designed for RTEHS that schedules tasks sharing resources. A noteworthy contribution of this work is the proposition of a novel blocking function - the *blocking energy function*. This function augments the conventional blocking time function and is designed specifically for RTEHS. The illustrative example emphasized the shortcomings of the existing schedulability test, especially when dealing with dependent tasks that share resources. To rectify this, we offered a new theoretical analysis that incorporates the dual blocking factors - time and energy - into the processor demand and energy demand functions.

Furthermore, we established a new theorem for scheduling periodic tasks using shared resources with ED-H and managing resources with DPCP. While other dynamic resource access protocols could be applied, the schedulability test would remain consistent. The insights gained from this theoretical progression creates a solid basis for the real-world application of ED-H. Following chapters will shift focus towards the practical aspects, exploring the implementation of ED-H within the Xenomai Kernel.

Chapter 6

An Energy-Aware Scheduler for Xenomai

In this chapter, we extend our exploration of scheduling schemes within the real-time operating system Xenomai-4 (EVL), paying special attention to the energy needs of mobile robots at the operating system level of the computing platform. We have previously discussed an energy-aware scheduling scheme from a theoretical perspective. Now, we aim to further develop this concept into a more practical framework. In Section 6.1, we establish the requirements needed to transition this proof of concept to the operating system level. The primary requirement involves verifying the scheduler behavior through a real-time simulator. Section 6.2 provides a detailed overview of the newly developed REACTSim (Real-time Energy-Aware Computing Task scheduler Simulator) tool. It explains how it can be effectively used. We also demonstrate the distinctive outcomes when scheduling energy-constrained tasks using the traditional EDF scheduler versus the optimal energy-aware scheduler, specifically ED-H. Building on this foundation, we delve into the implementation of two new scheduling policies in Section 6.3. These policies include the traditional `SCHED_EDF` (Earliest Deadline First Scheduler) and the energy-aware `SCHED_EDH` (Earliest Deadline Harvest scheduler). We thoroughly examine key aspects of the EVL scheduler core, detailing its behavior and scheduling class. This section also offers extensive details about the implemented scheduling policies and their application, and outlines the development platform and debugging tool used during the process. Lastly, in Section 6.4, we consider the limitations of addressing the energy requirements of industrial mobile robots at the system level. To counter these challenges, we propose a novel strategy to refine our focus. This exploration sets a firm groundwork for continued research into energy-aware scheduling schemes for mobile robots.

6.1 Problem Statement

While energy-aware scheduling algorithms proposed by Moser et al. (2007) and Abdedda et al. (2014) have been innovative, their features have not been integrated into real-time operating systems (RTOS). Previous simulation studies by Abdallah (2014) and Rola El

Osta (2017) have critically examined and compared various metrics, including the rate of preemption, the processor usage, and the energy usage under ED-H [Chetto (2014)] with respect to other comparable scheduling algorithms. However, the primary focus of our work is on implementing the ED-H scheduling scheme within an RTOS, thereby extending its applicability to real-world applications. We have examined the dual-kernel approach offered by Xenomai for Linux distribution. The implementation of a scheduling scheme into a kernel is not straightforward; it requires deep understanding of the scheduling classes, a work that is further complicated by the inherent complexity of the Linux kernel's scheduling classes and an insufficient documentation. To circumvent these challenges, we have opted for Xenomai, given its unique functionality and the supportive infrastructure outlined in Section 4.3.2.

The EVL project[1] has been instrumental in simplifying the intricacies of Xenomai, enabling a more in-depth understanding of its core implementation. This has facilitated the integration of a novel scheduling scheme alongside pre-existing ones. However, implementing ED-H comes with its own challenges, especially given its reliance on computations like *slack time* and *preemption slack energy* to make decisions. To alleviate these challenges and before attempting to integrate this into the kernel framework, which could lead to additional debugging complexities, we propose to build a simulation framework. This will essentially emulate the kernel's scheduler functionality and provide a practice platform for our implementation. Consequently, we aim to design a simulation framework that is not only user-friendly but also facilitates the development of the scheduling scheme and enables the verification of its functionalities. Moreover, creating an offline schedulability test environment is essential. This environment will enable developers to define effective task parameters, thereby helping in verifying the schedulability of the task set. This tool will also enable the simulation and understanding of power consumption and power harvesting behaviors. Using this simulation framework, we can observe the behavior of realistic applications scheduled with an operating system. In summary, our goals are threefold. First, we will develop a simulation framework that emulates the kernel's structure and is easily usable. Second, we will create an offline schedulability test environment to assist developers in selecting appropriate task parameters and system parameters (e.g., energy storage size, etc). Finally, we will implement the ED-H scheduling policy into the Xenomai kernel and verify the efficacy of task scheduling with time and energy constraints.

[1] https://evlproject.org/

6.2 RT Simulator

Focusing on real-time task scheduling, numerous simulators have been developed, offering profound insights into the performance characteristics of scheduling algorithms. One such simulator is Cheddar [Singhoff and Plantec (2007)], an open-source platform designed to simulate real-time systems software architectures and assess their schedulability and performance metrics. Another simulator, RTsim [Manacero et al. (2001)], serves as a collection of programming libraries encapsulated in C++. Its primary role is to facilitate the simulation of real-time control systems. On the other hand, SimSo [Cheramy et al. (2014)] focuses on architectures with multiple processors working in real-time. It uses Python and a system called SimPy to allow for quick simulations and easy creation of scheduling policies. These tools have proven to be influential in the study and analysis of scheduling algorithms, particularly for their role in computing task scheduling. Another tool worthy of mention is the Linux Scheduler Simulator (LinSched)[2], which provides developers the flexibility to alter the behavior of the Linux scheduler and assess these changes in a user-space environment. LinSched's key strength lies in the fact that it necessitates minimal modifications in the kernel sources, allowing developers to create kernel code and have kernel-ready patches readily available post-testing. The Linux kernel also benefits from PRAcTISE (PeRformance Analysis and TestIng of real-time multicore SchEdulers)[3], which is a framework for the Linux kernel. It helps developers create, test, and debug scheduling algorithms. It also allows developers to compare different implementations by providing early performance estimates. This helps in choosing the best structures for data and schedulers. Unlike other tools like LinSched, PRAcTISE allows true parallelism which allows for a full test in a realistic scenario. These frameworks significantly ease the process of developing and modifying kernel code for the Linux kernel. In this work, we have developed a user-friendly simulator, REACTSim (Real-time Energy-Aware Computing Task scheduler Simulator). This simulator integrates a straightforward task structure identical to Xenomai and the Linux Kernel, while crucially incorporating energy constraints. REACTSim has a user-friendly Graphical User Interface (GUI) interface which simplifies its usage, making it an accessible tool for real-time energy-aware task scheduling.

[2]https://github.com/jserv/linsched
[3]https://github.com/Pippolo84/PRAcTISE

6.2.1 REACTSim

REACTSim is a tool developed in C for the analysis and scheduling of real-time computing tasks with energy requirements. In order to create an interactive and user-friendly environment, we have incorporated a GUI supported by QT[4]. The tool has been built with several scheduling policies including: Earliest Deadline First scheduler (`SCHED_EDF`), Earliest Deadline Harvest Scheduler (`SCHED_EDH`), Rate Monotonic Scheduler (`SCHED_RM`), and Deadline Monotonic Scheduler (`SCHED_DMS`). Our main focus during the development phase was on `SCHED_EDF` and `SCHED_EDH`, with the inclusion of energy constraints.

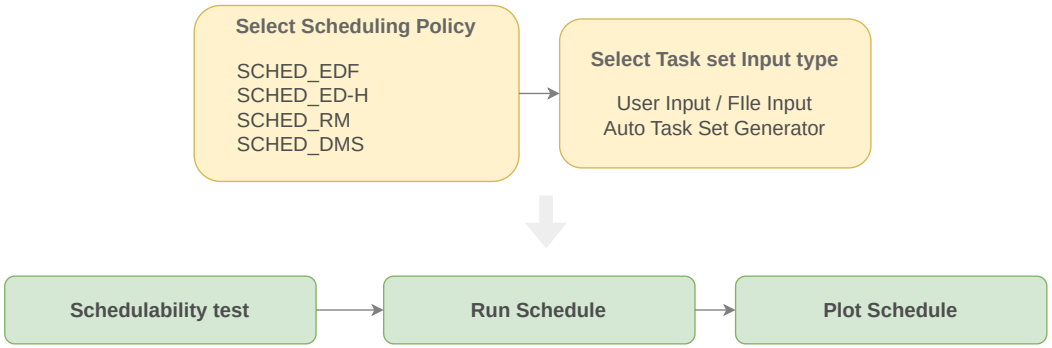


Figure 6.1 REACTSim Overview.

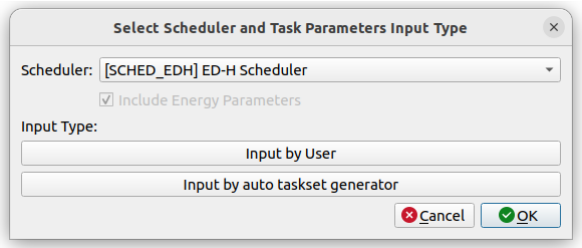
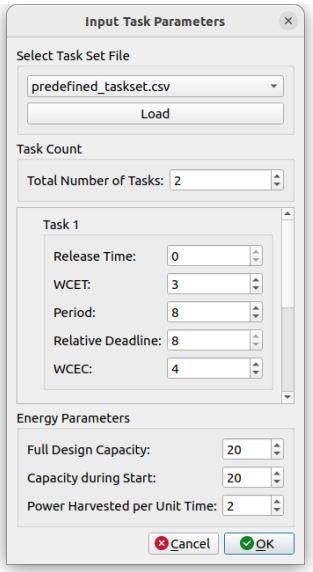


Figure 6.2 Interface to select the scheduling policy and task parameters input type.

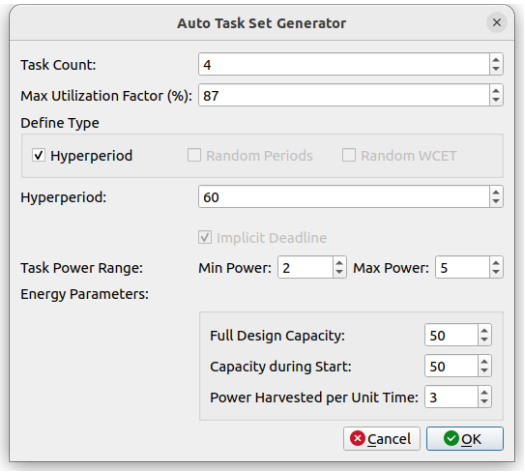
The overview of the REACTSim tool, as depicted in Figure 6.1, illustrates the selection of a scheduling policy as the initial step. The user then chooses the task parameter input type, conducts the schedulability test, and executes the schedule. Finally, the user can visualize the scheduling plot. As shown in Figure 6.2, REACTSim begins with the user selecting their desired scheduling policy from a drop-down menu. The option to 'Include energy parameter' can either be selected or deselected for the scheduling policy, except in the case of `SCHED_EDH` where it is enabled by default. We

[4]https://wiki.qt.io/Main

have incorporated two modes for task set input: user input and an automatic task set generator, which employs the UUniFast algorithm [Bini and Buttazzo (2005)].



(a) Input by User pop-up window



(b) Input by auto taskset generator pop-up window

Figure 6.3 Task parameter input pop-up windows.

Upon clicking the 'Input by User' button, a pop-up window opens. Figure 6.3a illustrates the layout for defining parameters for the 'n' number of tasks. If the 'Include energy parameter' option from the previous window is selected, energy parameters are included in this window. Otherwise, only time parameters are displayed. Additionally, users can load a predefined task set in a CSV file by selecting the desired filename and clicking the 'Load' button, which automatically loads the task parameters. Alternatively, if the 'Input by auto taskset generator' button is selected, a pop-up window titled 'Auto Task Set Generator' appears, as shown in Figure 6.3b. In this window, the user can specify the number of tasks to generate and the maximum utilization of the task set. Other task parameters can be defined by selecting one of three options under 'Define type': 'Hyperperiod', 'Random Periods' and 'Random WCET'. When 'Hyperperiod' is defined, the system generates task periods as factors of the given hyperperiod. With 'Random Periods', the user can define a min-max range, within which periods are randomly assigned to tasks, and then WCETs are calculated. Conversely, if 'Random WCETs' is chosen, the process operates in reverse. We recommend selecting the 'Hyperperiod' option, as other options could assign high period values to the tasks, or sometimes set the WCETs of the tasks equal to 1, given that the tool only supports integer values. If the energy parameter is enabled, users can set the maximum power

(power_max) for the task that could be consumed per unit of time, chosen randomly within the min and max power range. For simplicity, we consider the WCEC of the task to be the maximum energy that will be consumed by the task while computing, calculated as $power\_max \times WCET$. Similar to 'Input Task Parameter', users can define other energy-related parameters.

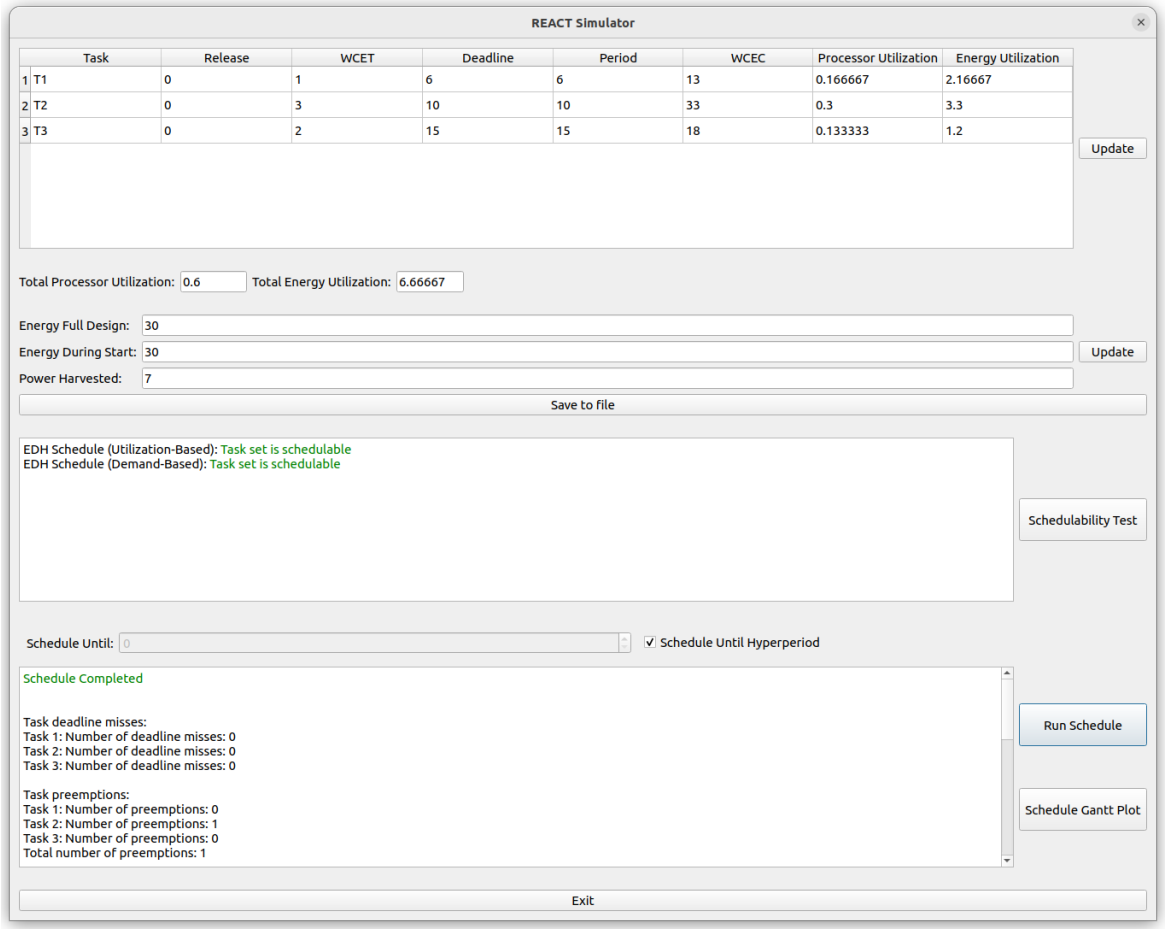


Figure 6.4 REACTSim Simulator Interface.

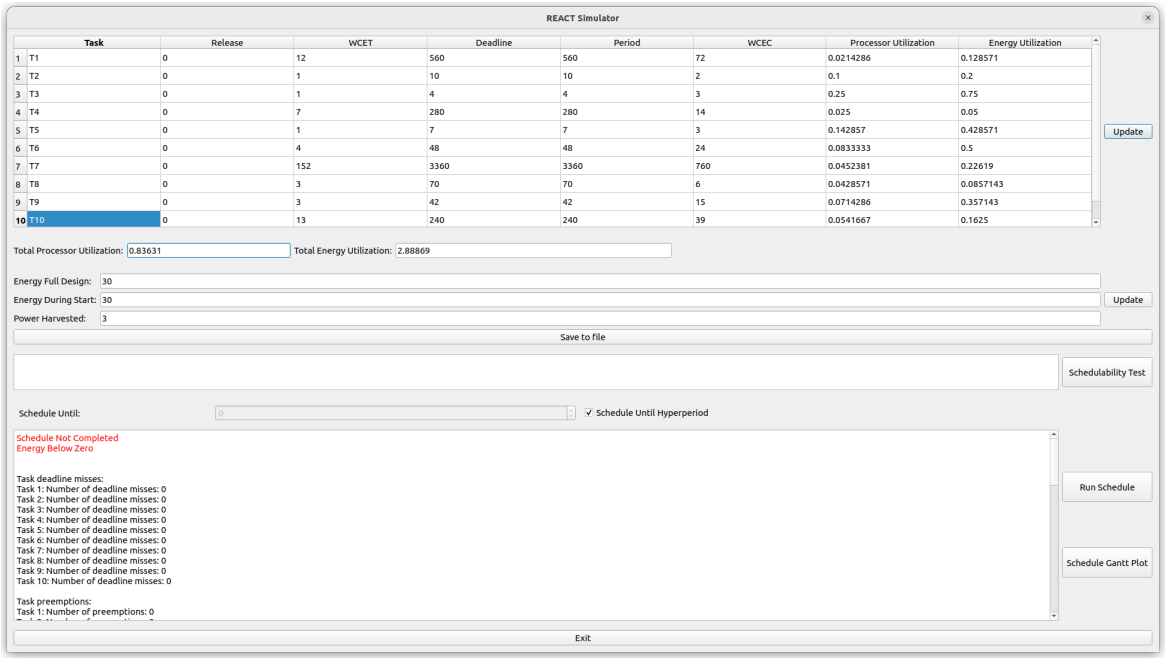
Once the task parameters are defined and the 'Ok' button is clicked, a subsequent dialog box appears with various layouts, as displayed in Figure 6.4. Initially, the task parameters defined in the previous step are visualized in a table. The user is permitted to modify the values in the 'WCET' and 'Period' columns. The updated parameters in the task structure are reflected when the 'Update' button is clicked. Below this table, the overall processor utilization and energy consumption of the defined task set are displayed. Users have the ability to update the energy parameters. To incorporate these new parameters, the 'Update' button on the side must be clicked. Additionally, there is an option to save the parameters to a file for easy retrieval in the

future. A schedulability test can be conducted on the defined task set, the results of which are subsequently displayed. Users can specify the time window for the schedule to be produced, with a maximum value of 1000, or schedule until the hyperperiod. Additionally metrics of the schedule such as deadline misses, preemptions, response times, and processor idle rates are also displayed. For a visual representation of the schedule, users can select the 'Schedule Gantt Plot' button, which results in the display of a schedule plot along with the corresponding energy level. The simulator is designed to function within a uniprocessor environment and is exclusively engineered to support synchronous periodic tasks with implicit deadlines. Moreover, all time-related and energy parameters are represented as integers. When the "Include Energy Parameters" is activated, and during the associated scheduling process, the energy in the storage incrementally increases corresponding to the power harvested. This happens at the beginning of each instance prior to the scheduling event. The tool operates under certain assumptions. It assumes that power harvested is constant, energy discharge only occurs during the execution of a task, and all tasks execute completely according to their WCET. These assumptions have been integrated into the simulator for simplification.

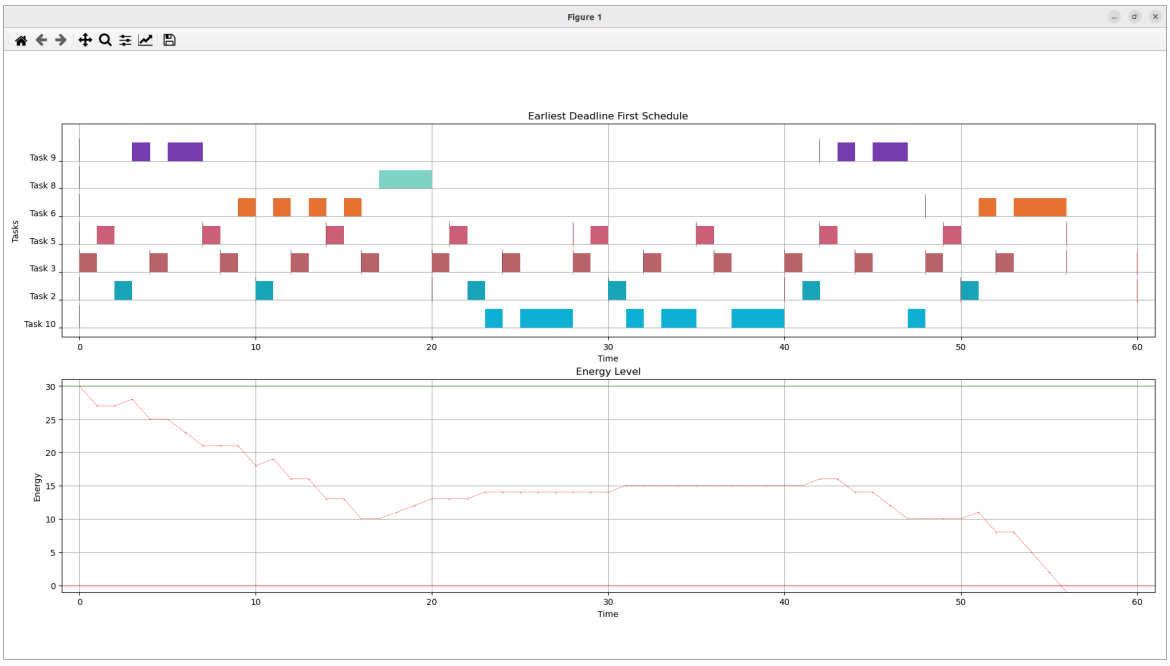
6.2.2 ED-H Simulation

The development of REACTSim was primarily motivated by the need to validate the `SCHED_EDH` scheduling policy ahead of its implementation in the Xenomai Kernel. The ED-H strategy relies on the computation of *slack time* and *preemption slack energy* to make decisions regarding the active and idle states of the processor. Furthermore, the tool is designed to determine the offline schedulability of a task set. The functional architecture of the tool has been deliberately designed to align with the kernel structure, simplifying the process of implementation.

In our demonstration, a periodic task set consisting of 10 tasks was configured with a hyperperiod of 3360. The parameters assigned to this task set are shown in the scheduling interface window as illustrated in Figure 6.5a. Initially, we applied the `SCHED_EDF` policy to execute the schedule. We chose not to perform a schedulability test as the tasks incorporated energy parameters. Upon running the schedule, it is obvious that the schedule cannot be completed as the energy level falls below zero, which is clearly highlighted in the display layout of the interface window. The energy level dropping below zero can be clearly observed in the energy level plot depicted in Figure 6.5b. The non-optimality of the EDF scheduler for the real-time energy harvesting model was previously proven by Chetto and Queudet (2014a). Consistently, our simulation results are in line with this conclusion.



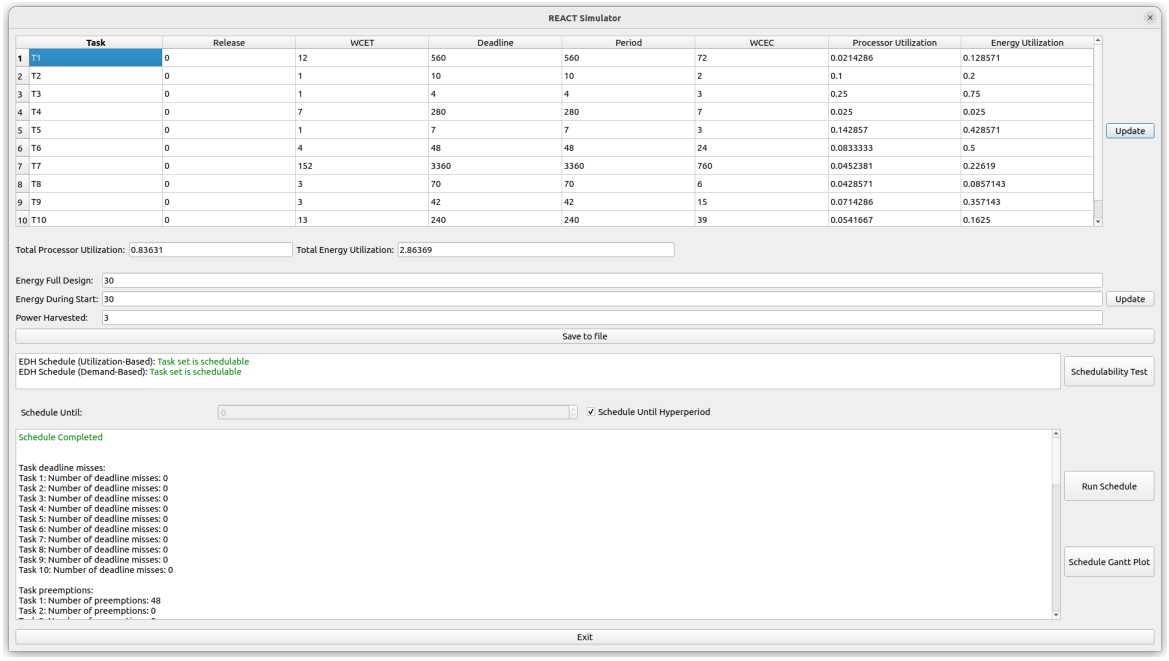
(a) Simulator Interface - EDF Scheduler.



(b) Tasks schedule and energy level.

Figure 6.5 EDF scheduling with energy simulation.

We then scheduled the same task set of 10 periodic tasks using the ED-H scheduling scheme. As seen in the interface window Figure 6.6a, the task set is both time and energy feasible. When we execute the schedule, it completes successfully with no missed



(a) Simulator Interface - ED-H.

(b) Tasks schedule using ED-H and energy level.

Figure 6.6 ED-H scheduling simulation.

deadlines. Figure 6.6b presents the schedule of the tasks, along with the corresponding energy level. It's evident that the energy level never falls below the threshold, and is fully replenished at the end of the hyperperiod. From these results, we can deduce that

the ED-H scheduler's behavior is correct and ready for implementation into the kernel framework. For practical simplicity, we opted not to define any specific units, enabling the time and energy parameters to correlate directly to their respective units.

6.3 Novel Scheduling Policies in Xenomai

Scheduling policies determine when and how tasks are prioritized and selected to run. This fundamental aspect of operating system design is illustrated in Figure 4.5, where traditional scheduling policies implemented in Xenomai-4 (EVL) are presented. Historically, the Linux Kernel process scheduling[5] employed an `O(n)`[6] scheduler from versions 2.4 to 2.6. In Linux Kernel 2.6, an `O(1)`[7] scheduler was introduced, signifying a considerable advancement. This scheduler, alternatively referred to as the `Big O` of `1` scheduler or the constant time scheduler, marked a shift in task management efficiency. In a subsequent transformation, Ingo Molnar introduced a novel scheduler structure in Linux, superseding the `O(1)` scheduler [Lelli (2014)]. The novel scheduler had a logarithmic complexity (`O(log N)`), as opposed to the constant time complexity (`O(1)`) of the older version. This added complexity was a result of incorporating a data structure called a *red-black tree*[8] for managing thread execution.

In contrast, the EVL kernel uses a custom form of *priority queue* implemented by Philippe Gerum. This implementation employs *linked lists*[9] to arrange thread elements. The `__list_add_pri` macro ensures that elements are inserted into the list based on their priority, creating an `O(n)` complexity for insertion. The `list_get_entry` macro, on the other hand, retrieves and removes the first item in the list, resulting in an `O(1)` complexity for removal/access. The EVL scheduling process comprises a core block, which provides fundamental functionalities, and a set of scheduling classes. Each class encapsulates one specific scheduling policy except the *fifo* class. In the EVL scheduling class, we introduce two new scheduling policies. The first, `SCHED_EDF`, is the

[5] https://www.scaler.com/topics/operating-system/process-scheduling/

[6] O(n) scheduler, divided processor time into "epochs", adjusted for incomplete thread, and enabled simultaneous scheduling of n-processes, enhancing the efficiency over the previous circular queue scheduler.

[7] O(1) scheduler, or constant time scheduler, schedules processes within a consistent timeframe, regardless of the number of processes. It uses two queues: a run queue for active processes and an expired queue for processes with expired time allotments, favoring interactive tasks and lowering non-interactive tasks' priorities.

[8] Red-Black Tree, a self-balancing binary search tree, for scheduling. Each task is stored in this tree based on its virtual runtime, with the leftmost node, indicating the least virtual runtime, being selected next for execution.

[9] linux-evl-list

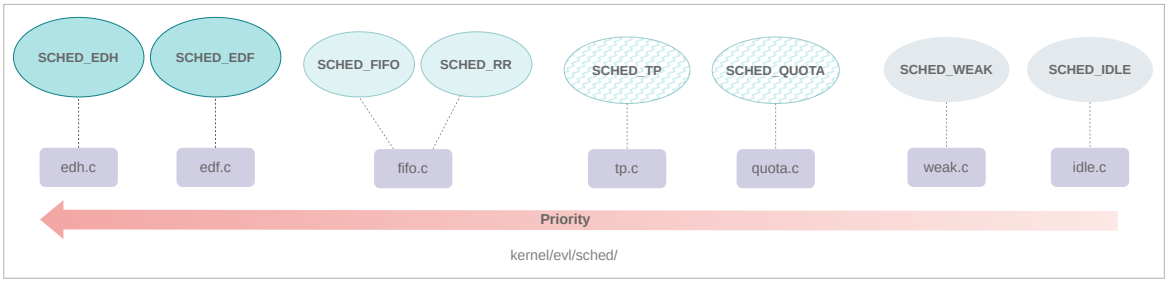


Figure 6.7 Xenomai-4 EVL Kernel scheduling policies, highlighting the newly implemented policies `SCHED_EDH` and `SCHED_EDF`. `SCHED_EDH` being the highest priority scheduler among the other policies.

traditional earliest deadline first scheduler. The second is a novel scheduler, `SCHED_EDH`, an optimal energy-aware scheduling scheme proposed by Chetto (2014). Figure 6.7 displays these novel scheduling policies within the hierarchy of the schedulers, and the respective files where the scheduler exists in the kernel codebase. Before delving into the specific details of these scheduling policies, it is crucial to describe the integral components of the EVL scheduler core and the entities within the scheduling classes.

EVL Scheduler Core

The integration of the *dovetail* framework merges the real-time and regular capabilities of the Linux kernel into a unified control flow. This integration enables a form of dual kernel setup, where you can switch execution context between the in-band (regular kernel) and out-of-band (EVL real-time) stages as needed. The Dovetail mechanism introduces a two-stage interrupt pipeline as depicted in Figure 6.8, which prioritizes immediate processing of device interrupts (IRQs) in an "out-of-band" stage, while the regular kernel operations ("in-band" stage) are momentarily delayed, thus ensuring prompt responses to external events. This process enables an autonomous core to efficiently manage real-time applications, while maintaining regular kernel activities with minimal interruptions. The function `dovetail_context_switch()` facilitates this switch. It inserts a preemption window for out-of-band IRQs just before the context switch is finalized, allowing for the possibility of interrupting partially switched in-band contexts. In the `context_switch()` function, if dovetail is enabled, it creates a short window for preemption by out-of-band interrupts just before finalizing the context switch. This is done by first setting the `active_mm`[11] of the previous task to `next->mm`

[9] https://evlproject.org/dovetail/pipeline/

[11] Associated `mm_struct` structure that holds memory management information related to the process.

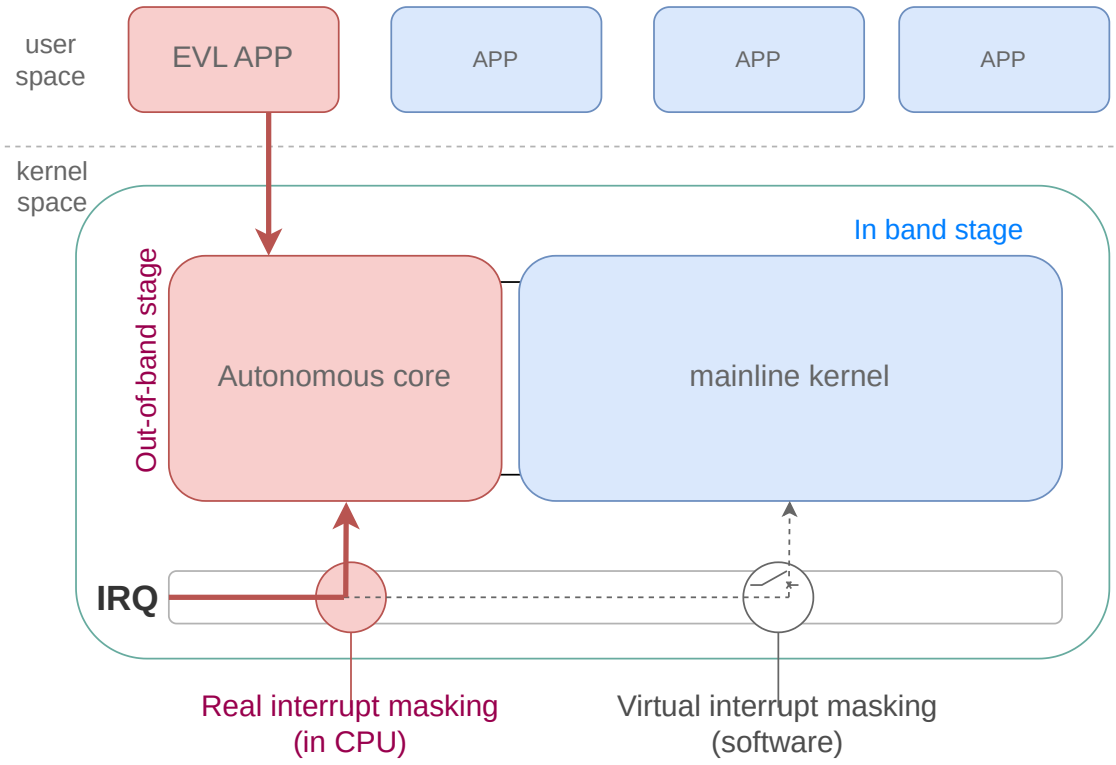


Figure 6.8 Dual-kernel interrupt pipeline[10].

and then calling `hard_local_irq_sync()`. After these calls, the `active_mm` of the previous task is set back to the old `mm` structure. This window allows for the possibility of interrupting the context switch process. This ensures that high priority real-time tasks (out-of-band) can preempt lower priority tasks even if they are in the process of switching context. Thus, the framework offers lower latency response times to such out-of-band interrupts and tasks.

At the end of the function, `inband_switch_tail()` is called. If the next task is on its way to the out-of-band stage, the *context switch epilogue*[12] is delayed. The context switch epilogue is executed only when the task switches back to the in-band stage, making sure that the context switch is entirely completed before allowing the next task to run in the in-band stage. The function `finish_task_switch()` is then called to complete the rest of the context switch process which includes housekeeping tasks such

[12]The "context switch epilogue" is a phase that finalizes the context switch process. It involves cleaning up and restoring the state of the system after the execution has been transferred from one process or thread to another.

as decrementing the reference count of the `mm_struct` of the previous task, performing scheduler housekeeping, and then finally enabling preemption.

In addition to the context switch mechanism, dovetail plays an integral role in Xenomai's real-time scheduling system. The Xenomai-4 (EVL) real-time subsystem introduces its own scheduler, managing and scheduling threads in the system according to their scheduling classes and maintaining a *runqueue* of active threads for each CPU. Every *runqueue* is protected by a spin-lock to guarantee correctness on concurrent updates. Runqueues are modular, in the sense that there is a separate sub-runqueue for each scheduling class. Tasks are *enqueued* on a runqueue when they wake up and are *dequeued* when they are picked to run or suspended. The core function of this system, `__evl_schedule()`, is responsible for selecting the next thread to run and performing the context switch to start its execution. This function begins by confirming that it is not running in the in-band context with enabled hardware interrupts which could disrupt the real-time operations of the system. It then disables the local IRQs for the duration of the scheduling operation. If the current thread has user-level state (`EVL_T_USER`), *priority ceiling* requests are committed before putting the thread to sleep. The `pick_next_thread()` function is then used to select the next thread to run, considering all threads, blocking conditions, preemption requirements, and scheduling classes in decreasing weight order. The first runnable thread found with the highest priority is selected from the highest scheduler class. After selection, `set_next_running()` is called to prepare the thread for execution. Finally, a context switch to the selected thread is performed by preparing the run queue switch with `prepare_rq_switch()` and performing the actual context switch with `finish_rq_switch()`. The Xenomai scheduling system is a highly modular, preemptive, and class-based scheduling system, where each scheduling class is handled separately, and the highest priority thread is always run next. It follows strict locking protocols and carefully maintains the states of threads and *runqueues* to ensure safe and correct scheduling.

Scheduling Class

The `evl_sched_class` structure describes a scheduling class in the Xenomai EVL real-time subsystem. Here are some of the significant members of this structure:

- `sched_init`: This function *initializes* a runqueue.
- `sched_enqueue`: This function is used to *enqueue* a thread in the runqueue of the respective scheduling class.

- `sched_dequeue`: This function is used to *dequeue* a thread from the runqueue of the respective scheduling class.
- `sched_pick`: This function is used to pick the *next thread* to run from the runqueue. This is where the scheduling class's scheduling policy would come into play to determine which thread should be run next.
- `sched_setparam`: This function is used to set the scheduling parameters for a thread. The scheduling parameters typically include the priority of the thread.
- `sched_getparam`: This function is used to retrieve the scheduling parameters for a thread.
- `nthreads`: This is the count of threads in this scheduling class.
- `weight`: The weight of this scheduling class. It could be used in schedulers which consider the weight of tasks for making scheduling decisions.
- `policy`: The scheduling policy of this class. This is typically a constant that represents the scheduling policy.
- `name`: The name of this scheduling class.

In addition to the scheduler class structure, we have the `evl_thread` structure, which contains instances of the scheduling class entities. The ioctl methods (specifically `thread_ioctl` and `thread_oob_ioctl`) serve as the interface for *userspace* to configure and control thread behavior, including scheduling, state, mode, and interaction with other system components. Notably, `thread_oob_ioctl` provides out-of-band control operations, enabling the system to switch execution modes and signal events among threads, enhancing the real-time capabilities of the system. These operations are critical for managing threads and their scheduling characteristics within a running system.

6.3.1 Development and Debugging Platform

Several tools exist to help kernel developers during development and debugging. An example of debugging tool is *perf* [De and Arnaldo (2010)] which can be employed for collecting performance data and extracting execution traces from running operating system. There is also an infrastructure, *trace/ftrace*[13], that is beneficial for debugging

[13]https://www.kernel.org/doc/html/v5.0/trace/ftrace.html

as well as scrutinizing latencies and performance issues within the kernel. Further, Kernelshark[14] serves as a graphical interface for processing *ftrace* reports, proving to be an effective tool for understanding kernel behavior.

Typically, once kernel modifications are made, the next step is to run the updated kernel in a virtualized environment. In this context, the *Kernel Virtual Machine* (KVM) can be highly beneficial. It can be controlled and attached by the GNU Project Debugger (GDB), facilitating the debugging process. However, it should be noted that the effectiveness of this solution can be limited in scenarios involving high concurrency [Lelli (2014)]. It may also potentially impact the reproducibility of certain bugs. Furthermore, we identified the use of KGDB, in the context of halting EVL code, has its limitations, which make it unsuitable. In our work, we have utilized QEMU[15], a hardware virtualization software, to emulate a system with defined specifications. To provide a filesystem for the emulated system, we employed minimal configured Buildroot[16]. For debugging purposes, we employed the `printk()` function. The kernel utilized for our development was linux-evl-v5.15[17].

6.3.2 `SCHED_EDF` Xenomai scheduler policy

We describe the implementation of the `SCHED_EDF` scheduling policy and evaluate its key property: assigning the highest priority to the thread with the earliest absolute deadline. Our implementation strategy closely follows the approach adopted for the EVL `SCHED_FIFO` scheduler. This strategy is commonly referred to as the distributed run-queue [Lelli (2014)]. This approach implies that each CPU possesses its own private data structure that executes its unique ready queue. Therefore, each CPU's tasks are stored in a distinct run-queue specific to that CPU. This queue is designed as a linked list that orders tasks by their (absolute) deadlines. We designed the `evl_sched_edf` scheduling class structure with a weight attribute set to `5`. This value is higher than that of the `evl_sched_fifo` class, thus conferring a higher priority to `evl_sched_edf`. We register the `evl_sched_edf` as the top most class which enables the `__pick_next_thread()` function to pick thread from the `evl_sched_edf` class first. Furthermore, we added two extra members, `sched_periodic` and `sched_wait`, to the scheduling class. These members are specifically designed to manage the periodic timer handler and the wait timer event, respectively.

[14]https://kernelshark.org/
[15]https://www.qemu.org/
[16]https://buildroot.org/
[17]Xenomai-linux-evl_v5.15

EVL EDF thread entities, as presented in Listing 6.1, have been integrated into the `evl_thread` structure. Whenever the thread is activated, the absolute deadline of the job is updated. This update is signaled by the newly incorporated `EVL_T_UPDATE` thread info flag and occurs within the `sched_enqueue` member of the `evl_sched_edf` class.

```
struct evl_edf_entity {
  struct list_head rq_next;
  /*thread params*/
  ktime_t release_time; /* Thread first arrival time */
  ktime_t rel_deadline; /* relative deadline */
  ktime_t period;     /* period */
  //calculated absolute deadline for job instance
  ktime_t job_release_time; /* instance release time */
  ktime_t abs_deadline;     /* absolute deadline */
  u64 job_instance;     /* number of job instance */
  unsigned int new_thread; /* flag for new edf thread */
  /*Thread timers*/
  struct evl_timer edf_timer;
  struct evl_wait_queue wq;
  struct evl_poll_head ph;
  bool ticked;
};
```

Listing 6.1 EVL EDF thread entities.

We have expanded the `evl_sched_attrs` structure by adding EDF scheduling parameters. These parameters can be user-defined and are transferred to the kernel space using libevl functions. Listing 6.2 provides an example of a simple application in which the user can create a thread that is scheduled under the `SCHED_EDF` policy within the EVL core. Additionally, the `evl_thread_stats` structure has been created that enables the deadline monitoring feature incorporated within the `sched_wait` function. This modification enables users to retrieve this value through the `evl_thread_wait_period` function.

```
void *Application(void *parm)
{
  int efd, ret = 0;
  struct evl_sched_attrs attrs;   /* evl scheduling attributes */
  struct evl_thread_stats stats;  /* thread statistics */

 /* Set the thread scheduling policy */
  attrs.sched_policy = SCHED_EDF;
```

```
/* Put 0 to create synchronous threads */
 attrs.edf.release_time = 0;
/* Period of the thread in ns (nanosecond) */
 attrs.edf.period = 2000000000;
/* Relative deadline of the thread in ns */
 attrs.edf.rel_deadline = attrs.edf.period;
 /* Add the posix thread as EVL thread */
 efd = evl_attach_self("/edfthread:%d",gettid());
/* Set the thread attributes under EVL scheduling attributes */
 ret = evl_set_schedattr(evl_get_self(),&attrs);
/* Make the thread periodic which is handled within the kernel */
 evl_set_thread_periodic();

 while (1) {
   /* user function */
   /* Wait for the thread period */
   evl_thread_wait_period(efd,&stats);
 }
 return NULL;
}
```

Listing 6.2 EVL EDF User-level API.

During our developmental phase, we consistently utilized the QEMU emulator, configured specifically to operate on two cores. After compiling the kernel, we loaded it into the QEMU environment. A test application, which was created and compiled with *libevl* integration, was transferred into the Buildroot environment. This application was subsequently initiated from the QEMU console. To facilitate the debugging process throughout these stages, we employed the `printk()` function. We executed various tests to verify the correct functioning of the `timer handle` (which wakes up the thread), `sched_enqueue` (which calculates the absolute deadline and adds the thread to the list), and `sched_pick` (which selects the highest-priority thread from the list ordered by the absolute deadline).

From the console print, we observed the schedule for threads created with periods of 4, 2, 6, and 3 seconds (for better printing). The print lines labeled with the `[evl]` index displayed the threads in the list during the `sched_pick` operation. Here, the release time (rel) and absolute deadline (abs) were also printed. The lines labeled with `[Thread x][start]` and `[Thread x][end]` indicate the beginning and end times of the user threads. The number of deadline misses and overruns were also logged. As depicted in Figure 6.9, we present a single time window where all four threads are listed, and the schedule pick operation and preemptions were correctly performed.

```
[  137.136620] [evl] Curr time = 136967181905 Thread in list: edfthread1:194, rel=136967089457, abs=140967089457

[  137.137950] [evl] Curr time = 136968512655 Thread in list: edfthread2:195, rel=136968464771, abs=138968464771
[  137.137951] [evl] Curr time = 136968513929 Thread in list: edfthread1:194, rel=136967089457, abs=140967089457

[  137.139625] [evl] Curr time = 136970187398 Thread in list: edfthread2:195, rel=136968464771, abs=138968464771
[  137.139626] [evl] Curr time = 136970188477 Thread in list: edfthread1:194, rel=136967089457, abs=140967089457
[  137.139626] [evl] Curr time = 136970189144 Thread in list: edfthread3:196, rel=136970143050, abs=142970143050

[  137.141949] [evl] Curr time = 136972511992 Thread in list: edfthread2:195, rel=136968464771, abs=138968464771
[  137.141950] [evl] Curr time = 136972512827 Thread in list: edfthread4:197, rel=136972467995, abs=139972467995
[  137.141951] [evl] Curr time = 136972513471 Thread in list: edfthread1:194, rel=136967089457, abs=140967089457
[  137.141951] [evl] Curr time = 136972514046 Thread in list: edfthread3:196, rel=136970143050, abs=142970143050

[  137.560064] [evl] Curr time = 137390625826 Thread in list: edfthread4:197, rel=136972467995, abs=139972467995
[  137.560068] [evl] Curr time = 137390630823 Thread in list: edfthread1:194, rel=136967089457, abs=140967089457
[  137.560069] [evl] Curr time = 137390631944 Thread in list: edfthread3:196, rel=136970143050, abs=142970143050

[  137.982536] [evl] Curr time = 137813097904 Thread in list: edfthread1:194, rel=136967089457, abs=140967089457
[  137.982540] [evl] Curr time = 137813102824 Thread in list: edfthread3:196, rel=136970143050, abs=142970143050

[  138.402620] [evl] Curr time = 138233182252 Thread in list: edfthread3:196, rel=136970143050, abs=142970143050

[Thread 1][start] Start Time: 136967192359
[Thread 2][start] Start Time: 136968516528
[Thread 2][end]
[Thread 2] no_of_dmiss = 0, dmiss_time = 0, overrun = 0
[Thread 4][start] Start Time: 137390635502
[Thread 4][end]
[Thread 4] no_of_dmiss = 0, dmiss_time = 0, overrun = 0
[Thread 1][end]
[Thread 1] no_of_dmiss = 0, dmiss_time = 0, overrun = 0
[Thread 3][start] Start Time: 138233230403
[Thread 3][end]
[Thread 3] no_of_dmiss = 0, dmiss_time = 0, overrun = 0
```

Figure 6.9 EVL `SCHED_EDF` scheduler thread pick and execution console debug. Time units are in ns. 'rel' referes to release time and 'abs' refers to absolute deadline of the thread.

To validate the scheduling policy, we employed a technique similar to that used by Lelli (2014) for validating the `SCHED_DEADLINE` policy in the Linux kernel. We configured four periodic threads with respective periods of 12ms, 16ms, 48ms, and 70ms. To simulate the computation time of the threads, we created dummy loads using *sqrt* and *pow* loops. Each process was statically bound to a CPU core using the `cpuset` mechanism, and the loads were incrementally increased from 0.5 (50%) to 0.95 (95%). Our observations from the `deadline_miss` field of `evl_thread_stats` revealed that under the `SCHED_EDF` policy, no deadlines were missed. This outcome can be attributed to the fact that the load on each core never exceeded 1 (100%). These results assert that the `SCHED_EDF` scheduling policy is operational under the EVL core. We have successfully applied the kernel patch to run on an *i5-7200U 4-core CPU*. This implementation serves as a basis for the further development of the ED-H energy-aware scheduling algorithm.

6.3.3 SCHED_EDH Xenomai scheduler policy

While we have successfully developed a simulator for the ED-H scheduler, the complexity of developing the scheduler within an operating system framework remains high, primarily due to debugging challenges. We created a new scheduling structure, `evl_sched_edh`, that maintains the same scheduling class structures as `evl_sched_edf`. We set the weight of `evl_sched_edh` to `6`, signifying it is the highest amongst all EVL scheduler classes. Like `SCHED_EDF`, the queue for `evl_sched_edh` is designed as a linked list that orders tasks by their absolute deadlines, complying with RULE 2 ED-H, which is to pick the thread following the EDF policy. We included additional members WCET and WCEC in the `evl_edh_entity` structure, as illustrated in Listing 6.3. These additional parameters are crucial for calculating the ED-H scheduler's two main parameters: slack time and preemption slack energy.

```
struct evl_edh_entity {
  struct list_head rq_next;

  /*thread params*/
  ktime_t release_time; /* Thread first arrival time */
  ktime_t wcet;     /* worst case execution time */
  ktime_t rel_deadline; /* relative deadline */
  ktime_t period;     /* period */
  u64 wcec;       /* worst case energy consumption */
  u64 power_max;      /* Thread maximum power */

  ktime_t job_release_time; /* instance release time */
  ktime_t abs_deadline;     /* absolute deadline */
  u64 job_instance;     /* number of job instance */

  ktime_t prev_exec;
  unsigned int new_thread; /* flag for new edh thread */

  /*Thread timers*/
  struct evl_timer edh_timer;
  struct evl_wait_queue wq;
  struct evl_poll_head ph;
  bool ticked;
};
```

Listing 6.3 EVL EDH thread entities.

The time and energy demand bound function, which calculates the time and energy demand, necessitates *floor* and *ceil* math functions from the C library. These

functions are not readily available in the kernel space due to the kernel's lack of direct support for floating-point operations [Corbet et al. (2005)]. To circumvent this limitation, we employed the `DIV_ROUND_UP` macro for *ceil* operations and a custom macro for *floor* operations that can handle negative numbers. The traversal of EVL threads is facilitated by `for_each_evl_edh_thread`, while list traversal is handled by `list_for_each_entry_safe`.

A critical challenge lies in identifying when to perform a schedulability check. Following the computation of parameters, the ED-H rules are verified. We employ flags to denote the `EVL_EDH_THREAD_REQUEUE` and `EVL_EDH_THREAD_RUN`, representing the *idle* and *active* states of the ED-H rule, respectively. Listing 6.4 shows the use of the user-level API for creating an EVL thread scheduled under the `SCHED_EDH` policy. We represent the WCEC parameter unit in μW to maintain uniform attribute units within the Linux power supply class[18].

```
void *Application(void *parm)
{
  int efd, ret = 0;
  struct evl_sched_attrs attrs;   /* evl scheduling attributes */
  struct evl_thread_stats stats;  /* thread statistics */

 /* Set the thread scheduling policy */
  attrs.sched_policy = SCHED_EDH;
 /* Put 0 to create synchronous threads */
  attrs.edh.release_time = 0;
 /* WCET of the thread in ns (nanosecond) */
    attrs.edh.wcet = 1000000000;
 /* Period of the thread in ns (nanosecond) */
  attrs.edh.period = 2000000000;
 /* Relative deadline of the thread in ns */
  attrs.edh.rel_deadline = attrs.edh.period;
 /* WCEC of the thread in µW */
    attrs.edh.wcec = 550000;
  /* Add the posix thread as EVL thread */
  efd = evl_attach_self("/edhthread:%d",gettid());
 /* Set the thread attributes under EVL scheduling attributes */
  ret = evl_set_schedattr(evl_get_self(),&attrs);
 /* Make the thread periodic which is handled within the kernel */
  evl_set_thread_periodic();

  while (1) {
```

[18]https://docs.kernel.org/power/power_supply_class.html

```
    /* user function */
    /* Wait for the thread period */
    evl_thread_wait_period(efd,&stats);
  }
  return NULL;
}
```

Listing 6.4 EVL EDH User-level API.

To validate the scheduler, which requires energy parameters for calculating preemption slack energy, we developed a kernel module to simulate energy dynamics. The kernel module, when loaded, initiates a kernel thread that runs on a separate core. For simplicity, we assumed a constant harvested power that updates the battery level periodically (every 1s), as the test thread periods are considered in seconds. Using `evl_update_account`, which updates the `prev_exe` entity of the thread, the energy to discharge is relayed to the energy simulator. The callbacks `evl_get_bat_remaining_energy()` and `evl_get_harvested_power()` are employed to fetch the remaining battery energy and power harvested during a specific time interval from the loadable kernel module, respectively. This approach significantly simplifies the kernel module that emulates dynamic battery energy and harvested power, making it straightforward to replace the simulated battery energy and harvested power with actual values. Any required modifications can be readily made within the loadable kernel module.

However, due to the intricate nature of the scheduler and debugging difficulties, complete verification of the scheduler's functionality remains challenging. Power consumption measurement at the thread level is complex because power is not consumed by threads directly. Instead, various hardware components including the CPU, memory, and I/O systems consume power when executing instructions on behalf of threads. Furthermore, the low granularity of energy creates difficulties in determining the scheduler's practical value at the system level, especially for devices like mobile robots.

6.4 Limitation

While the characterization of tasks in a simulation environment is relatively straightforward when considering energy constraints, the complexity significantly escalates when translating this into a realistic scenario. We have successfully incorporated the functional elements of the ED-H scheduler into the operating system framework, yet it remains a challenge to create a scenario that accurately illustrates the efficacy of the

decisions made by the ED-H scheduler. This difficulty stems from the characterization of the energy parameters and the inherent debugging challenges within the kernel framework. Despite the existing difficulties in modeling the Worst-Case Execution Time (WCET) and Worst-Case Energy Consumption (WCEC) for the threads [Sieh et al. (2017),Wägemann (2020)], this thesis does not focus on this aspect.

At this stage, the impact of scheduling threads at the operating system level, where the granularity of energy is relatively small compared to the device's total energy consumption, doesn't seem significant. This has led us to reconsider the criterion of energy at a broader scope, (i.e., at the application level), where the total energy requirements of the mobile robot can be considered. Additionally, it is notably complex to design threads, especially the ROS threads, with consideration of the energy parameter. We propose that the ED-H scheduler would be highly effective at the system level for threads dealing with transmission scenarios. Nonetheless, considering these complexities, we recommend the implementation of the ED-H scheduler at the application level rather than at the system level. This approach not only simplifies the process but also provides a more encompassing perspective on energy management, thereby maximizing its potential advantages for mobile robots.

We have decided not to proceed with the implementation of the ED-H scheduler with the dynamic priority ceiling protocol enhancement proposed in Chapter 5. We recognize that Xenomai has a priority ceiling protocol, which opens up the possibility of enhancements for the dynamic priority-based scheduler. Nevertheless, if adherence to deadline constraints is a requirement, we could utilize the `SCHED_EDF` policy to schedule the mobile robot threads.

6.5 Conclusion

This chapter delivered a deep exploration of the Xenomai framework's scheduling scheme. We started by outlining the prerequisites for implementing the energy-aware scheduler ED-H in the real-time operating system. As a primary step, we developed REACTSim, a simulator tool that accommodates energy considerations. We provided a guide for using this tool, which integrates various scheduling policies. A comparison of task scheduling with energy constraints using both the EDF and ED-H policies showed that the ED-H delivers results as anticipated, leading to an energy-neutral operation.

Subsequently, our focus shifted towards implementing the energy-aware scheduling scheme as a kernel scheduler. To facilitate understanding, we detailed the scheduling

aspects of the Linux kernel and Xenomai kernel. The action of the EVL scheduler core when the system schedules standalone Linux kernel threads and real-time EVL threads was clearly outlined. We initiated the implementation process with the `SCHED_EDF` policy, which was not initially present in Xenomai's scheduling class, given that ED-H is based on the EDF policy. The dynamic priority nature of the scheduler was verified, and the scheduling policy was validated using the deadline miss metric.

In our effort to implement the ED-H scheduler, we encountered several challenges, especially in validating and debugging the intricate scheduler framework. The complexity of characterizing energy parameters and estimating the WCET and WCEC further added to the difficulties. Despite these challenges, we believe that the ED-H scheduler, when implemented at the application level rather than the system level, would significantly contribute to energy management, thus enhancing the operational efficiency of mobile robots and other similar devices.

Chapter 7

Energy-Neutral Real-Time Mobile Robotic Operation

This chapter brings into focus the application-level utility of energy-aware scheduling theory, specifically tailored for mobile robots equipped with energy harvesting technologies. We present a novel and initial proof-of-concept attempt at implementing an energy-aware scheduling scheme for real-time energy harvesting mobile robotic devices. By implementing this strategy, we aim to maintain energy-neutral operation. In Section 7.1, we will provide a clear problem formulation, setting the stage for identifying the necessary requirements for the successful functioning of our energy-aware scheduler, ED-H, at the application level. In Section 7.2, we first review previous works to gain insight into the mission context of the mobile robotic device. We describe existing mission allocation approaches; and we put in light that mission allocation strategies do not consider energy-harvesting devices. In Section 7.3, we characterize the mission model with precedence and energy constraints. We delve into the concept of precedence constraints associated with real-time computing tasks and their relevance at the mission level for mobile devices. In Section 7.4, we provide crucial details of the energy estimation techniques that could be used to define the worst-case mission energy (WCME), predict future energy that may be harvested during a given time window, and monitor the battery's energy. These elements are vital for the performance of the ED-H scheduler. Before deploying these techniques in a real robot, we validate the proof of concept through simulations. In Section 7.5, we perform two simulations: one with a simulation tool to demonstrate that the mission characterization is valid for the context of mobile robotic devices, and another simulation of a robot in Gazebo with precedence and energy constraints. We strive to simulate a near-real-world scenario, though it remains a considerable challenge. Finally, Section 7.6 we provide the implementation details on a real robot. We highlight the limitations due to technological barriers. We finally propose future methods to achieve energy-neutral operation while mitigating the barriers.

7.1 Problem Statement

The central objective of this dissertation is to research and devise a strategy for achieving energy-neutral operations in industrial mobile robots. Despite the valuable insights gained from our initial approach, which focused on evaluating the energy-aware scheduler at the operating system level, we encountered significant limitations, particularly when attempting to verify the functionality of the scheduler at the operating system level proved to be challenging.This challenge motivated us to reassess the broader framework of energy management in mobile robots, leading us to concentrate on a more granular level of energy management.

Industrial mobile robots perform a range of tasks, or 'missions', offering an ideal context for examining and manipulating energy granularity. By addressing energy management at this application level will simplify the complexities encountered in earlier stages, offering a more practical framework for implementation and a streamlined development process. Nevertheless, certain requirements must be met to successfully establish the concept of energy-neutral operations at the application level.

The initial step requires correlating the essential components of the energy-aware scheduler, ED-H, with those of the robotic mission. This correlation can address the complex problem of characterizing the WCET and WCEC requirements of the scheduler. Further, we need to consider additional elements, such as the retrieval of the storage unit's energy level and an estimation of energy to be harvested. These variables are critical for calculating the preemption slack energy for the scheduler. Identifying strategies to simplify these complexities and meet these prerequisites is crucial for the desired functioning of the scheduler.

To validate our new approach, we plan to conduct tests using actual technologies and robots. However, given the challenges of the real-world environment, we first plan to verify our approach in a simulated environment. We will use a TurtleBot[1], equipped with ROS2 navigation[2], running on the Gazebo simulator[3] and an energy simulator for preliminary testing. This phase will enable us to understand the operations of the ED-H scheduler and assist in the development process. Upon successful simulation testing, we will then proceed to real-world implementation, with the aim of achieving energy-neutral operation for industrial mobile robots. By navigating this intricate landscape, we will gain insights into the domain of industrial mobile robot energy management. As we untangle these complexities, we will move closer to our ultimate

[1]ROS2-Turtlebot
[2]https://navigation.ros.org/
[3]https://docs.ros.org/en/humble/Tutorials/Advanced/Simulators/Gazebo/Gazebo.html

goal of creating an energy-neutral operation for industrial mobile robots, marking a significant milestone in robotic efficiency.

7.2 Previous Works

We will review some work that diverges from the traditional approaches focusing on energy management for mobile robots presented in Chapter 3. These studies focus specifically on power constraints and mission-level considerations in the context of mobile robots.

Power-aware scheduler

The study conducted by Liu et al. (2001a) serves as a seminal work in the realm of power-aware scheduling algorithms for robots. Their study utilizes NASA/JPL's Mars rover to demonstrate the efficiency and applicability of their proposed scheduler. They propose a system that relies on two power sources: a non-rechargeable battery and a solar harvesting unit. Their primary motivation is to optimally utilize energy from the free source - solar energy, which cannot be stored. Additionally, they aim to improve battery usage by controlling the system-level power curves. The power-aware scheduler presented in this paper is a key component in the proposed IMPACCT system-level design framework. Its primary function is to create a schedule based on the constraints of timing and power at a system level, thereby enabling efficient energy use. It first constructs a constraint graph based on the given problem. This graph outlines the relationships between tasks and their execution timings. The scheduler then searches for a time-valid schedule where all tasks satisfy their respective timing constraints, ensuring no deadlines are missed. After a time-valid schedule is obtained, the scheduler applies the maximum power constraint. This is achieved by redistributing tasks and manipulating slack times to eliminate power spikes, ensuring that the system does not exceed its maximum power capacity at any given time. The final step involves the application of the minimum power constraint. Here, the scheduler reorders tasks within their slack times to reduce power gaps, thus improving the utilization of minimum power. This step aims to take full advantage of available power resources and to manage power jitter effectively. Despite these significant capabilities, the scheduler falls short to support energy neutral operation. Energy neutrality is a scenario where a system's energy demand is perfectly balanced by its energy harvesting.

Mission Manager

Dinh-Khanh (2020) presented an noteworthy research work on Quality of Service (QoS) aware energy management for mobile robots, focusing on self-adaptation, which allows a system to adjust its behavior in response to the operational environment [Macías-Escrivá et al. (2013)]. In this context, QoS refers to performance metrics such as obstacle safety distance, mission completion, and energy consumption reduction by dynamically adjusting the robot's velocity using the Monitoring, Analyzing, Planning, and Executing (MAPE) paradigm. The decision-making process features a local Mission Manager that utilizes reinforcement learning-based decision-making to automatically reconfigure mission-specific parameters, minimizing performance and energy objective violations. A global Multi-Mission Manager employs rule-based decision-making and case-based reasoning techniques to monitor system resources and Mission Manager responses, reallocating energy budgets and regulating the quality of service. Despite the innovative approach, this work has limitations in its management methodology, as it is primarily data-driven, focusing on monitoring necessary metrics rather than events. The proposed algorithms also require careful effort to ensure learning performance, and the study's primary focus is on energy savings and self-adaptability.

Mission Allocation

Task (or Mission) allocation is a widely studied area in mobile robot research. Efficient task allocation is a necessity for multi-robot systems operating in dynamic environments. It allows robots to adjust their workloads in response to the actions of other robots or the system's missions to enhance overall device performance. While various techniques and requirements exist in the context of multi-robot systems, an allocation problem deals with tasks requiring single or multiple robots for completion [Roche (1999), Liekna et al. (2013), Aziz et al. (2022)]. Multi-robot task allocation (MRTA) is an optimization problem that aims to solve this allocation by either minimizing a cost function or maximizing a profit. The cost function could be the distance to travel [Elango et al. (2011)], time for completion [Matarić et al. (2003)], or energy [Kaleci and Parlaktuna (2013)]. With these techniques, missions to be performed by a robot can be queued. This queueing method can be combined with our approach to schedule missions for a single robot. This is the first study to explore an energy-neutral operation strategy specifically tailored for an industrial mobile robot, marking a novel approach in the field of mobile robotics.

We recall that the objective of our research is to create an industrial mobile robot capable of achieving energy-neutral operation. This signifies that all the energy required by the robot is harvested from its surroundings, eliminating the need for traditional charging stations and reducing periods of idleness, thereby enhancing productivity. Despite our ambitious goal, we acknowledge the need for significant advancements in energy harvesting technologies. Another crucial element of our research is the introduction of mission characterization specific to energy-harvesting mobile robotic systems. We recognize that increasing the autonomy duration of the robot remains a significant challenge. Existing research suggests various energy-saving strategies to minimize energy usage and prolong operation time. However, we believe that harnessing ambient energy sources via advanced harvesting technologies presents a more sustainable and promising solution to maximize not only the robot's autonomy duration but also its overall sustainability. In essence, through our research, we aim to realize the true potential of energy harvesting technologies in the realm of mobile robotics.

7.3 System Characterization

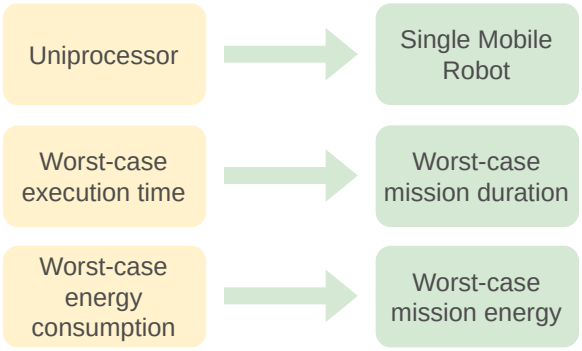


Figure 7.1 Analogies of Real-time system terms to mobile robots.

In this work, we establish an essential correlation between real-time system theory and mobile robots. The relationship is critical as it will allow us to apply real-time computing methods for higher-level mobile robot applications. Figure 7.1 represents the principal relationship we establish. Given that a single mobile robot can execute only one mission at a time, it can be analogously treated as a uniprocessor system that executes only one task at a time. Therefore, we refer to real-time computing tasks as the missions performed by the mobile robot. Consequently, all relevant aspects associated with the system workload model of a Real-Time Energy Harvesting System

(RTEHS), such as worst-case execution time (WCET), worst-case energy consumption (WCEC), and deadlines, can be translated into worst-case mission duration (WCMD), worst-case mission energy (WCME), and mission deadlines, respectively.

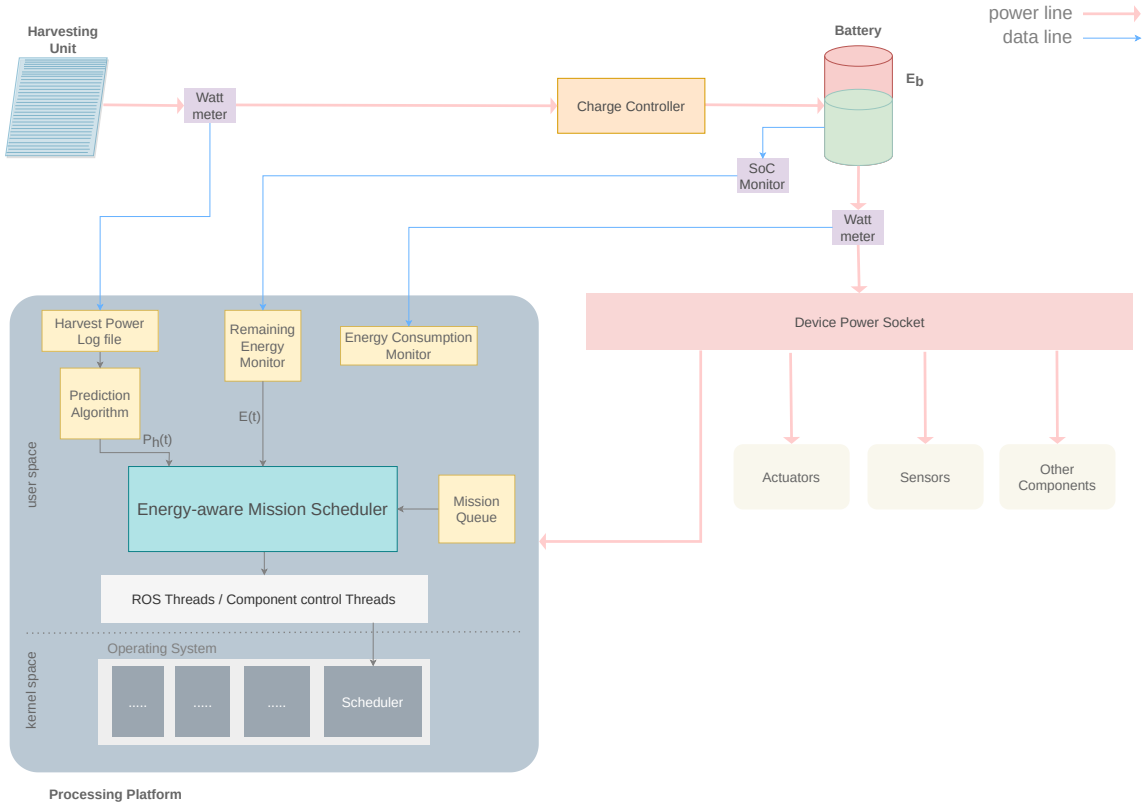


Figure 7.2 Real-time energy harvesting mobile robot.

In this work, the established model of a real-time energy harvesting system is adjusted to function as a real-time energy harvesting mobile robot, as depicted in Figure 7.2. An energy-aware scheduler is implemented at the user level to schedule missions with deadline, energy, and precedence constraints. We employ the ROS2 middleware, together with the nav2[4] stack and Simple Commander API[5], in order to faciliate navigation commands that are compatible to handle missions with the scheduler. The following subsections will provide detailed characterization of the mission model and the energy model.

[4]https://navigation.ros.org/
[5]https://navigation.ros.org/commander_api/index.html

7.3.1 Mission Model

We define a new workload model for mobile robots: $\mathcal{M} = \{J_i(r_i, C_i, d_i, E_i), i = 1 \; to \; n\}$. Here, each mission is treated as a job J_i with a release time r_i, worst-case mission duration C_i, mission deadline d_i, and worst-case mission energy E_i. For sake of simplicity, although some missions executed by the mobile robot could be periodic, we exclude periodic missions. However, it should be noted that these periodic missions can be directly associated with the periodic tasks discussed in the previous chapters.

Figure 7.3 Precedence constraints for missions.

We then consider synchronization issues framed in the real-time perspective for mobile robots. Imagine a scenario where two jobs, or missions, denoted as J_a and J_b, must be performed by the robot. These jobs are scheduled without energy constraints using an Earliest Deadline First (EDF) scheduler. In the situation depicted in Figure 7.3, job J_b preempts job J_a when the robot is performing a mission - for instance, picking up a trolley. The robot cannot execute another job until it completely finishes its current work, which makes the preemption invalid. This issue is similar to the precedence constrains problem in real-time computing. Two different approaches from real-time computing theory can be employed to address this issue. One relies on resource access techniques discussed in Chapter 5, where we had expanded the proof to show how the optimal energy-aware scheduler ED-H handles shared resource constraints. In this case, we can consider the robot itself as a resource and apply a mutual exclusion mechanism using the ED-H scheduler with dynamic priority ceiling protocol.

However, our primary focus is on implementing a robust workload model for missions using a Direct Acyclic Graph (DAG) that indicates precedence relations. This approach simplifies the job definition process by representing them as a graph. We will provide a more detailed explanation of this approach in the sections that follow. Before, let us establish the set of missions as a job set represented by $J = \{J_1, \ldots J_n\}$, which corresponds to its respective precedence graph $\mathcal{G}$. Here, J also signifies the set of nodes present in the graph $\mathcal{G}$. The graph $\mathcal{G}$ signifies a partial order $\prec$ on J such that

$J_i \prec J_j$ only if a directed path exists in $\mathcal{G}$ that leads from node J_i to node J_j. In such a situation, J_i is a *predecessor* of J_j (or conversely, J_j is a *successor* of J_i). We use $J_i \rightarrow J_j$ to represent a precedence relation between J_i and J_j. It is important to note that a job set can have one or more jobs that serve as the beginning or end. A beginning job does not have a predecessor job, and similarly, an ending job does not have a successor job.

Deadline jobs with precedence constraints

The challenge of scheduling a set of jobs, with deadlines and release times, under precedence constraints, can be solved with polynomial time complexity. Chetto et al. (1990) proposed an effective method for this problem by transforming a group of *dependent* jobs into a set of *independent* jobs by adjusting the timing parameters of each job. The main idea here is to adjust all the start times and finish times in a way that no job begins before its predecessor jobs have completed. Also, no job should preempt the jobs that succeed it. This is achieved through a topological sort, adjusting the deadlines of jobs sequentially, starting with the jobs that do not have any successors. Once these jobs deadlines are adjusted, the method proceeds to adjust the deadlines of successor jobs.

For each job that is at the end within the partial order, the deadline is set as it is. For all other jobs, the deadline calculation of d_i^* for J_i is done using a particular formula, as follows:

$$d_i^* = \min(d_i, d_j^* - C_j \parallel J_i \rightarrow J_j) \tag{7.1}$$

The same logic applies to the release time of each job. For every job at the start within the partial order, the release time remains the same. For all other subsequent jobs, the earliest possible start time for J_i is determined as follows:

$$r_i^* = \max(r_i, r_j^* - C_j \parallel J_j \rightarrow J_i) \tag{7.2}$$

With these adjustments, the nature of the problem remains consistent. Therefore, for any two jobs where one succeeds the other, the start and end times of the predecessor job are always less than or equal to those of the successor job, thereby preserving the order of tasks and their precedence constraint.

Further with this approach, Chetto and Osta (2022) advanced the technique by establishing a method to uphold the precedence constraint using the existing deadline assignment protocol, specifically tailoring it to the requirements of the ED-H scheduler

instead of the EDF scheduler, with no energy limitation. This achievement affirmed the scheduler's capacity to aptly handle jobs considering constraints related to deadlines, energy, and precedence. With this foundation, the current research aims to integrate a mission scheduler that efficiently employs ED-H scheduler with precedence constraints for a real-time energy harvesting mobile robot. An illustrative example of the ED-H scheduler with precedence constraints for real-time tasks can be seen in the paper. Additionally, we present the mission schedule in Figure 7.8.

7.3.2 Energy Model

Now, let us describe precisely the model we consider for the energy harvesting mobile robot. We consider a mobile robot powered by a lithium-ion battery. We utilize a state of charge (SoC) monitor to read the capacity of the battery which is denoted as E_b. At any given time t, the energy level of the battery is represented as $E(t)$. It is important to note that the robot cannot execute any mission when the energy level falls below a defined threshold, E_{min}. This work is centred around a real-time energy harvesting mobile robot. The primary harvesting technology employed in this context is solar power, utilizing photovoltaic (PV) cells, due to the vast array of prediction algorithms available for this technology. The power harvested by the unit at a given time t is denoted as $P_h(t)$. Furthermore, the total energy generated by the harvesting unit over a specified time interval $[t_1, t_2)$ is represented as $E_h(t_1, t_2)$.

We have defined the mission model and the energy model specifically for a real-time energy harvesting mobile robot. Following sections we will present methodologies to calculate the worst-case mission duration (WCMD) and worst-case mission energy (WCME) values for the mobile robots. In addition, we will discuss about the battery's energy level monitoring and energy prediction techniques to forecast future harvestable energy. These factors are crucial for the efficient operation of the energy-aware mission scheduler ED-H. Our ultimate goal is to implement a mission scheduling strategy that allows for an energy-neutral mobile robot. The aim is to eliminate the need for recharging stations, which currently leave the robot entirely inactive during recharging periods. However, realizing this goal demands a thorough exploration of energy-harvesting platforms. We therefore assume, for the purpose of this study, that the robot is capable of harvesting energy sufficient to meet its energy demand. Moreover, the current mission characterization of industrial mobile robots mainly considers the notion of priority, with tasks being scheduled based on this attribute. We propose to extend this by introducing notions of deadline, energy, and precedence constraints.

The inclusion of these elements will enable the robot to handle tasks more intelligently, leading to increased productivity. By adopting this approach, we can effectively determine the optimal requirements for storage capacity (i.e., battery size) and the dimensions of the energy harvesting unit. Consequently, this leads to potential cost reductions in the manufacturing of the robotic device.

7.4 Estimation Techniques

In the context of mobile robots, the process of estimating the WCMD and WCME is less complex when compared to the estimation of the WCET and WCEC. This relative simplicity arises due to the larger granularity which permits their estimation through practical techniques and dedicated libraries. Firstly, in the context of a mission, time values are defined in "seconds". The total duration of the mission can be estimated using a practical technique: running the missions and measuring the total duration. For the worst case, this method must be executed multiple times to measure the WCMD. Alternatively, a time estimation library[6] can be used to estimate the time it will take to travel from the start point to the goal point at the robot's maximum velocity. We employ a similar technique provided by the feedback from the Simple Commander API. This technique calculates the total estimated time to complete the navigation based on the generated path and the robot's average velocity. However, we have not considered scenarios where the robot may be obstructed by obstacles. If such a situation arises, we can define a timeout after which the job can be cancelled to prevent the subsequent jobs from missing their deadlines.

Another significant challenge involves estimating the energy used during a mission. An initial, straightforward method might be to measure the energy level of the battery at the beginning and at the end of the mission. However, this method does not offer a precise estimate of the mobile robot's energy consumption. Consequently, in the next subsection, we will delve into more sophisticated techniques and provide a mathematical energy estimation approach specifically tailored for the robot under study.

7.4.1 Energy Estimation

Power consumption in a robot is determined by the number of active components. On the other hand, energy consumption depends on both power consumption and the duration of each component's activity [Parasuraman et al. (2015)]. In the case

[6]https://github.com/oKermorgant/mrs_monitor

of industrial mobile robots, device components are generally active for most of the time. Consequently, energy consumption can be broadly categorized into two groups: navigation system (which includes motors for navigation) and other subsystems, which include computing platform and sensing units. Figure 7.4 illustrates the sequence of power consumption for various components in industrial mobile robots. For smaller robots, navigation power consumption is lower compared to other subsystems [Mei et al. (2005), Parasuraman et al. (2015), Mohamed et al. (2021)]. Conversely, larger robots operating in industrial or outdoor settings have navigation systems as their primary power consumers [Morales et al. (2006), Xie et al. (2016), Liu et al. (2019)].

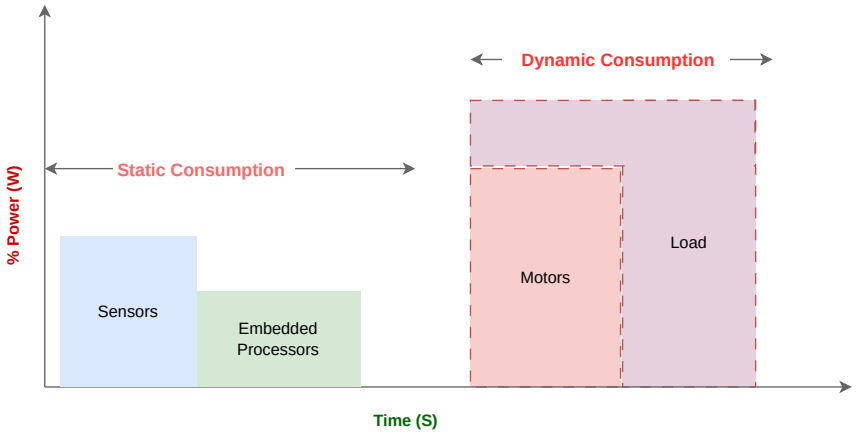


Figure 7.4 Power consumption of various components in industrial mobile robots.

In our assessment of an industrial mobile robot's energy consumption, we utilized a component Yocto-Watt[7] component. This component was interfaced on the main device power distribution socket of the robot to measure the voltage and current drawn by the robot's various components. The Yocto-Watt component includes specialized APIs that allow us to determine the power consumption of the load. In an idle state with all components operational, the robot consumption amounted to 95W. The distribution of this power consumption is as follows: active motors utilized 30%, the processing platform consumed 15%, the sensors accounted for 35%, while remaining components like LEDs and electrical circuits, consumed the residual 15%. We conducted further tests with the robot navigating at its maximum speed of 1.6 m/s along a straight path. In this circumstance, the motors' energy consumption rose to account for 52% of the total power usage, indicating they were the primary power consumers. The collected data underscores the importance of encapsulating all energy-consuming components when managing energy at the application level. It is

[7]https://www.yoctopuce.com/EN/products/usb-electrical-sensors/yocto-watt

crucial to consider every component's energy usage for efficient energy management and potential improvements in the design and operation of the mobile robot.

Due to the complexity of estimating energy consumption for dynamic consumers, there is a need for defining an energy estimation model for the system. Energy estimation models for mobile robots quantify the energy usage of different subsystems and activities, helping in estimating and optimizing energy consumption. These models inform decision-making around motion and path planning, and power allocation [Mei et al. (2005)]. Additionally, they provide a benchmark for evaluating energy efficiency across various robotic systems or control algorithms [Kim and Kim (2007)]. Accurate models guide the design and selection of components that meet energy needs while delivering desired performance. By factoring in velocity and acceleration, these models calculate power and, subsequently, energy consumption. Estimations of the maximum energy consumption can be achieved by considering the maximum velocity for a certain travel duration.

Research Work	Robot	Robot Type	Energy Estimation Model							Motor
			PF	KM	DM	MM	FC	SC	Load	
Mei et al. (2005)	Pioneer 3DX	DD	✓					✓		DC
Kim and Kim (2007)	Pioneer 3DX	DD		✓	✓	✗	✓	✗	✗	DC
Liu and Sun (2012)	Pioneer 3DX	DD		✓	✗	✗	✓	✓	✗	DC
Wahab et al. (2015)		DD		✓	✗	✗	✓	✓	✗	DC
Jaiem et al. (2016)	Pioneer 3DX	DD	✓					✓		DC
Jaramillo Morales et al. (2018) Jaramillo Morales et al. (2020)	Nomad Super Scout	DD		✓	✓	✓	✗	✓	✓	DC
Touzout et al. (2022)	Turtelbot-3	DD		✓	✗	✗	✓	✓	✗	DC
Gürgöze and Türkoğlu (2022)		DD		✓	✓	✓	✓	✓	✗	DC

Table 7.1 Summarizing the research works that provide energy-estimation model of mobile robots. Abbreviations: PF - Polynomial Fitting, KM - Kinematic Model, DM - Dynamic Model, MM - Motor Model, FC - Friction Coefficient, SC - Static Components, Load - payload, DC - Direct Current Motor.

Table 7.1 summarizes the research work that proposes energy-estimation models for differential drive mobile robots. In this context, the energy-estimation model refers to the estimation of the energy consumed during the operation of mobile robots. The criteria used to compare these works include robot type, energy estimation technique, and factors considered in the model. Polynomial fitting and mathematical modeling are two widely studied energy-estimation models. Polynomial fitting, which involves fitting the consumption of different velocity profiles. However it lacks accuracy. In contrast, mathematical models prove more accurate, as they take into account kinematic, dynamic, and motor models of the robot. Some research studies also take static friction on different surfaces and slopes into account when estimating energy consumption. It

is important to include static consumption of electronic components and sensors in the estimation model, as well as the load a robot carries. Jaramillo Morales et al. (2020) demonstrated that including the load in the energy estimation model leads to more accurate results. In addition to these works, Xiao and Whittaker (2014) conducted a comparison of energy expended by previous Mars and Lunar Rover missions based on their mass and speed. They provided a model to estimate the achievable range or distance a rover can cover with its full battery charge. This model can be useful for robots employed in exploration missions, where energy-efficient operation is crucial to extend mission durations and cover larger areas.

From the existing literature, we can derive the equation to calculate the energy consumption of the mobile robotic system:

$$E_c = \int_0^t P_{static}(t)dt + \int_0^t P_{dynamic}(t)dt \tag{7.3}$$

It is essential to develop an energy-estimation model specific to a particular robot, as the model depends on the robot's physical and dynamic properties. In this dissertation, we will provide an energy-estimation model tailored to the cobot under study, taking its mathematical model into consideration. This model will prove useful in estimating the energy consumed during the robot's mission, which is a significant parameter for the energy-aware mission scheduler.

Energy estimation model

We utilize a standalone energy consumption model of a two-wheeled mobile robot. As we stated previously the primary energy consumption sources in a robot are the motors and various other components, including processing platform, sensors, and electrical circuits. The energy consumption for motors encapsulates two significant parts: energy conversion into the robot's kinetic energy and the energy necessary to overcome traction resistance. This energy model is expressed as per the formula presented in Liu and Sun (2014).

$$\begin{aligned} E_c = & \int_0^t P_{static}(t)dt \\ & + \int_0^t \left(m \max\{v(t)a(t), 0\}\right) dt \\ & + \int_0^t \left(I \max\{\omega(t)\beta(t), 0\}\right) dt \\ & + \int_0^t \left(2\mu mg \max\{|v(t)|, |b\omega(t)|\}\right) dt \end{aligned} \tag{7.4}$$

In this equation, v and ω symbolize the linear and angular velocities relative to the robot's center of mass. The distance from the robot's center to the two wheels is represented by b. The robot's mass and the moment of inertia are denoted by m and I, while a and β denote the linear and angular accelerations respectively. The gravitational acceleration is denoted by g, and mg signifies the combined weight and payload of the robot. μ is the rolling friction coefficient, which is contingent on the ground surface type. By substituting the robot's parameters into this equation, we can estimate the energy that would be consumed for the travelled path.

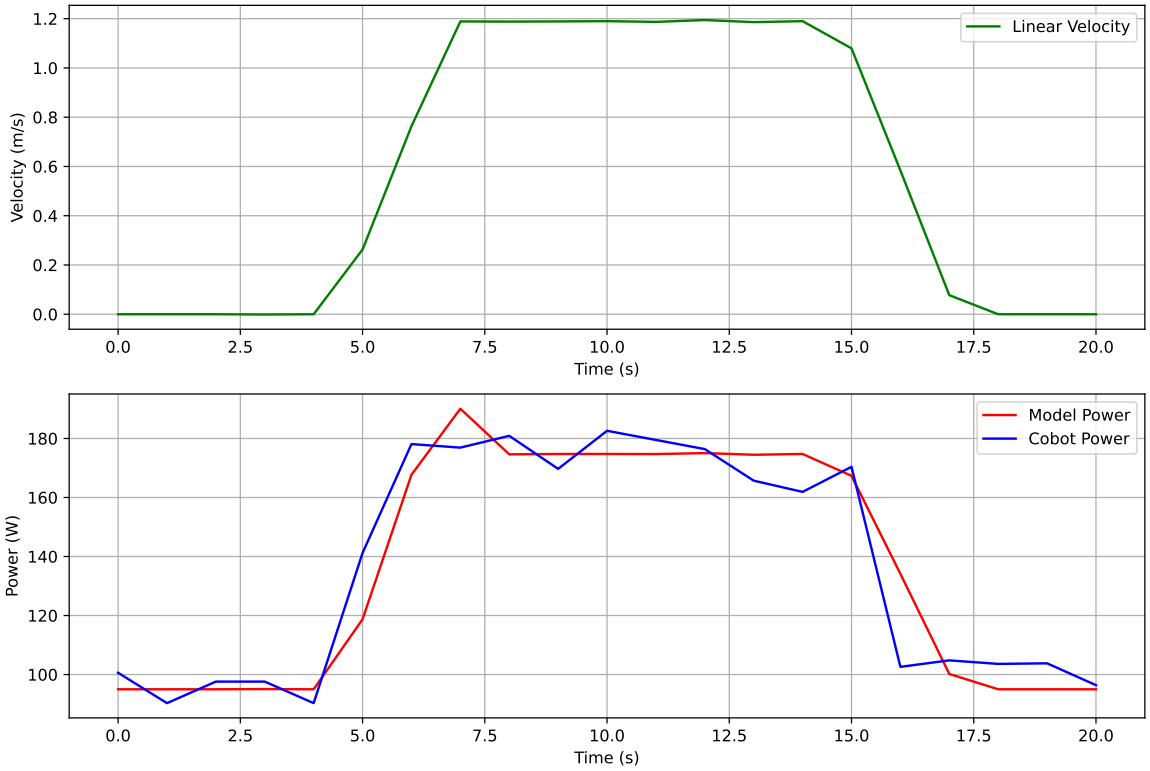


Figure 7.5 Energy model verification. Comparison between modeling and cobot results.

During our test with the cobot, we measured its power consumption while navigating at a linear velocity of 1.2m/s. We recorded the power consumption data and compared it to the energy model. As depicted in Figure 7.5, the model's results are in line with the experimental data, thereby confirming the validity of the parameters we identified. In addition, the model indicated a potential for energy regeneration during deceleration, with an estimated regeneration rate of approximately 5%. Exploring regenerative harvesting technologies in the future could be an interesting avenue of research. In Figure 7.6 we present a comparison between the energy model results and the robot's power consumption during the navigation phase of its mission. The velocity

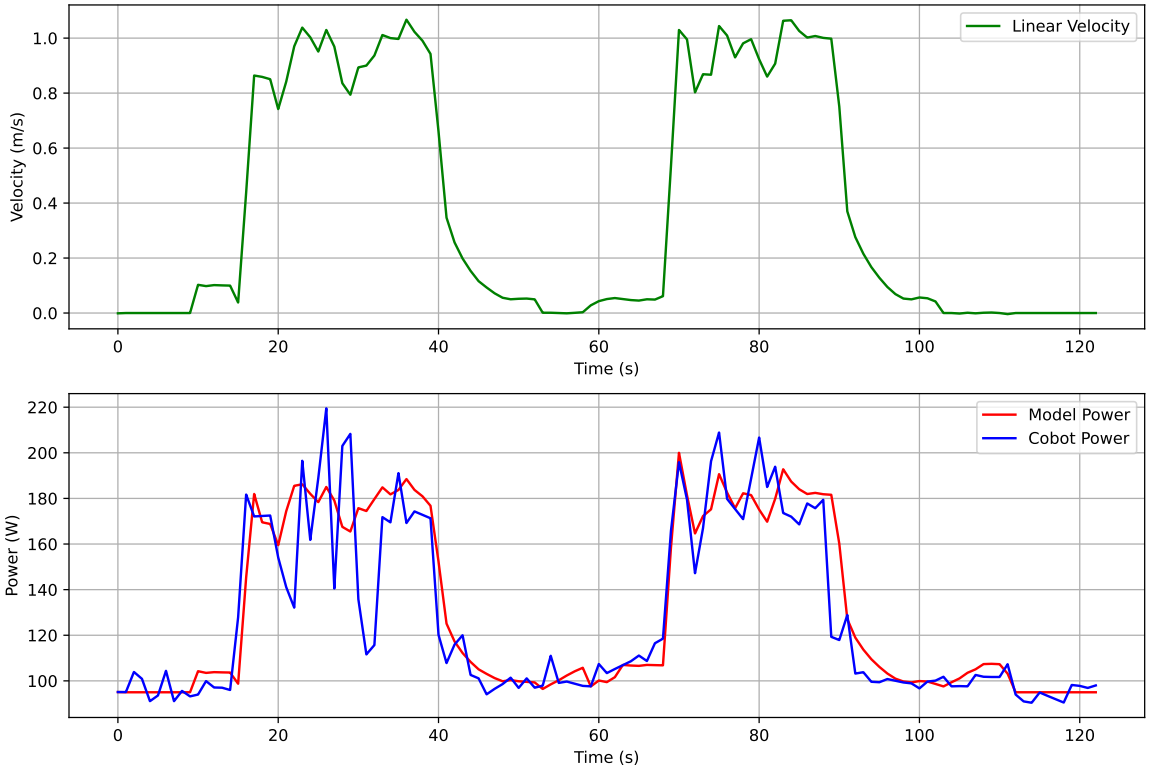


Figure 7.6 Comparison between modeling and cobot results for a mission navigation.

profile, recorded during the navigation, was incorporated into the energy model. To estimate the worst-case mission energy, we employ an approximation technique that considers the total mission duration and the robot's average velocity. We determine the average velocity by logging the velocity measurements across multiple mission iterations. Using these data, we can apply the energy model to estimate the worst-case mission energy. However, it is important to note that the worst-case mission energy estimation is an approximation and may not always be precise, especially in highly dynamic environments where motor acceleration varies due to frequent stops and starts.

7.4.2 Battery remaining energy monitoring

IMR uses a high-precision Coulomb Counter [Martin Murnane (2017)] to monitor the battery's State of Charge (SoC). This SoC, defined as the ratio of the remaining capacity of the battery to its rated capacity specified by the manufacturer, ranges from 0% when the battery is completely drained to 100% when it is fully charged. The remaining battery energy at any given time can be calculated by multiplying the SoC by the battery's total energy:

$$E(t) = total\ energy \times \frac{SoC(t)}{100} \tag{7.5}$$

Here, *total energy* signifies the product of (nominal) voltage (V) and (nominal) capacity (Ah). For better visibility by the scheduler, the expression of *total energy*, initially given in watt-hours (Wh), can be converted to joules. However, it is important to note that the total energy that a battery can store may decrease over time due to battery degradation factors such as aging and temperature effects. While these factors are not the focus of this dissertation, future research should consider techniques like machine learning approaches [Ren et al. (2019), Niri et al. (2020)] for estimating the remaining battery energy.

7.4.3 Energy-Harvest Prediction

The energy prediction algorithms for solar power harvesting were discussed in Section 3.6.2. However, all systems in which these techniques were tested were in a static position. In this work, the solar panels are mounted on a mobile robot, adding a dynamic motion factor to the uncertainty challenges of solar prediction. Predicting the energy to be harvested thus becomes a more complex task. While this prediction study is not the primary focus of this thesis and requires an entirely separate field of expertise, we opt for a simplified approach with a couple of assumptions. These assumptions are used to facilitate the verification of the scheduler.

In the first scenario, the harvested power is considered to be a constant for the sake of simplicity. This assumption is primarily used for preliminary simulations and provides a baseline for comparison with more complex scenarios. In the second scenario, we aim to simulate a closer-to-reality situation. We record the power harvested during the time window of a mission, defined as the period between the mission's release time and its deadline. For example, if a mission is to navigate from point A to B, the time window begins when the robot starts moving and ends when it reaches the destination. Running the mission multiple times allows us to calibrate the prediction by recording the power harvested during each run's time window. We can then employ the simplest prediction algorithm, Exponential Weighted Moving Average (EWMA), to predict the energy that will be harvested during the mission. This assumption holds reasonably well for indoor environments where the intensity of light does not vary significantly. However, outdoor scenarios present a much higher level of uncertainty. For such situations, a prediction algorithm like Weighted Cumulative Moving Average (WCMA) would be required.

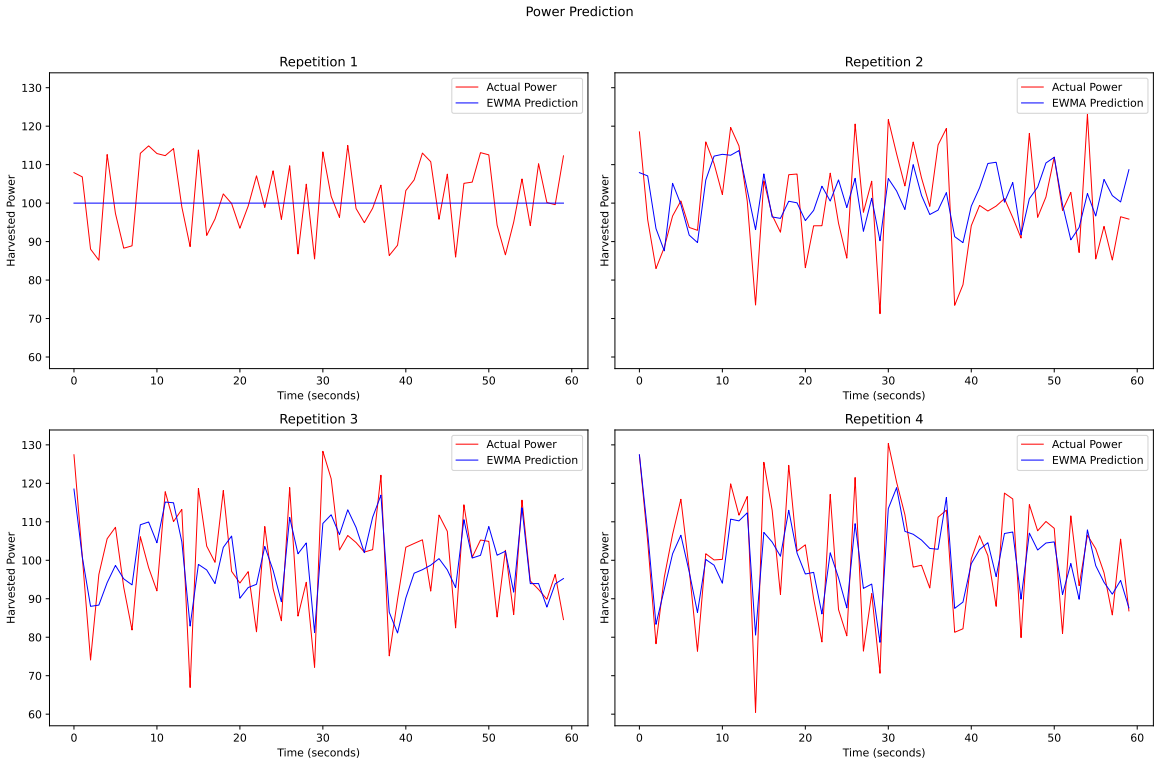


Figure 7.7 Evaluation of the EWMA prediction algorithm.

In order to illustrate the nature of our prediction algorithm in context of mission scenarios, we executed a simulation that utilizes randomly generated power values over the period of a 60-second mission. This simulation was conducted under the assumption that both the light intensity and the mission path would remain constant, leading to relatively predictable power production throughout the mission's duration. We used EWMA to estimate the power that the robot would harvest during the mission. Our test showed that this method was accurate, with less than 10% error, as shown in Figure 7.7. Recently, Stricker and Thiele (2022) conducted a comparison of EWMA prediction with Random Forest technique for indoor harvesting applications. Their proposed Random Forest predictor exhibited a Mean Absolute Deviation Percentage (MADP) of 27.4%, whereas the MADP for the EWMA methodology was significantly higher at 113.8%. However, their study considered indoor harvesting akin to the diurnal cycle outdoors, meaning that lights were switched 'OFF' during the night. In an industrial setting, we assume that the light intensity does not vary significantly, and if we consider a situation where lights are switched 'ON' all the time, EWMA may be a more suitable prediction method. The performance of the scheduler is dependent on the precise forecasting of future energy. The duration of the mission, defined

within a time window, is usually captured in 'seconds' but can extend to 'minutes' for missions of longer durations. Therefore, it becomes imperative to delve further into energy prediction algorithms, especially those suitable for dynamic and uncertain environments. This will undoubtedly enhance the performance of the energy-aware mission scheduler in our study.

7.5 Energy-aware mission scheduler

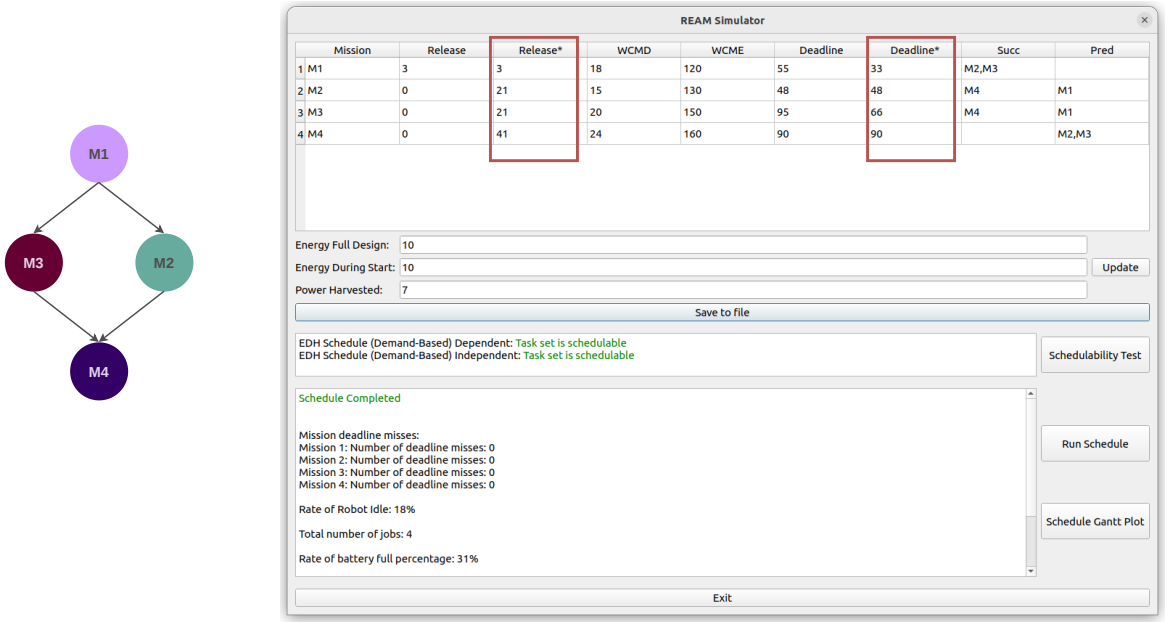
First, we introduce the modified optimal energy-aware scheduler designed for real-time computing tasks within mission scheduling. Algorithm 1, showcases the ED-H mission scheduler. Inputs for the scheduler include the mission class, which is characterized by release time, deadline, WCMD, WCME, and goal points. Other inputs include the remaining battery energy, predicted power to be harvested (using the prediction algorithm), full battery capacity, and energy threshold. The scheduler determines the state of the mission manager, which can either be active (RUN_MISSION) or idle (PAUSE_MISSION or IDLE), based on the ED-H scheduler's set of rules. The slack time (ST) and preemption slack energy (PSE) remain consistent. When the mission manager state is RUN_MISSION, the simple commander API is executed to launch the mission, using the mission's goal point (i.e., the destination that the robot must navigate to from its current location). The missions are managed and executed in accordance with the state of the manager.

Initially, we validated the precedence and energy constraints for the mission set using our simulation framework. To this end, we have modified the REACTSim tool to accommodate the mission set, and we have subsequently renamed this modified tool as REAMSim (Real-time Energy-Aware Mission scheduler Simulator). All corresponding parameters of a task set are modified to match those of a mission set. For the initial version of this tool, we have only allowed mission set input from a file, disabling other methods. Each mission's predecessor and successor can be defined in the respective column. After loading the mission set, we apply the transformation algorithm we discussed previously to convert the dependent mission set into an independent mission set. Following this transformation, we can perform a feasibility test for both dependent and independent mission sets. Finally, the missions can be scheduled, and the simulation can be visualized. Figure 7.8a illustrates the DAG model of the considered mission set, and the associated parameters for the mission are displayed in the table. We can observe both the release time and the deadline of the dependent and independent mission sets. The constraint here is that Mission 2 must not be executed before Mission

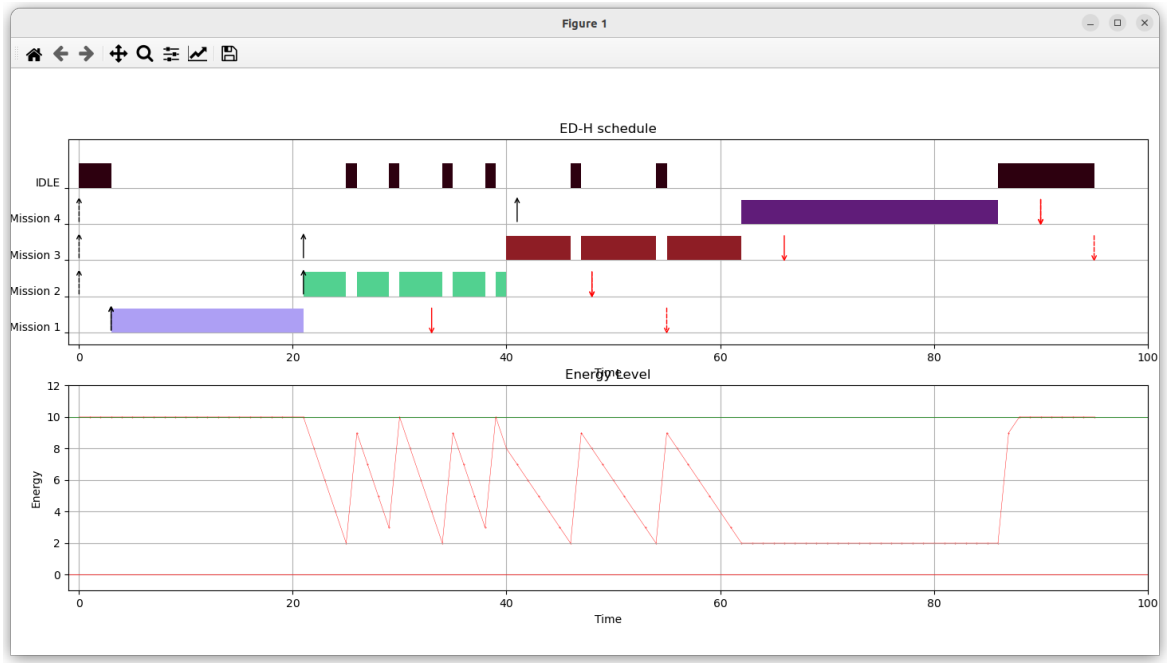
Algorithm 1 The ED-H Mission Scheduler

Input
Current time t, $\mathcal{M}$, $E(t)$, $P_h(t)$, E_b, E_{min}
Output Mission schedule

$t \leftarrow 0$
$Manager\_state \leftarrow IDLE$
$Q_m \leftarrow update\_ready\_queue$(M)
while (1) **do**
  $hpM \leftarrow select\_high\_priority\_mission(Q_m(t))$ //Rule 1
  $ST(t) \leftarrow slack\_time\_computation(t, Q_m)$
  $PSE(t) \leftarrow preempt\_slack\_energy\_computation(t, Q_m, E(t), P_h(t))$
  if $Q_m(t) \neq \emptyset$ **then**
    if $E(t) < E_{min}$ or $PSE(t) < E_{min}$ **then**
      $Manager\_state \leftarrow PAUSE\_MISSION$ //Rule 3
    else if $E(t) == E_b$ or $ST(t) == 0$ **then**
      $Manager\_state \leftarrow RUN\_MISSION$ //Rule 4
    else if $E(t) < E_b$ and $ST(t) > 0$ and $PSE(t) > 0$ **then**
      $Manager\_state \leftarrow PAUSE\_MISSION$ or
      $Manager\_state \leftarrow RUN\_MISSION$ //Rule 5
    end if
  else
    $Manager\_state \leftarrow IDLE$ //Rule 2
  end if
  if $Manager\_state == RUN\_MISSION$ **then**
    $run\_navigation(hpM)$
  else if $Manager\_state == PAUSE\_MISSION$ **then**
    $pause\_navigation(hpM)$
  end if
  $t := t + 1$
end while



(a) Mission Simulator Interface - ED-H.



(b) Mission schedule using ED-H and precedence constraints. Both the dependent jobs and independent jobs release and deadline are marked as dashed arrows and continuous arrows.

Figure 7.8 ED-H mission scheduling simulation.

1. As shown in the Gantt of the schedule displayed in Figure 7.8b, Mission 1 is executed before Mission 2, which satisfies the precedence constraints, and the same applies for Missions 3 and 4. Furthermore, the schedule also illustrates the energy level, clearly

showing that depletion does not fall below the threshold and that energy-neutral operation is achievable. As a subsequent step, we have simulated a realistic scenario of a mobile robot using the ROS simulation tools.

ROS2 Simulation

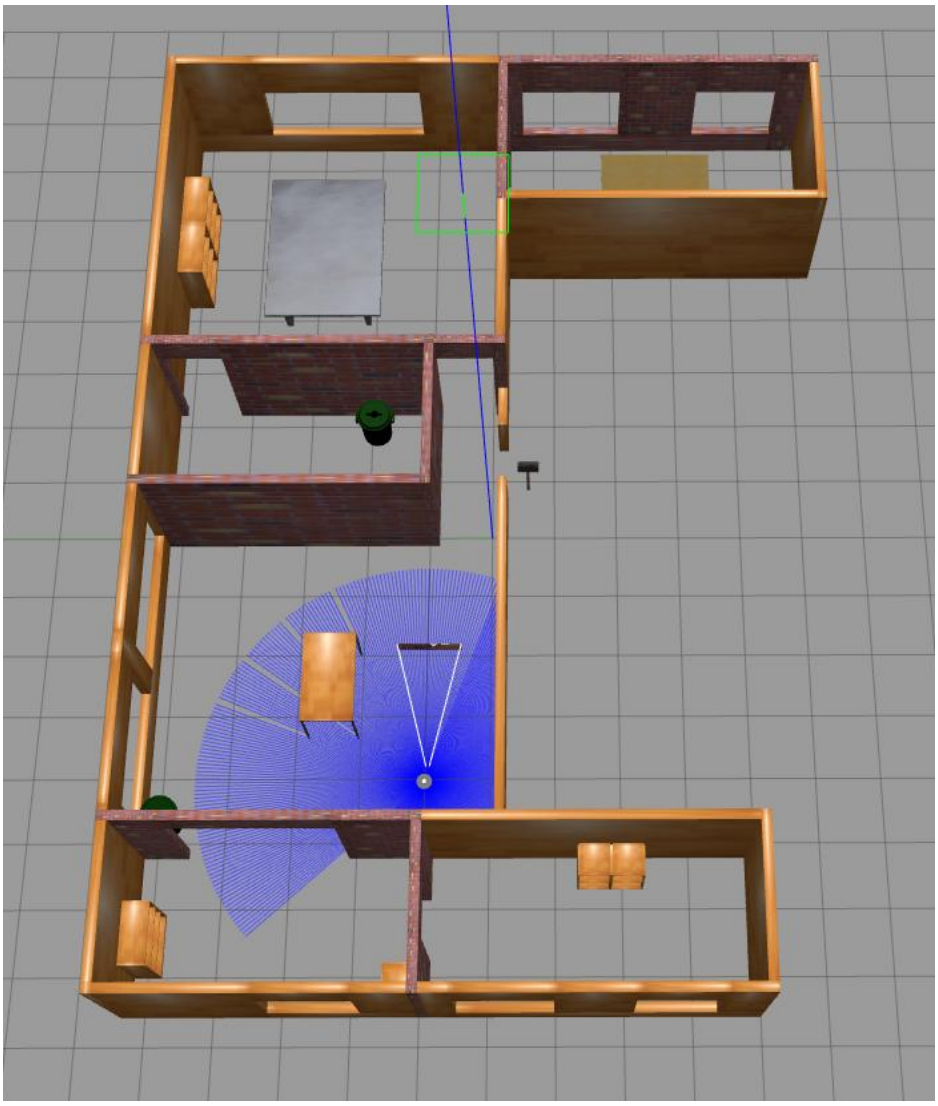


Figure 7.9 Gazebo environment with Turtlebot3 in house world.

In the ROS2 simulation, we implement the turtlebot3[8] simulation with Gazebo, using the environment setting of a house world as depicted in Figure 7.9. The purpose of this simulation is to command the robot to execute various missions, namely

[8]https://github.com/ROBOTIS-GIT/turtlebot3

navigating from one point to another, by utilizing the energy-aware mission scheduler ED-H. We receive the velocity at which the robot travels and, using the energy model of the turtlebot3, we simulate the instantaneous power consumption of the robot. The parameters of the robot, defined in the work of Touzout et al. (2022), are used for this simulation, including a static power consumption of 10W for the turtlebot. Given that the robot lacks energy-related equipment like a battery or a harvesting unit, we use separate ROS nodes to manage battery energy, harvested power, and power consumption estimation. Each node has respective topics to facilitate data communication amongst them. Upon launching the simulation with the turtlebot3, the navigation stack (nav2), and the energy simulator, a complete node graph and corresponding topics recorded using the rqt_graph[9] is shown in Figure B.1.

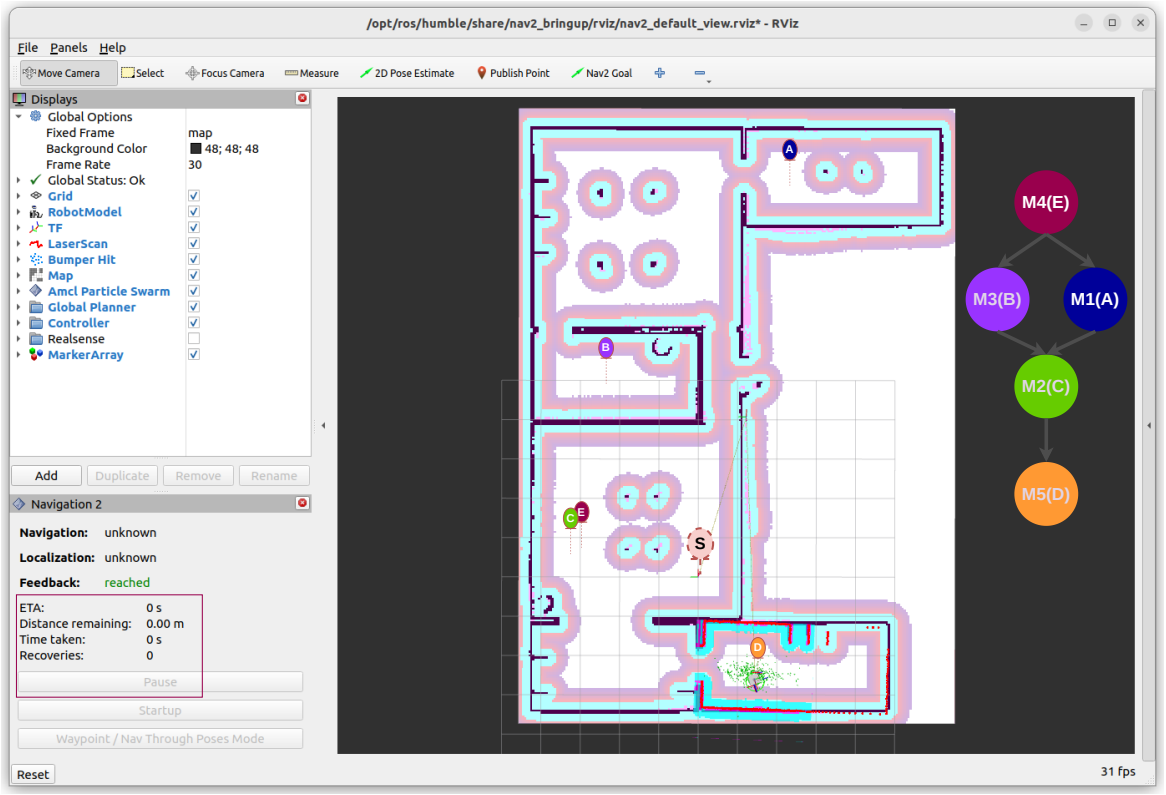


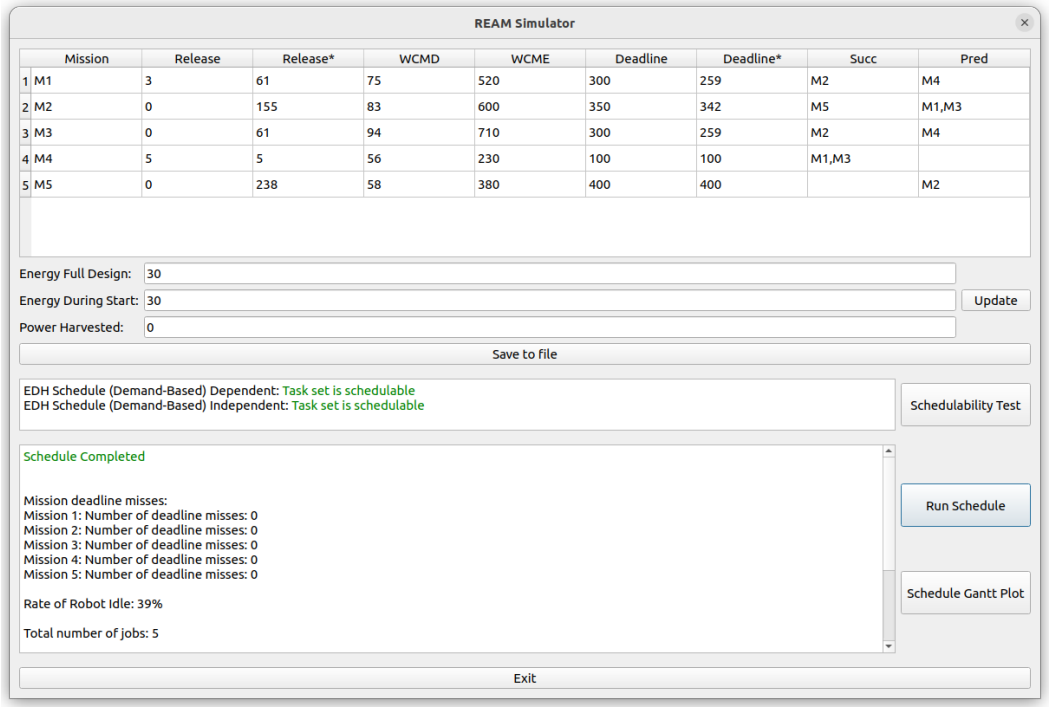
Figure 7.10 Rviz visualization of the map, robot, and planner along with the missions. 'S' is the initial starting point.

We perform the mapping using the cartographer slam and save the map using the map server packages of nav2, which is subsequently provided to the localization and planner. Figure 7.10 shows the map and the localized robot with the defined

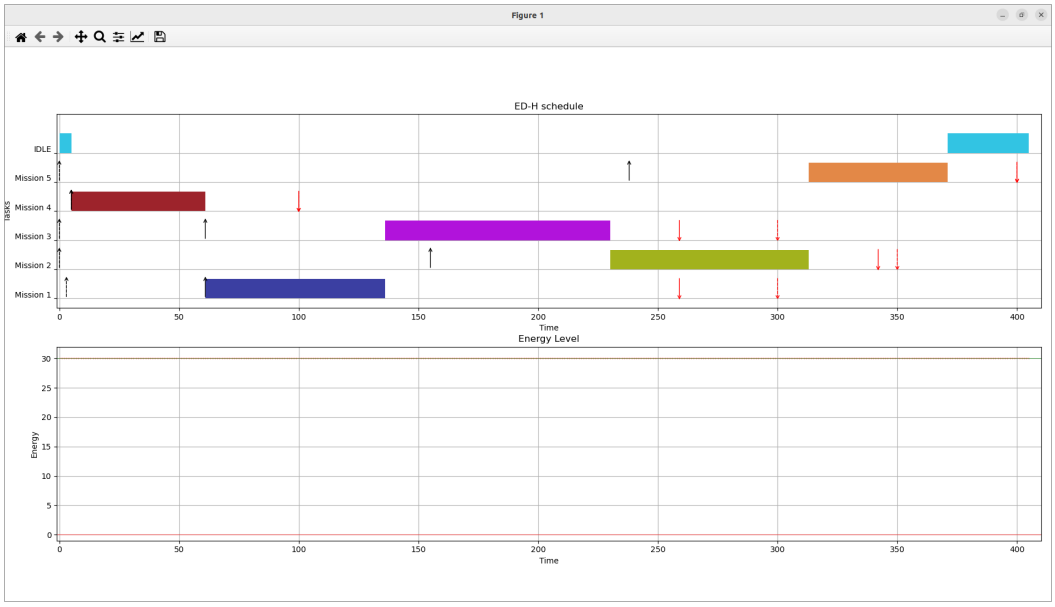
[9]http://wiki.ros.org/rqt_graph

mission points to which the robot must navigate. We specify a set of five missions, the precedence constraints of which are depicted in the figure. Initially, we launched each mission individually to measure the WCMD. As previously stated, the duration of the mission can be obtained through the feedback of the navigation stack, which is also represented in the Rviz window located at the bottom left, as highlighted in Figure 7.10. We include an upper bound with this mission duration, as occasionally the robot may be obstructed by obstacles due to localization issues or network loss. We do not take into account the mission duration when the robot is temporarily blocked and attempting recovery behavior, as it is considered lost. This issue is often observed when the robot is entering a narrow passage, such as passing through a doorway. We measure the WCME with the recorded velocity during the navigation of the mission, and utilizing the energy model. We operate under the assumption that the robot must complete a mission within a certain time frame, defining release times and deadlines accordingly.

After the definition of the mission parameters, we use the REAMSim tool to generate an independent mission set, with release times and deadlines calculated using a sorting algorithm that considers the precedence constraint. Figure 7.11a presents the parameters of the mission set used for the simulation. We simulate the harvesting power for the mission time window, randomly ranging from a minimum of 9W to a maximum of 12W. REAMSim is modified to consider this simulated variable harvested power if the 'Power Harvested' data is set to zero. We conduct an offline schedulability test for the mission set, which yields successful results. Figure 7.11b illustrates the schedule of the mission set using the ED-H energy-aware mission scheduler. It is observed that the power harvested is fully utilized by the mission and the battery's energy is never depleted. However, this scenario cannot be replicated in the robot simulation or real-world application due to the fact that REAMSim only considers the energy consumed during the execution and not in the idle state. To explore this further, we simulate the robot under different harvesting settings for the mission set.

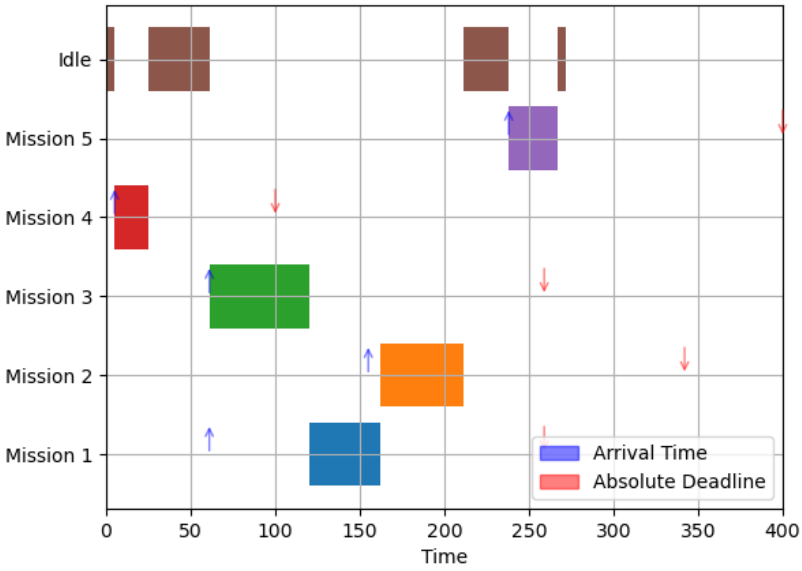


(a) Mission set used for the simulation. Power Harvested is set to 0 for considering variable harvested power.

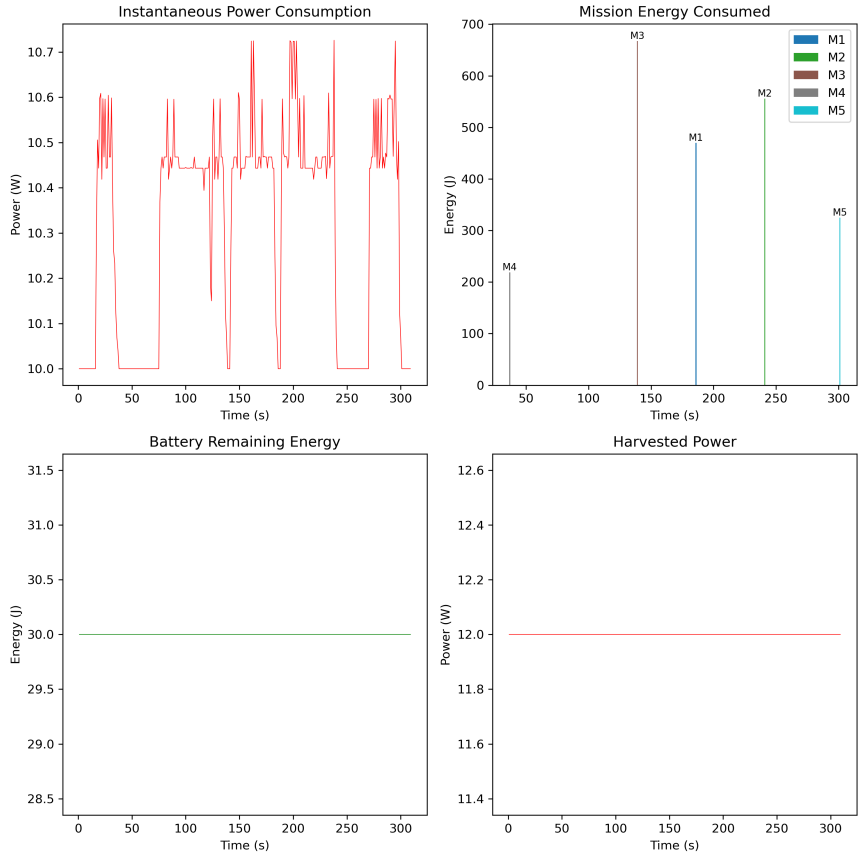


(b) Schedule of the mission.

Figure 7.11 Offline simulation of the mission set using REAMSim.

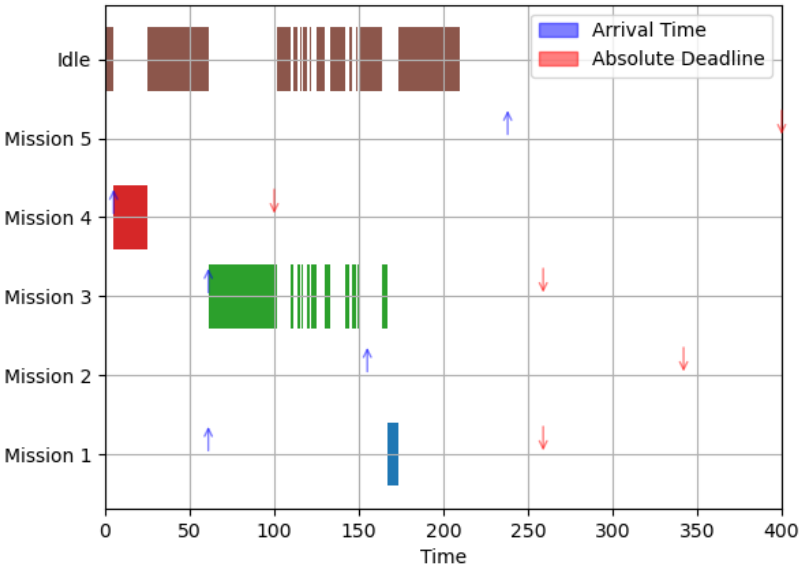


(a) Mission schedule with constant harvest power.

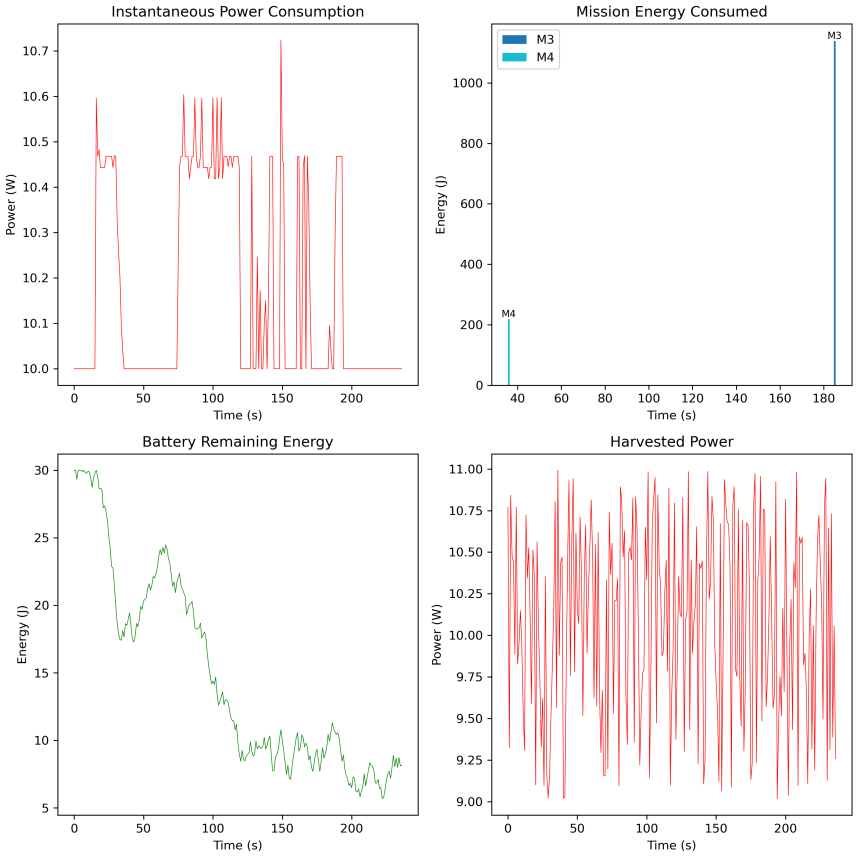


(b) Energy and power logs of the Mission schedule simulated on ROS2.

Figure 7.12 Mission schedule and energy logs for the setting 1.

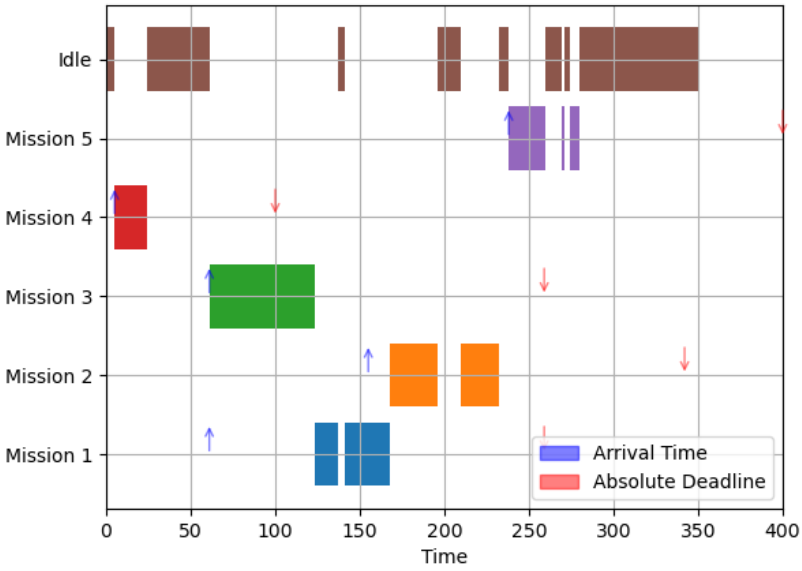


(a) Mission schedule.

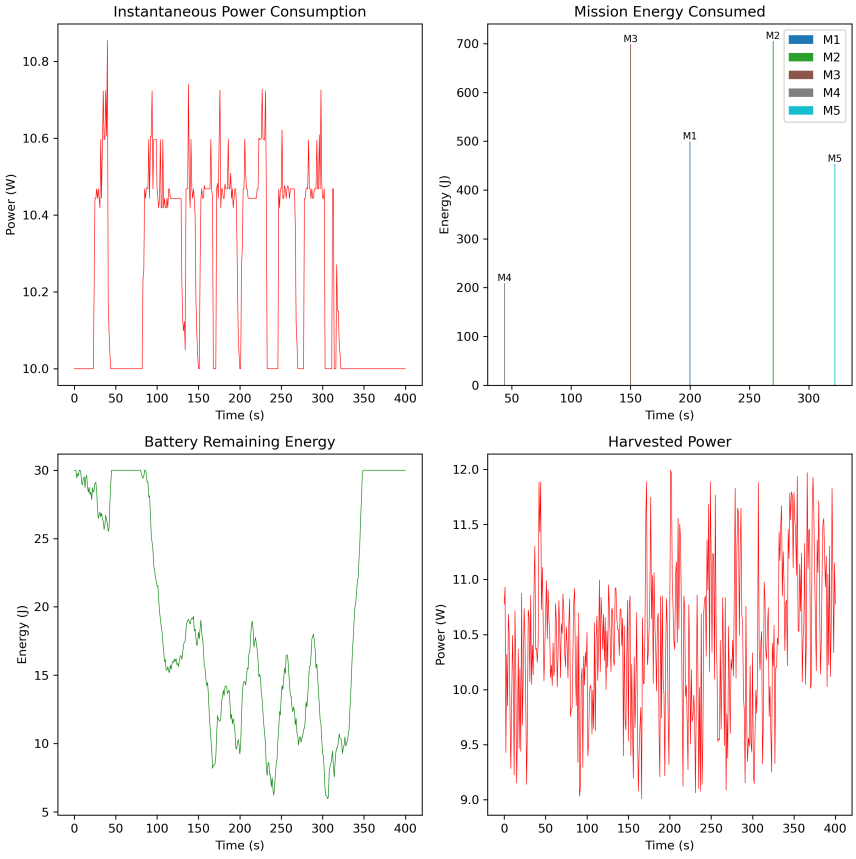


(b) Energy logs.

Figure 7.13 Mission schedule and energy log for simulation with setting 2.



(a) Mission schedule.



(b) Energy logs.

Figure 7.14 Mission schedule and energy log for simulation with setting 3.

Setting 1: (Constant) Harvested Power >= Total energy utilization

In this scenario, we consider a constant harvested power of 12W, which is either greater than or equal to the total energy utilization (≈11W). We then perform the simulation under this setting. We initialize a mission class with the independent mission set, which includes release time*, deadline*, WCMD, WCME, and goal point, to a runqueue. The mission is selected from the runqueue according to the EDF rule. Figure 7.12a illustrates the schedule as recorded in the mission manager performing the mission execution with the ED-H mission scheduler and navigation commander. Figure 7.12b shows the power and energy log of the simulation. The instantaneous power consumption plot exhibits the power consumed during navigation. The total energy consumed during the mission is presented in the top right corner. The subplot at the bottom left displays the remaining battery energy in joules, and the subplot at the bottom right plots the harvested power, which is constant. From the schedule, we can infer that the robot has utilized the harvested power to execute the missions, thus preventing any depletion in the battery energy. Furthermore, we can note that the precedence constraints are satisfied, leading to the conclusion that the scheduler is capable of executing critical missions with time constraints. Nevertheless, we cannot make the assumption that the power is always constant. Therefore, we move to the next setting, which incorporates variable power.

Setting 2: Variable Harvested Power Profile 1

In this scenario, we assume that the power harvested during the simulation is variable, ranging from 9W to 11W. As depicted in Figure 7.13, while the robot is executing Mission 3 and the energy falls below the threshold, the ED-H scheduler efficiently allocates an idle time period by effectively pausing the execution of the mission. This ensures there is enough energy to successfully complete the mission. However, energy surpasses the threshold during the performance of Mission 1. If we hypothesize that the harvesting unit fails to gather the required power and reaches a timeout, the mission is consequently skipped. This causes subsequent missions to experience time and energy starvation. In response to this situation, we propose a "breakout" period. During this time, the robot can maneuver towards high-energy zones (that can be defined by the Costmap filter[10]), enabling more power harvesting and the completion of subsequent missions. However the mission will not be completed before the deadline which can be

[10]https://navigation.ros.org/tutorials/docs/navigation2_with_keepout_filter.html

denoted using the QoS profile. The QoS measure provides an assessment of the robot's performance under the constraints of time and energy for mission completion.

Setting 3: Variable Harvested Power Profile 2

In this scenario, we conduct a simulation of mission scheduling using the same power harvesting profiles previously employed with the REAMSim tool for schedulability tests. The power ranges from 9W to 12W. Figure 7.14 presents the schedule and the energy log. We can observe that the missions are scheduled in accordance with the precedence constraints and are completed before their respective deadlines. The total energy used for the mission is represented in the plot, including the idle time during the mission processes. The battery's energy level never falls below the threshold and is replenished, indicating an energy-neutral operation. However, achieving this outcome remains a challenge due to technological limitations associated with the harvesting and charging system. Hence, a practical implementation of energy-neutral operation still presents difficulties. Nevertheless, we have developed a mission scheduler that is capable of handling time, energy and precedence constraints effectively. This scheduler could be further improved by incorporating a variety of harvesting technologies, paving the way towards realizing energy-neutral operation.

7.6 Implementation and Technological Barriers

Our research aimed to develop a real-time energy harvesting mobile robot equipped with an energy-aware mission scheduler and a harvesting unit, as depicted in Figure 7.2. Regrettably, we encountered certain challenges that hindered further progression. One issue was getting the navigation stack (nav2) to function with the new cobotic architecture. Moreover, a significant challenge lay in devising an indoor harvesting system capable of fulfilling the energy demands of the cobot. Considering the current state of technology, achieving such a system remains an elusive goal. In our initial tests, we opted to use readily available photovoltaic cells made of monocrystalline material[11]. As seen in Figure 7.15, we fabricated three panels, each consisting of 40 cells, capable of powering a load with a 12V battery. To regulate the power distribution between the battery and the harvesting unit, we employed a charge controller. We used a Yocto wattmeter to record the harvested power and, as expected, we were able to harvest only maximum power of 320mW in an indoor environment with a light intensity of 1700-1800 lux. For instance, over 2 hours and 30 minutes, we only harvested a total

[11] https://www.seeedstudio.com/0-5W-Solar-Panel-55x70.html

of 1500 joules indoors. On the other hand, outdoor settings with 65K lux allowed us to harvest up to 24W, with a total of 50K joules harvested over 1 hour and 30 minutes. Considering the available space on the cobot, we determined that it could accommodate four panels, each measuring 700mm x 700mm. Theoretically, these panels could generate approximately 2W of power indoors, which falls short of the power requirements for the cobotic system. As a result, we need to explore different technologies for indoor environments. While outdoor settings could potentially meet the harvesting requirements, they introduce further complications, such as the need for more complex navigation systems, making them less ideal for cobot applications.

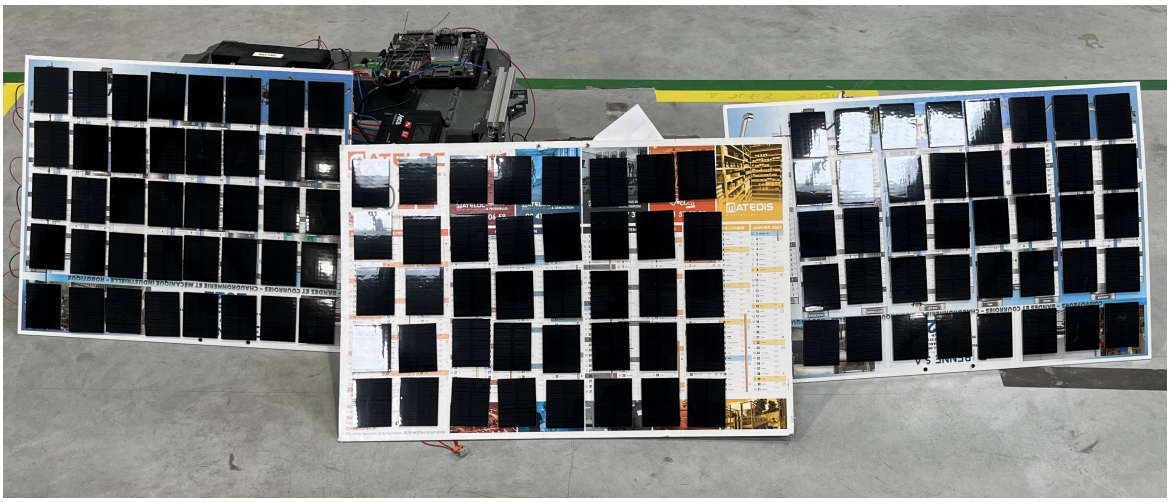


Figure 7.15 Preliminary test of indoor harvesting.

Despite this challenge, it is crucial to highlight that our proposed mission scheduler operates on an event-driven basis with time and precedence constraints suitable for critical mission scheduling. This attribute potentially allows it to integrate with other data-driven mission managers, as suggested in [Dinh-Khanh (2020)], to facilitate energy-efficient scheduling, while still depending on conventional charging stations. Looking forward, we expect advancements in harvesting technology and the evolution of innovative energy scavenging methods for indoor settings to lead to more efficient energy-neutral operations for cobotic systems. Current technological barriers present significant challenges, but we believe that progress in this field will help us overcome these obstacles, leading to more sustainable and energy-efficient cobotic systems. Additionally, the robot can incorporate energy consumption minimizing techniques, as outlined in current state-of-the-art approaches, to reduce energy demand. For instance, when the robot is idle, power to the motors can be regulated using switches to toggle 'ON' and 'OFF', thereby reducing power consumption. The energy model can also be

studied in depth to identify the optimal velocity profile for the robot, allowing it to navigate while consuming less power. Such techniques could contribute significantly to reducing the overall energy demand of the device. Consequently, this would allow us to focus on a harvesting unit with lower power output that can still meet the robot's energy demand.

7.7 Conclusion

In this chapter, we highlighted the limitations of using an energy-aware scheduler at the operating system level (i.e. at the computing task level) for industrial mobile robots. We proposed investigating the scheduler at the application level, specifically as a mission scheduler that takes into account the energy constraints of the cobotic device as a whole. We explored how concepts from real-time theory could be applied to mobile robots, and provided simple solutions for calculating the WCMD and WCME as part of mission characterization. We introduced a new concept for mission characterization that includes precedence constraints and makes use of a deadline sorting algorithm. This approach prompts us to think beyond priority triggers and consider more sophisticated deadline constraints for intelligent mission management in industrial settings. Next, we detailed the energy model used to estimate the robot's energy consumption and validated it using real robot results. We demonstrated the application of the EWMA prediction algorithm to robotic missions through a simulation test.

Following that, we described the modified ED-H scheduler designed for mission scheduling and introduced REAMSim, a tool for testing the schedulability of a given mission set offline. To showcase the performance of the scheduler under different power harvesting profiles, we used the ROS simulation environment and the turtlebot3 model. We also introduced the notion of Quality of Service (QoS) that accounts for the performance of the robot in terms of mission completion. Our proposed approach proved effective in scheduling critical missions with time, energy, and precedence constraints. However, implementing this methodology on real robotic devices for indoor settings poses challenges due to technological barriers. We discussed these limitations and showed preliminary results from testing indoor energy harvesting using monocrystalline photovoltaic cells. Although achieving energy-neutral operation currently presents technological hurdles, our work lays the groundwork for this goal. Meanwhile, the energy-aware mission scheduler can still prove beneficial for scheduling critical missions while relying on conventional charging stations.

Chapter 8

Conclusion and Future Perspectives

8.1 Conclusion

In this dissertation, we primarily focused on two significant limitations prevalent in the realm of industrial mobile cobots: the absence of *deterministic execution* and the need for effective *energy management*. Deterministic execution refers to the system's ability to consistently and predictably perform tasks, particularly in terms of task execution time and response times to events.

Our aim is to bring these aspects by applying insights derived from the field of real-time computing systems. Our objective for this dissertation can be summarized as follows:

1. Our first aim is to design a system architecture, both hardware and software, that adheres to real-time properties. We strive to offer an easy-to-use design that overcomes the dynamic configuration and programming limitations of current architectures.

2. Our principal objective is to make the system architecture energy-aware. We aim to develop a platform that facilitates *energy-neutral* operation. For mobile robots, energy neutrality implies a state where the robot's energy usage is perfectly offset by the energy it harvests or generates from environmental sources. This ensures seamless operation without exhausting its energy source, and in turn, improves its performance, extends its life expectancy, and increases the chances of mission accomplishment.

Objective 1 - Contribution

Our objective was to enhance the architectural design of cobotic devices, ensuring functional correctness, completeness, and appropriateness—these are key aspects of product quality as per ISO standard. The architecture we developed successfully meets both functional (FR) and non-functional (NFR) requirements of the mobile robotic device. We have proposed and implemented a unified hardware abstraction layer. This supports easy interfacing with the robot's peripheral components and complies with the software abstraction layer to satisfy the NFR properties. In the

software abstraction layer, we incorporated elements that ensure real-time capabilities. At the system level, we utilized a Linux kernel patched with the Xenomai kernel. This supports the deterministic execution of threads and improves scheduling latency. Furthermore, we applied deterministic execution to ROS threads, using the PiCAS framework to execute these threads under the Xenomai kernel. To further these advances, we have developed a detailed methodology. This methodology serves to assist developers in building and refining a robust mobile architecture that adheres to real-time properties while satisfying both FR and NFR of the device. The quality of the proposed architecture aligns with ISO standards, encompassing aspects such as modularity, reliability, reusability, maintainability, and portability.

Improvement Opportunities

Despite the proposed architecture adhering to the requirements, we have not been able to fully validate its functionality on an actual robot. We have conducted tests on peripheral components, providing assurance that all necessary parts will function as expected. However, we were unable to assess the complete functionality of the architecture within the robotic system, an aspect that requires further engineering effort to validate. There also exists a challenge in building and flashing the processing platform. We have attempted to simplify this process, but it could be further streamlined by utilizing Yocto[1] support for the Jetson modules. A significant limitation we encountered was component shortage. We employed the Jetson AGX Xavier module, the availability of which is currently limited. To the best of our knowledge, we were unable to identify a module that inherently satisfies the peripheral requirements of the cobotic device apart from the Jetson AGX modules. If we were to consider using the Jetson AGX Orin, it would necessitate modification of the kernel package to incorporate the Xenomai kernel. This would be a time-consuming engineering effort. Nonetheless, it is crucial to mention these aspects are not limitations, but rather opportunities for further development and improvement of the proposed architecture.

Objective 2 - Contribution

The principal aim of our research was centered around robotic operations with a focus on timing behavior and resource usage, essential elements contributing to performance efficiency. We addressed this aim by incorporating the concept of energy-neutrality drawn from real-time computing systems into our methodology. For the realization

[1] https://github.com/OE4T/meta-tegra

of this objective, we selected the ED-H scheduler, an optimal energy-aware real-time scheduler. However, translating this theoretical aspect into practical application presented a complex challenge. Our initial contribution was the development of a novel theorem for the ED-H scheduler that takes into account tasks with shared resource constraints. We provided a theoretical proof demonstrating that real-time tasks with constraints on time, energy, and shared resources can be scheduled using the optimal ED-H scheduler. Subsequently, we worked on the development of additional scheduling policies for the Xenomai kernel. The `SCHED_EDF` policy was integrated first to facilitate task scheduling with the earliest deadline first scheduler. Following the verification of its functionality, we proceeded to develop an energy-aware scheduler for Xenomai, resulting in a new scheduling class, `SCHED_ED-H`, which schedules tasks in accordance with the ED-H scheduler. However, due to the complexity of debugging, a full validation of the scheduler's performance at the system level was not possible. Furthermore, we found that the granularity of energy at the system level for the mobile cobot was significantly lower compared to the energy usage of the device as a whole. This led us to shift our focus from system-level concepts of real-time computing theory to the application level for the mobile robot. At the application level, we introduced a mission manager that incorporates the energy-aware ED-H scheduler for mission scheduling, and we proposed a novel concept of mission characterization that considers precedence constraints. Finally, we created an energy-aware mission scheduler that adheres to the time, energy, and precedence constraints of the mission for mobile robotic devices. Yet, the implementation of energy-neutral operation—an efficacy target of this objective—remained elusive due to technological barriers related to energy harvesting in indoor settings. Despite these challenges, our efforts contribute to a novel landscape for mobile robots by implementing real-time computing concepts.

Improvement Opportunities

Despite the contributions of our energy-aware mission scheduler, there are areas that require further attention: it can only consider static missions, meaning the missions must be created prior to the start of execution. The system does not permit the dynamic inclusion of missions during the schedule's operation with the energy-aware mission manager. The main shortcoming, however, lies in the harvesting technology currently available, which impedes the practical application of our methodology in indoor settings. Consequently, there is a distinct need for additional research in diverse fields to effectively incorporate various harvesting methods and deliver a robust prediction algorithm for the power harvested. This multi-disciplinary approach will strengthen

the applicability and efficiency of energy-neutral operations in indoor robotic systems.

In essence, this dissertation explored solutions for industrial mobile cobots that is that achieve both determinism and energy management, leveraging the concepts of real-time computing systems.

8.2 Scope for Future Research

Our research has successfully navigated several areas of development and has proposed solutions for industrial mobile cobots. However, there are additional opportunities for enhancement and expansion that can be realized through multidisciplinary integration. Furthermore, several intuitive aspects could drive advancements and innovation in the field of industrial mobile cobots.

- An area of focus could be the benchmarking of software abstraction using tracing tools, particularly for ROS2 with ros2_tracing [Bedard et al. (2022)], aiming to create a low-overhead framework.
- Dynamic power management techniques could be employed to control the state of the motor, fostering energy conservation during the robot's rest period.
- The exploration of harvesting techniques, such as regenerative energy through the use of efficient motors and recovery systems, presents an intriguing scope for research.
- Another compelling avenue for research would be to concentrate on *fixed priority energy-aware scheduling* [Chetto (2021)].
- Finally, our work could extend towards a multi-mission manager and expanding the energy-aware scheduler from single robot scheduling to multiple robots by enhancing the scheduler with bin packing heuristics [Shin and Ramanathan (1994), Coffman et al. (1999), Ghor et al. (2018)].

Each of these potential advancements underscores the multi-disciplinary nature of this research and highlights the significant opportunities that exist for continued innovation in the field of industrial mobile cobots.

8.3 Disseminations of this work

The following is the list of publications that have resulted from the research conducted during this thesis.

International Conferences

- M. I. M. Abdulla, M. Chetto, A. Queudet and L. Belouaer, "On designing cyber-physical-social systems with energy-neutrality and real-time capabilities,"2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), Victoria, BC, Canada, 2021, pp. 369-374, doi: 10.1109/ICPS49255.2021.9468260.
- M. I. Mohamed Abdulla, A. Queudet, M. Chetto and L. Belouaer, "Real-time Resource Management in Smart Energy-Harvesting Systems,"2022 IEEE Symposium on Computers and Communications (ISCC), Rhodes, Greece, 2022, pp. 1-7, doi: 10.1109/ISCC55528.2022.9912792.
- M. I. M. Abdulla, M. Chetto, A. Queudet "Real-time Scheduling and Resource Management for Energy Autonomous Sensors," IFAC World Congress 2023 ,Yokohama and Fukuoka, Japan, July 2023.

Chapter 9

French Summary

9.1 Introduction

Cette thèse explore le rôle de l'ingénierie logicielle dans la robotique, avec une attention particulière à la gestion de l'énergie pour les robots autonomes mobiles. Les systèmes robotiques, de plus en plus présents dans notre société, nécessitent une intégration parfaite des éléments matériels et logiciels. Des recherches, comme celles d'Ahmad and Babar (2016), montrent une tendance croissante à utiliser des méthodes d'ingénierie logicielle pour réduire la complexité et améliorer l'efficacité des systèmes robotiques. Toutefois, les pratiques actuelles d'ingénierie logicielle conduisent souvent à des logiciels susceptibles d'être erronés et difficiles à maintenir. De plus, l'étude de García et al. (2020) a révélé une lacune importante chez les professionnels de l'industrie en ce qui concerne la garantie d'un fonctionnement déterministe, un concept clé dans les systèmes informatiques temps réel. Cette thèse vise à combler ces lacunes, en se concentrant en particulier sur les robots mobiles. L'objectif principal est de traiter la question de la gestion de l'énergie, un enjeu majeur pour ces robots. L'objectif est d'améliorer leur efficacité opérationnelle et d'assurer l'exécution de leurs fonctions de manière autonome, tout en respectant les normes ISO 25010.

9.2 Problem Statement

L'étude de l'architecture des robots mobiles industriels est au cœur de cette thèse, en particulier l'exécution déterministe et la gestion des contraintes de temps et d'énergie. Les exigences des robots mobiles sont classifiées comme fonctionnelles et non fonctionnelles. Les exigences non-fonctionnelles englobent la performance déterministe, la fiabilité et l'utilisation de l'énergie. En raison de l'importance des réponses en temps réel face à des environnements dynamiques, des stratégies pour optimiser l'utilisation de l'énergie sans compromettre les performances temps réel sont explorées. Les algorithmes d'ordonnancement économes en énergie pour les robots mobiles sont examinés, ainsi que les compromis entre performance, fiabilité et consommation d'énergie. Les défis principaux comprennent le développement d'une architecture pour les robots mobiles qui respecte à la fois les exigences fonctionnelles et non fonctionnelles, l'amélioration des

algorithmes d'ordonnancement économes en énergie pour les systèmes de récupération d'énergie en temps réel (Real-time Energy Harvesting) et l'implémentation de ces algorithmes dans un système d'exploitation temps réel (RTOS). L'objectif ultime est de favoriser le développement d'une architecture robotique efficace, fiable et économe en énergie. Bien que le focus soit mis sur le robot mobile industriel HUSKY développé par E-COBOT, les contributions pourraient être étendues à d'autres systèmes robotiques avec les modifications appropriées.

1. *Conception de l'architecture matérielle/logicielle pour un robot mobile avec des aptitudes temps réel* : Le défi est de concevoir une architecture matérielle et logicielle qui peut gérer les performances temps réel, la charge de calcul, et la compatibilité avec le système d'exploitation et les middlewares sélectionnés. L'utilisation d'un système d'exploitation temps réel (RTOS) qui permet une exécution déterministe et un ordonnancement efficace des tâches est essentielle.

2. *Reconfigurabilité et reprogrammation dynamique pour une meilleure adaptabilité selon les exigences de l'utilisateur* : Il est crucial de permettre la reconfigurabilité et la reprogrammation dynamique, permettant des changements en temps réel dans les tâches du robot sans nécessiter un arrêt complet ou un accès physique à l'appareil.

3. *Intégration d'un algorithme d'ordonnancement sensible à l'énergie avec des contraintes de ressources partagées* : Le défi est de gérer les contraintes d'énergie dans l'architecture des robots mobiles. Cela inclut l'intégration d'algorithmes d'ordonnancement sensibles à l'énergie qui permettent un fonctionnement énergétiquement neutre, équilibrant la demande d'énergie avec l'énergie récupérée à partir de sources renouvelables.

4. *Implémentation de l'algorithme d'ordonnancement optimal ED-H économe en énergie dans un RTOS* : Il est difficile d'intégrer ces algorithmes dans un RTOS conjointement avec les politiques et mécanismes d'ordonnancement existants. Il est crucial de s'assurer que l'algorithme d'ordonnancement sensible à l'énergie peut coexister avec d'autres politiques d'ordonnancement sans impacter négativement la performance en temps réel du système.

5. *Implémentation de l'algorithme d'ordonnancement sensible à l'énergie pour assurer que les applications de mission des robots mobiles opèrent sous contraintes d'énergie tout en maintenant les performances et la fiabilité* : Le défi est de gérer efficacement les ressources énergétiques pour permettre l'achèvement réussi des

missions, en particulier à mesure que la complexité et le nombre de missions augmentent.

6. *Estimation de la durée et de la consommation d'énergie des missions des robots mobiles* : Il est difficile d'harmoniser la planification intelligente des missions en temps réel pour les robots mobiles en raison de plusieurs facteurs interdépendants, tels que la détermination de la consommation d'énergie pendant une mission et l'intégration de solutions de récupération d'énergie.

Cette thèse fournit un guide complet pour concevoir et intégrer un robot mobile déterministe temps réel et neutre en énergie, répondant aux défis de l'ordonnancement des missions, de la récolte d'énergie et des contraintes de ressources. Elle présente une analyse détaillée de chaque défi, des solutions proposées et leur efficacité, soutenues par des résultats expérimentaux, visant à améliorer les exigences fonctionnelles et non fonctionnelles des robots mobiles.

9.3 Contributions

Pour relever les défis mentionnés, nous avons apporté deux contributions majeures dans le domaine des robots mobiles industriels, et deux contributions significatives aux systèmes informatiques temps réel. Les contributions sont résumées comme suit:

9.3.1 Contribution 1

Conception Matérielle et Logicielle d'une Architecture de Robot Mobile Industriel avec Aptitudes Temps Réel : Il s'agit d'une contribution au niveau de l'architecture et du système. Notre objectif était d'améliorer la conception architecturale des dispositifs cobotiques, en garantissant l'exactitude, l'exhaustivité et la pertinence fonctionnelles, qui sont des aspects clés de la qualité du produit selon la norme ISO. L'architecture que nous avons développée répond avec succès aux exigences fonctionnelles (FR) et non fonctionnelles (NFR) du dispositif robotique mobile. Nous avons proposé et mis en œuvre une couche d'abstraction matérielle unifiée. Cela facilite l'interface avec les composants périphériques du robot et respecte la couche d'abstraction logicielle pour satisfaire les propriétés NFR. Dans la couche d'abstraction logicielle, nous avons incorporé des éléments qui garantissent les capacités temps réel. Au niveau du système, nous avons utilisé un noyau Linux patché avec le noyau Xenomai. Cela favorise l'exécution déterministe des threads et améliore la latence

de l'ordonnancement. L'exécution déterministe se réfère à la capacité du système à effectuer des tâches de manière constante et prévisible, en particulier en termes de temps d'exécution des tâches et de temps de réponse aux événements. De plus, nous avons appliqué l'exécution déterministe aux threads ROS, en utilisant l'environnement PiCAS pour exécuter ces threads sous le noyau Xenomai. Pour approfondir ces avancées, nous avons développé une méthodologie détaillée. Cette méthodologie vise à aider les développeurs à construire et affiner une architecture mobile robuste qui respecte les propriétés temps réel tout en satisfaisant les FR et NFR du dispositif. La qualité de l'architecture proposée est conforme aux normes ISO, englobant des aspects tels que la modularité, la fiabilité, la réutilisabilité, la maintenabilité et la portabilité.

Background

Avant d'aborder les autres contributions, il est essentiel de présenter le contexte de l'algorithme d'ordonnancement sensible à l'énergie, optimal, ED-H pour les systèmes à récupération d'énergie temps réel (RTEHS). Habituellement, un RTEHS comprend un module de récolte d'énergie, un module de stockage d'énergie et un module processeur. Dans le système d'exploitation, les tâches à effectuer sont organisées selon un schéma d'ordonnancement défini. Un de ces schémas traditionnels est "Earliest Deadline First" (EDF). Cependant, EDF ne prend pas en compte les contraintes énergétiques. Selon Chetto and Queudet (2014b), EDF n'est plus optimal et a un facteur de compétitivité nul dans le cadre de RTEH. Chetto (2014) a adapté l'algorithme EDF pour démontrer l'optimalité de ED-H, un ordonnanceur oisif avec lookahead D, où D est l'échéance relative la plus longue des tâches dans l'application. Comme EDF, l'ordonnanceur ED-H prend des décisions en temps réel. Cependant, il implique une clairvoyance sur le futur à chaque instant sur au moins D unités de temps, et il est oisif. Dans une plateforme RTEH, l'énergie disponible ne permet parfois pas d'exécuter toutes les tâches à temps, sauf si le processeur entre délibérément dans le mode sommeil. Lorsque l'ordonnanceur vise uniquement à respecter les échéances des tâches, il n'y a aucun avantage à terminer une tâche plus tard que nécessaire. Les capacités à rendre le processeur inactif font que ED-H peut anticiper l'épuisement de l'énergie dans l'unité de stockage, ce qui, avec l'ordonnanceur non oisif et non clairvoyant EDF, peut se manifester par le non-respect des échéances.

Avant de décrire les principes de ED-H, voici quelques définitions nécessaires :

- Le "Slack Time" de l'ensemble de tâches J à t_c, noté $ST_{J(t_c)}$, représente la durée la plus longue pendant laquelle le processeur peut rester inactif à partir de t_c.

- Le 'Preemption Slack Energy" de l'ensemble de tâches J à t_c, notée $PSE_J(t_c)$, représente la plus grande quantité d'énergie qui peut être consommée par la tâche active à t_c tout en préservant la faisabilité énergétique des tâches qui peuvent la préempter. $PSE_J(t_c) = \min_{t_c<r_i<d_i<d} SE_{J_i}(t_c)$.

- La "Slack Energy" de la tâche J_i à t_c, notée $SE_{J_i}(t_c)$, est la plus grande quantité d'énergie qui peut être consommée dans $[t_c, d_i)$ tout en garantissant assez d'énergie pour les tâches lancées à ou après t_c avec une échéance à ou avant d_i. $SE_{J_i}(t_c) = E(t_c) + E_p(t_c, d_i) - g(t_c, d_i)$ où $g(t_c, d_i)$ est la demande d'énergie de J sur $[t_c, d_i)$.

Désignons par $Q_r(t_c)$ la liste des tâches prêtes à être exécutées à t_c et non terminées. Voici les règles qui définissent l'algorithme d'ordonnancement ED-H :

Rule1: La prochaine tâche à exécuter dans $Q_r(t_c) = \emptyset$, est choisie selon l'ordre de priorité EDF.

Rule2: Si $Q_r(t_c) \neq \emptyset$ est vide, alors le processeur reste inactif dans $[t_c, t_c + 1)$.

Rule3: Si $Q_r(t_c) \neq \emptyset$ n'est pas vide alors que $E(t_c) = 0$ ou $PSE_J(t_c) = 0$, alors le processeur reste inactif dans $[t_c, t_c + 1)$.

Rule4: Si $Q_r(t_c) \neq \emptyset$ n'est pas vide alors que $E(t_c) = C$ ou $ST_J(t_c) = 0$, alors le processeur reste occupé dans $[t_c, t_c + 1)$.

Rule5: Si $Q_r(t_c) \neq \emptyset$ n'est pas vide, $0 < E(t_c) < C$, $ST_J(t_c) > 0$, et $PSE_{t_c} > 0$, le processeur peut rester inactif ou occupé dans $[t_c, t_c + 1)$ selon une règle prédéfinie.

ED-H est un ordonnanceur optimal, ce qui signifie qu'un ensemble de tâches est faisable si et seulement si au moins un ordonnancement ED-H peut satisfaire toutes les échéances, compte tenu de la capacité de stockage d'énergie et de la puissance de la source environnementale. La faisabilité de ce problème a été étudiée dans une publication précédente. Les limites de faisabilité peuvent être dues au temps de traitement et/ou à l'énergie. Ainsi, en plus de l'analyse de la demande de traitement utilisée classiquement dans les systèmes temps réel sans contraintes d'énergie, il faut prendre en compte l'analyse de la demande d'énergie. Cela nous amène à définir la pénurie de temps et la pénurie d'énergie :

1. La pénurie de temps se produit lorsque le temps processeur disponible pour traiter une tâche avant son échéance n'est pas suffisant, alors qu'il reste de l'énergie disponible lorsque la violation de l'échéance se produit.

2. La pénurie d'énergie se produit lorsque le temps processeur disponible pour traiter une tâche avant son échéance est suffisant, mais l'énergie est épuisée lorsque la violation de l'échéance se produit.

Ainsi, le test de faisabilité temporelle vérifie s'il y a pénurie de temps, tandis que le test de faisabilité énergétique vérifie s'il y a pénurie d'énergie, pour n'importe quel intervalle de temps de longueur finie.

Dans cette thèse, nous appliquons cette approche, prouvée de manière formelle, pour faire fonctionner le robot mobile industriel de manière neutre en énergie. Cette approche diffère des approches de pointe sur les économies d'énergie pour les robots mobiles, qui consistent en la planification de mouvements économes en énergie, la planification de trajets économes en énergie, le contrôle de la vitesse des articulations et la gestion de l'alimentation du processeur. Nous nous concentrons d'abord sur cette approche au niveau système du module de traitement. Voici les contributions en ce qui concerne la gestion de l'énergie dans les robots mobiles industriels.

9.3.2 Contribution 2

Algorithme d'ordonnancement temps réel sensible à l'énergie ED-H sous contraintes de ressources partagées : Nous avons étudié le test de faisabilité ED-H en considérant des tâches qui accèdent en exclusion mutuelle à des ressources partagées. Il est essentiel de bien gérer ces ressources pour éviter des situations de blocage. Nous avons créé un modèle supplémentaire pour organiser des tâches qui partagent des ressources. Une avancée clé est la nouvelle fonction de calcul de la quantité d'énergie liée au blocage, qui complète la fonction traditionnelle de calcul du temps de blocage. Nous avons souligné les limites du test de faisabilité actuel et avons proposé une nouvelle analyse qui intègre conjointement le temps et l'énergie. Enfin, nous avons présenté un nouveau théorème relatif au test d'ordonnançabilité de tâches accèdant à des ressources partagées sous l'ordonnancement ED-H.

9.3.3 Contribution 3

Mise en œuvre de l'algorithme d'ordonnancement ED-H sous Xenomai, un co-noyau temps réel pour distribution Linux: Nous avons approfondi l'étude du système d'ordonnancement de Xenomai afin d'y intégrer l'ordonnanceur sensible à l'énergie ED-H. Pour cela, nous avons créé REACTSim, un outil de simulation intégrant des aspects énergétiques, avec un guide d'utilisation. Nos tests ont confirmé

que ED-H permet un fonctionnement neutre en énergie. Nous avons ensuite détaillé les principes d'ordonnancement des noyaux Linux et Xenomai. Nous avons ajouté la politique `SCHED_EDF` à Xenomai, une étape nécessaire car ED-H est basée sur cette politique. Après validation, nous avons créé une nouvelle classe d'ordonnancement pour l'ordonnanceur ED-H, `SCHED_EDH`, adaptée aux tâches soumises à des contraintes de temps et d'énergie. Malgré les défis liés à la complexité de l'ordonnancement et à la caractérisation difficile des paramètres énergétiques, nous pensons que l'ordonnanceur ED-H, utilisé au niveau de l'application, peut améliorer significativement la gestion de l'énergie et l'efficacité opérationnelle des robots mobiles.

9.3.4 Contribution 4

Mise en œuvre de l'algorithme d'ordonnancement ED-H pour l'exploitation économe en énergie des robots mobiles: Nous avons exploré l'implémentation d'un ordonnanceur sensible à l'énergie, ED-H, pour des robots mobiles industriels, en mettant l'accent non au niveau du système d'exploitation, mais au niveau de l'application, c'est-à-dire au niveau de l'ordonnancement des missions. Pour cela, nous avons introduit des concepts tirés de la théorie de l'informatique temps réel, ainsi que des outils pour estimer les besoins énergétiques de chaque mission. L'idée est de réfléchir non seulement à la priorité des tâches, mais aussi aux échéances et aux contraintes d'énergie pour une gestion plus intelligente des missions. Nous avons également présenté notre modèle d'énergie, validé par des expériences réelles, ainsi qu'un outil de simulation pour tester la faisabilité de nos missions en tenant compte de l'énergie. Nous avons montré comment cet ordonnanceur peut être performant, même dans des scénarios variés de récolte d'énergie. Et nous avons introduit la notion de Qualité de Service (QoS) pour évaluer la performance de nos robots. Enfin, nous avons discuté des défis à relever pour mettre en œuvre cette méthode sur des robots réels, notamment en ce qui concerne la récolte d'énergie en environnement intérieur (indoor). Malgré ces défis, nous pensons que notre travail est une base solide pour l'intégration de systèmes neutres en énergie, et que l'ordonnanceur de missions sensible à l'énergie peut être utile pour gérer efficacement les missions en utilisant des stations de charge traditionnelles.

9.4 Conclusion

Nous avons relevé six défis pour réaliser un robot mobile industriel. Nous avons mis en place une méthodologie complète pour créer une architecture matérielle et

logicielle robuste pour un robot mobile, avec des aptitudes de fonctionnement temps réel indispensables pour régir les fonctionalités et la réactivité du système, et pour optimiser les performances et l'utilisation des ressources. Nous avons concrétisé cette méthodologie. Nous avons mis en évidence des obstacles technologiques liés aux méthodes de récupération d'énergie en intérieur. Néanmoins, nous envisageons la possibilité, dans le futur, de concevoir un robot industriel totalement autonome en évitant la présence d'une station de charge conventionnelle car ne dépendant que de l'énergie issue de sources ambiantes. La méthodologie guidera également le choix de la taille de l'unité de stockage et la taille de l'unité de récupération d'énergie conduisant ainsi à concevoir des robots mobiles à la fois robustes et économiques pour travailler aux côtés des humains dans l'industrie.

References

Abdallah, M. (2014). *Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie*. PhD thesis, University of Nantes (France).

Abdedda, Y., Chandarli, Y., Davis, R. I., and Masson, D. (2014). Schedulability Analysis for Fixed Priority Real-Time Systems with Energy-Harvesting. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 311–320, New York, NY, USA. Association for Computing Machinery.

Abraham, K. M. (2020). How Comparable Are Sodium-Ion Batteries to Lithium-Ion Counterparts? *ACS Energy Letters*, 5(11):3544–3547.

Ahmad, A. and Babar, M. A. (2016). Software architectures for robotic systems: A systematic mapping study. *Journal of Systems and Software*, 122:16–39.

Allavena, A. and Mosse, D. (2001). Scheduling of Frame-based Embedded Systems with Rechargeable Batteries. *Workshop on Power Management for Real-time and Embedded systems (in conjunction with RTAS)*, pages 1–8.

Amarnath, A., Pal, S., Kassa, H., Vega, A., Buyuktosunoglu, A., Franke, H., Wellman, J.-D., Dreslinski, R., and Bose, P. (2022). HetSched: Quality-of-Mission Aware Scheduling for Autonomous Vehicle SoCs.

Andreasson, H., Grisetti, G., Stoyanov, T., and Pretto, A. (2023). Software Architectures for Mobile Robots. *Encyclopedia of Robotics*, pages 1–11.

Aziz, H., Pal, A., Pourmiri, A., Ramezani, F., and Sims, B. (2022). Task Allocation Using a Team of Robots. *Current Robotics Reports*, 3(4):227–238.

Baker, T. P. (1990). A stack-based resource allocation policy for realtime processes. *Proceedings - Real-Time Systems Symposium*, pages 191–200.

Barkas, D. A., Psomopoulos, C. S., Papageorgas, P., Kalkanis, K., Piromalis, D., and Mouratidis, A. (2019). Sustainable energy harvesting through triboelectric nano – Generators: A review of current status and applications. *Energy Procedia*, 157:999–1010.

Baruah, S. and Burns, A. (2006). Sustainable scheduling analysis. *Proceedings - Real-Time Systems Symposium*, pages 159–168.

Baruah, S. K. (2006). Resource sharing in EDF-scheduled systems: A closer look. *Proceedings - Real-Time Systems Symposium*, pages 379–387.

Baruah, S. K., Mok, A. K., and Rosier, L. E. (1990). Preemptively scheduling hard-real-time sporadic tasks on one processor. *Proceedings - Real-Time Systems Symposium*, pages 182–190.

Barut, S., Boneberger, M., Mohammadi, P., and Steil, J. J. (2021). Benchmarking Real-Time Capabilities of ROS 2 and Orocos for Robotics Applications. *Proceedings - IEEE International Conference on Robotics and Automation*, 2021-May:708–714.

Bedard, C., Lutkebohle, I., and Dagenais, M. (2022). ros2_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2. *IEEE Robotics and Automation Letters*, 7(3):6511–6518.

Belsare, K., Rodriguez, A. C., Sánchez, P. G., Hierro, J., Kołcon, T., Lange, R., Lütkebohle, I., Malki, A., Losa, J. M., Melendez, F., Rodriguez, M. M., Nordmann, A., Staschulat, J., and von Mendel, J. (2023). *Micro-ROS*, pages 3–55. Springer International Publishing, Cham.

Bender, A., Whelan, B., and Sukkarieh, S. (2020). A high-resolution, multimodal data set for agricultural robotics: A Ladybird's-eye view of Brassica. *Journal of Field Robotics*, 37(1):73–96.

Benini, L., Bogliolo, A., and De Micheli, G. (2000). A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316.

Bini, E. and Buttazzo, G. C. (2005). Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154.

Brateman, J., Xian, C., and Lu, Y. H. (2006). Energy-efficient scheduling for autonomous mobile robots. *IFIP VLSI-SoIC 2006 - IFIP WG 10.5 International Conference on Very Large Scale Integration and System-on-Chip*, pages 361–366.

Brown, J. H. and Martin, B. (2010). How fast is fast enough? choosing between Xenomai and Linux for real-time applications. *proc. of the 12th Real-Time Linux Workshop (RTLWS'12)*, pages 1–17.

Bruyninckx, H. (2001). Open robot control software: The OROCOS project. *Proceedings - IEEE International Conference on Robotics and Automation*, 3:2523–2528.

Buchmann, I. (2017). *Batteries in a Portable World*. Cadex Electronics Inc.„ 4 edition.

Burd, T. D., Pering, T. A., Stratakos, A. J., and Brodersen, R. W. (2000). A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580.

Burns, A. (2009). *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*.

Buttazzo, G. C. (1997). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Second Edition. In *Real-Time Systems Series*.

Buttazzo, G. C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition. In *Real-Time Systems Series*.

Canfield, S. L., Hill, T. W., and Zuccaro, S. G. (2019). Prediction and Experimental Validation of Power Consumption of Skid-Steer Mobile Robots in Manufacturing Environments. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 94(3-4):825–839.

Casini, D., Blaß, T., Lütkebohle, I., and Brandenburg, B. B. (2019). Response-time analysis of ROS 2 processing chains under reservation-based scheduling. *Leibniz International Proceedings in Informatics, LIPIcs*, 133.

Chen, M. I. and Lin, K. J. (1990). Dynamic priority ceilings: A concurrency control protocol for real-time systems. *Real-Time Systems*, 2(4):325–346.

Cheramy, M., Hladik, P.-E., Deplanche, A.-M., and Dube, S. (2014). Simulation of real-time scheduling with various execution time models. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems*, pages 1–4.

Chetto, H., Silly, M., and Bouchentouf, T. (1990). Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194.

Chetto, M. (2014). Optimal scheduling for real-time jobs in energy harvesting computing systems. *IEEE Transactions on Emerging Topics in Computing*, 2(2):122–133.

Chetto, M. (2021). Dynamic power management for fixed priority real-time systems with regenerative energy. In *The 8th International Conference on Future Internet of Things and Cloud (FiCloud 2021)*, Virtual, Italy.

Chetto, M. and Osta, R. E. (2022). Real-Time Scheduling of DAG Tasks in Self-Powered Sensors with Scavenged Energy. *ELECOM 2022 - Proceedings of the 2022 4th IEEE International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering*, pages 1–6.

Chetto, M. and Queudet, A. (2014a). Clairvoyance and online scheduling in real-time energy harvesting systems. *Real-Time Systems*, 50(2):179–184.

Chetto, M. and Queudet, A. (2014b). A note on edf schedulingfor real-time energy harvesting systems. *IEEE Transactions on Computers*, 63(4):1037–1040.

Chetto, M. and Queudet, A. (2019). Feasibility analysis of periodic real-time systems with energy harvesting capabilities. *Sustainable Computing: Informatics and Systems*, 22:84–95.

Choi, B. W., Shin, D. G., Park, J. H., Yi, S. Y., and Gerald, S. (2009). Real-time control architecture using Xenomai for intelligent service robots in USN environments. *Intelligent Service Robotics*, 2(3):139–151.

Choi, H., Enright, D., Sobhani, H., Xiang, Y., and Kim, H. (2022). Priority-Driven Real-Time Scheduling in ROS 2: Potential and Challenges. *The 1st Real-time And intelliGent Edge computing workshop*, pages 28–31.

Choi, H., Xiang, Y., and Kim, H. (2021). PiCAS : New Design of Priority-Driven Chain-Aware Scheduling for ROS2. (October).

Coffman, E. G., Galambos, G., Martello, S., and Vigo, D. (1999). *Bin Packing Approximation Algorithms: Combinatorial Analysis*, pages 151–207. Springer US, Boston, MA.

Corbet, J., Rubini, A., and Kroah-Hartman, G. (2005). Data Types in the Kernel. *Linux Device Drivers*, pages 288–301.

Datouo, R., Motto, F. B., Zobo, B. E., Melingui, A., Bensekrane, I., and Merzouki, R. (2018). Optimal motion planning for minimizing energy consumption of wheeled mobile robots. *2017 IEEE International Conference on Robotics and Biomimetics, ROBIO 2017*, 2018-Janua:2179–2184.

De, M. and Arnaldo, C. (2010). The New Linux 'perf' Tools. *Linux Kongress*.

de Oliveira, D. B., Casini, D., de Oliveira, R. S., and Cucinotta, T. (2020). Demystifying the real-time linux scheduling latency. *Leibniz International Proceedings in Informatics, LIPIcs*, 165.

De Ryck, M., Versteyhe, M., and Debrouwere, F. (2020). Automated guided vehicle systems, state-of-the-art control algorithms and techniques. *Journal of Manufacturing Systems*, 54(December 2018):152–173.

Delgado, R., You, B. J., and Choi, B. W. (2019). Real-time control architecture based on Xenomai using ROS packages for a service robot. *Journal of Systems and Software*, 151:8–19.

Diana Marieska, M., Kistijantoro, A. I., and Subair, M. (2011). Analysis and benchmarking performance of Real Time Patch Linux and Xenomai in serving a real time application. *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, ICEEI 2011*, (July).

Dinh-Khanh, H. (2020). *QoS-Aware Resource and Energy Management for Autonomous Mobile Robotic Systems*. PhD thesis.

Eichler, C., Wägemann, P., and Schröder-Preikschat, W. (2019). Genee: A benchmark generator for static analysis tools of energy-constrained cyber-physical systems. *CPS-IoTBench 2019 - Proceedings of the 2019 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*, (April):1–6.

Elango, M., Nachiappan, S., and Tiwari, M. K. (2011). Balancing task allocation in multi-robot systems using K-means clustering and auction based mechanisms. *Expert Systems with Applications*, 38(6):6486–6491.

Elkady, A. and Sobh, T. (2012). Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *Journal of Robotics*, 2012:1–15.

Farahmand, M. Z., Nazari, M. E., Shamlou, S., and Shafie-Khah, M. (2021). The simultaneous impacts of seasonal weather and solar conditions on pv panels electrical characteristics. *Energies*, 14(4).

Farooq, M. U., Eizad, A., and Bae, H. K. (2023). Power solutions for autonomous mobile robots: A survey. *Robotics and Autonomous Systems*, 159:104285.

Farsi, M. and Ratcliff, K. (1997). An introduction to canopen and canopen communication issues. In *IEE Colloquium on CANopen Implementation (Digest No. 1997/384)*, pages 2/1–2/6.

Fong, D. Y. (2017). Wireless sensor networks. *Internet of Things and Data Analytics Handbook*, 16(17):197–213.

García, S., Strüber, D., Brugali, D., Berger, T., and Pelliccione, P. (2020). Robotics software engineering: A perspective from the service robotics domain. *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 593–604.

Gerkey, B. P., Vaughan, R. T., Støy, K., Howard, A., Sukhatme, G. S., and Matarić, M. J. (2001). Most valuable player: A robot device server for distributed control. *IEEE International Conference on Intelligent Robots and Systems*, 3:1226–1231.

Ghor, H. E., Chetto, M., and Osta, R. E. (2018). Multiprocessor real-time scheduling for wireless sensors powered by renewable energy sources. In *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications*, pages 1–6.

Gibson, T. L. and Kelly, N. A. (2010). Solar photovoltaic charging of lithium-ion batteries. *Journal of Power Sources*, 195(12):3928–3932.

Guerrero-González, A., García-Córdova, F., and Ruz-Vila, F. d. A. (2010). A solar powered autonomous mobile vehicle for monitoring and surveillance missions of long duration. *International Review of Electrical Engineering*, 5(4):1580–1587.

Gürgöze, G. and Türkoğlu, (2022). A novel energy consumption model for autonomous mobile robot. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30(1):216–232.

Gurguze, G. and Turkoglu, T. (2017). Energy Management Techniques in Mobile Robots. In *World Academy of Science, Engineering and Technology International Journal of Energy and Power Engineering*, volume 2017, pages 1085–1093.

Hamadani, B., Long, Y. S., Tsai, M. A., and Wu, T. C. (2021). Interlaboratory Comparison of Solar Cell Measurements under Low Indoor Lighting Conditions. *IEEE Journal of Photovoltaics*, 11(6):1430–1435.

Hanschke, L. (2022). Scheduling Dependent Tasks for Energy-Neutral Operation of Sensor Nodes. Technical report.

Henkel, C., Bubeck, A., and Xu, W. (2016). Energy Efficient Dynamic Window Approach for Local Path Planning in Mobile Service Robotics. *IFAC-PapersOnLine*, 49(15):32–37.

Hoora, S., Hyunjong, C., and Hyoseung, K. (2023). Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors. In *29th IEEE Real-Time and Embedded Technology and Applications Symposium.*

Huang, C.-C., Lin, C.-H., and Wu, C. (2015). Performance Evaluation of Xenomai 3.

Iigo-Blasco, P., Diaz-Del-Rio, F., Romero-Ternero, M. C., Cagigas-Muiz, D., and Vicente-Diaz, S. (2012). Robotics software frameworks for multi-agent robotic systems development. *Robotics and Autonomous Systems*, 60(6):803–821.

Jaiem, L., Druon, S., Lapierre, L., and Crestani, D. (2016). A step toward mobile robots autonomy: Energy estimation models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9716:177–188.

Jaramillo Morales, M. F., Dogru, S., Gomez-Mendoza, J. B., and Marques, L. (2020). Energy estimation for differential drive mobile robots on straight and rotational trajectories. *International Journal of Advanced Robotic Systems*, 17(2):1–12.

Jaramillo-Morales, M. F., Dogru, S., and Marques, L. (2020). Generation of Energy Optimal Speed Profiles for a Differential Drive Mobile Robot with Payload on Straight Trajectories. *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2020*, pages 136–141.

Jaramillo Morales, M. F., Gómez Mendoza, and Juan B., a. (2018). Mixed energy model for a differential guide mobile robot evaluated with straight and curvature paths. *ICINCO 2018 - Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics*, 2(Icinco):473–479.

Jayaseelan, R., Mitra, T., and Li, X. (2006). Estimating the worst-case energy consumption of embedded software. *Real-Time Technology and Applications - Proceedings*, pages 81–90.

Jeong, J. and Culler, D. (2012). A practical theory of micro-solar power sensor networks. *ACM Transactions on Sensor Networks*, 9(1).

Jo, Y. H. and Choi, B. W. (2022). Performance Evaluation of Real-time Linux for an Industrial Real-time Platform. 11(1):28–35.

Joseph, M., Pandya, P. K., and Los, U. M. D. E. C. D. E. (1986). Finding Response Times in a Real-Time System. *Comput. J.*, 29:390–395.

Kaleci, B. and Parlaktuna, O. (2013). Performance analysis of bid calculation methods in multirobot market-based task allocation. *Turkish Journal of Electrical Engineering and Computer Sciences*, 21(2):565–585.

Kansal, A., Hsu, J., Zahedi, S., and Srivastava, M. B. (2007). Power Management in Energy Harvesting Sensor Networks. *ACM Transactions on Embedded Computing Systems*, 6(4):32.

Khosro Pour, N., Krummenacher, F., and Kayal, M. (2013). Fully integrated solar energy harvester and sensor interface circuits for energy-efficient wireless sensing applications. *Journal of Low Power Electronics and Applications*, 3(1):9–26.

Kim, C. H. and Kim, B. K. (2007). Minimum-Energy Translational Trajectory Generation for Differential-Driven Wheeled Mobile Robots. *Journal of Intelligent and Robotic Systems*, 49(4):367–383.

Ko, S. T., Park, S. S., and Lee, J. H. (2019). Regenerative battery charging control method for PMSM drive without a DC/DC converter. *Electronics (Switzerland)*, 8(10).

Kortenkamp, D., Simmons, R., and Brugali, D. (2016). Robotic Systems Architectures and Programming. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 283–306. Springer International Publishing, Cham.

Koubaa, A. (2016). *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer Publishing Company, Incorporated, 1st edition.

Kyaw, P. T., Le, A. V., Veerajagadheswar, P., Elara, M. R., Thu, T. T., Nhan, N. H. K., Van Duc, P., and Vu, M. B. (2022). Energy-Efficient Path Planning of Reconfigurable Robots in Complex Environments. *IEEE Transactions on Robotics*, 38(4):2481–2494.

Lampson, B. W. and Redell, D. D. (1980). Experience with processes and monitors in Mesa. *Communications of the ACM*, 23(2):105–117.

Lelli, J. (2014). Multiprocessor Real-Time Scheduling on General Purpose Operating Systems.

Leung, J. Y. and Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250.

Lever, J. H., Ray, L. R., Streeter, A., and Price, A. (2006). Solar power for an Antarctic rover. *Hydrological Processes*, 20(4):629–644.

Liang, Z., He, J., Hu, C., Pu, X., Khani, H., Dai, L., Fan, D. E., Manthiram, A., and Wang, Z.-L. (2022). Next-Generation Energy Harvesting and Storage Technologies for Robots Across All Scales. *Advanced Intelligent Systems*, page 2200045.

Liekna, A., Lavendelis, E., and Grabovskis, A. (2013). Experimental analysis of contract net protocol in multi-robot task allocation. *Applied Computer Systems*, 13(1):6–14.

Liu, C. L. and Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM (JACM)*, 20(1):46–61.

Liu, J., Chou, P. H., Bagherzadeh, N., and Kurdahi, F. (2001a). Power-aware scheduling under timing constraints for mission-critical embedded systems. *Proceedings - Design Automation Conference*, pages 840–845.

Liu, J., Chou, P. H., Bagherzadeh, N., and Kurdahi, F. (2001b). Power-aware scheduling under timing constraints for mission-critical embedded systems. *Proceedings - Design Automation Conference*, (March):840–845.

Liu, L., Brocanelli, M., Chen, J., and Shi, W. (2019). E2M: An energy-efficient middleware for computer vision applications on autonomous mobile robots. *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC 2019*, pages 59–73.

Liu, S. and Sun, D. (2011). Optimal motion planning of a mobile robot with minimum energy consumption. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pages 43–48.

Liu, S. and Sun, D. (2012). Modeling and experimental study for minimization of energy consumption of a mobile robot. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pages 708–713.

Liu, S. and Sun, D. (2014). Minimizing energy consumption of wheeled mobile robots via optimal motion planning. *IEEE/ASME Transactions on Mechatronics*, 19(2):401–411.

Los, U. M. D. E. C. D. E., Roundy, S., Wright, P., and Rabaey, J. M. (2004). *Energy Scavenging For Wireless Sensor Networks With Special Focus On Vibrations*.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66).

Macenski, S., Martín, F., White, R., and Clavero, J. G. (2020). The marathon 2: A navigation system. *CoRR*, abs/2003.00368.

Macenski, S., Soragna, A., Carroll, M., and Ge, Z. (2023). Impact of ROS 2 Node Composition in Robotic Systems. pages 1–8.

Macías-Escrivá, F. D., Haber, R., Del Toro, R., and Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279.

Maciaszczyk, M., Makieła, Z., and Miśkiewicz, R. (2023). Industry 5.0. eBook ISBN9781003384311. pages 51–61.

Madden, M. M. (2019). Challenges using linux as a real-time operating system. *AIAA Scitech 2019 Forum*, pages 1–13.

Maddikunta, P. K. R., Pham, Q. V., B, P., Deepa, N., Dev, K., Gadekallu, T. R., Ruby, R., and Liyanage, M. (2022). Industry 5.0: A survey on enabling technologies and potential applications. *Journal of Industrial Information Integration*, 26(February 2021):100257.

Maidana, R., Granada, R., Jurak, D., Magnaguagno, M., Meneguzzi, F., and Amory, A. (2020). Energy-aware path planning for autonomous mobile robot navigation. *Proceedings of the 33rd International Florida Artificial Intelligence Research Society Conference, FLAIRS 2020*, pages 362–367.

Manacero, A., Miola, M., and Nabuco, V. (2001). Teaching real-time with a scheduler simulator. In *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193)*, volume 2, pages T4D–15.

Martin Murnane, A. G. (2017). A Closer Look at State of Charge (SOC) and State of Health (SOH) Estimation Techniques for Batteries. *Analog devices*.

Maruyama, Y., Kato, S., and Azumi, T. (2016). Exploring the performance of ROS2. *Proceedings of the 13th International Conference on Embedded Software, EMSOFT 2016*, (October 2016).

Matarić, M. J., Sukhatme, G. S., and Østergaard, E. H. (2003). Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14(2-3):255–263.

Mei, Y., Lu, Y. H., Hu, Y. C., and Lee, C. S. (2004). Energy-efficient motion planning for mobile robots. *Proceedings - IEEE International Conference on Robotics and Automation*, 2004(5):4344–4349.

Mei, Y., Lu, Y. H., Hu, Y. C., and Lee, C. S. (2005). A case study of mobile robot's energy consumption and conservation techniques. *2005 International Conference on Advanced Robotics, ICAR '05, Proceedings*, 2005:492–497.

Mei, Y., Lu, Y. H., Lee, C. S., and Hu, Y. C. (2006). Energy-efficient mobile robot exploration. *Proceedings - IEEE International Conference on Robotics and Automation*, 2006(May):505–511.

Mohamed, S. A., Haghbayan, M. H., Miele, A., Mutlu, O., and Plosila, J. (2021). Energy-Efficient Mobile Robot Control via Run-time Monitoring of Environmental Complexity and Computing Workload. *IEEE International Conference on Intelligent Robots and Systems*, pages 7587–7593.

Mok, A. (1978). Multiprocessor scheduling in a hard real-time environment. In *Proc. Seventh Texas Conf. Compt. Syst.*

Mok, A. K. (1983). Fundamental design problems of distributed systems for the hard-real-time environment.

Montemerlo, M., Roy, N., and Thrun, S. (2003). Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. *IEEE International Conference on Intelligent Robots and Systems*, 3(October):2436–2441.

Morales, J., Martínez, J. L., Mandow, A., García-Cerezo, A. J., Gómez-Gabriel, J. M., and Pedraza, S. (2006). Power analysis for a skid-steered tracked mobile robot. *2006 IEEE International Conference on Mechatronics, ICM*, pages 420–425.

Moser, C., Brunelli, D., Thiele, L., and Benini, L. (2006). Real-time scheduling with regenerative energy. *Proceedings - Euromicro Conference on Real-Time Systems*, 2006:261–270.

Moser, C., Brunelli, D., Thiele, L., and Benini, L. (2007). Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems*, 37(3):233–260.

Nilsson, N. J. (1969). A Mobile Automaton: An Application of Artificial Intelligence Techniques. In *International Joint Conference on Artificial Intelligence*, volume 1969, pages 509–520.

Niri, M. F., Bui, T. M., Dinh, T. Q., Hosseinzadeh, E., Yu, T. F., and Marco, J. (2020). Remaining energy estimation for lithium-ion batteries via Gaussian mixture and Markov models for future load prediction. *Journal of Energy Storage*, 28(November 2019):101271.

Parasuraman, R., Kershaw, K., Pagala, P., and Ferre, M. (2015). Model based online energy prediction system for semi-autonomous mobile robots. *Proceedings - International Conference on Intelligent Systems, Modelling and Simulation, ISMS*, 2015-Septe:411–416.

Pardo-Castellote, G. (2003). OMG Data-Distribution Service: Architectural overview. *Proceedings - 23rd International Conference on Distributed Computing Systems Workshops, ICDCSW 2003*, pages 200–206.

Park, J., Delgado, R., and Choi, B. W. (2020). Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems: An Empirical Study. *IEEE Access*, 8:154637–154651.

Partovibakhsh, M. and Liu, G. (2015). An adaptive unscented kalman filtering approach for online estimation of model parameters and state-of-charge of lithium-ion batteries for autonomous mobile robots. *IEEE Transactions on Control Systems Technology*, 23(1):357–363.

Pillai, P. and Shin, K. G. (2001). Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 89–102, New York, NY, USA. Association for Computing Machinery.

Piorno, J. R., Bergonzini, C., Atienza, D., and Rosing, T. S. (2009). Prediction and management in energy harvested wireless sensor nodes. *Proceedings of the 2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace and Electronic Systems Technology, Wireless VITAE 2009*, (June):6–10.

Puschner, P. and Schedl, A. (1997). Computing Maximum Task Execution Times |. *Real-Time Systems*, 13(1):67–91.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., Others, Yoshida, H., Fujimoto, H., Kawano, D., Goto, Y., Tsuchimoto, M., and Sato, K. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Ram, J. P., Babu, T. S., and Rajasekar, N. (2017). A comprehensive review on solar PV maximum power point tracking techniques. *Renewable and Sustainable Energy Reviews*, 67:826–847.

Ramamritham, K. and Stankovic, J. (1994). Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67.

Ramos, O. E. (2019). Optimal control for time and energy minimization in the trajectory generation of a mobile robot. *Proceedings of the 2019 IEEE 26th International Conference on Electronics, Electrical Engineering and Computing, INTERCON 2019*, pages 1–4.

Ren, D., Lu, L., Shen, P., Feng, X., Han, X., and Ouyang, M. (2019). Battery remaining discharge energy estimation based on prediction of future operating conditions. *Journal of Energy Storage*, 25(May):100836.

Renner, C. and Turau, V. (2012). Adaptive energy-harvest profiling to enhance depletion-safe operation and efficient task scheduling. *Sustainable Computing: Informatics and Systems*, 2(1):43–56.

Roche, C. (1999). multi-robot cooperation through negotiated task allocation and achievement. (May):1234–1239.

Rola El Osta (2017). *Contributions to Real Time Scheduling for Energy Autonomous Systems*. PhD thesis, University of Nantes (France).

Schmid, U., Klügl, F., and Wolter, D. (2020). *KI 2020: Advances in Artificial Intelligence 43rd German Conference on AI, Bamberg, Germany, September 21–25, 2020, Proceedings: 43rd German Conference on AI, Bamberg, Germany, September 21–25, 2020, Proceedings*.

Schmitz, M. T. and Al-Hashimi, B. M. (2013). System-level Design Techniques for Energy-efficient Embedded Systems System-level Design Techniques for Energy-efficient Embedded Systems Kluwer Academic Publishers.

Seung, W., Yoon, H. J., Kim, T. Y., Kang, M., Kim, J., Kim, H., Kim, S. M., and Kim, S. W. (2020). Dual Friction Mode Textile-Based Tire Cord Triboelectric Nanogenerator. *Advanced Functional Materials*, 30(39).

Sha, L., Rajkumar, R., and Lehoczky, J. (1990). Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185.

Shin, K. and Ramanathan, P. (1994). Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24.

Shore, A., Roller, J., Bergeson, J., and Hamadani, B. H. (2021). Indoor light energy harvesting for battery-powered sensors using small photovoltaic modules. *Energy Science and Engineering*, 9(11):2036–2043.

Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition.

Sieh, V., Burlacu, R., Honig, T., Janker, H., Raffeck, P., Wagemann, P., and Schroder-Preikschat, W. (2017). An end-to-end toolchain: From automated cost modeling to static WCET and WCEC analysis. *Proceedings - 2017 IEEE 20th International Symposium on Real-Time Distributed Computing, ISORC 2017*, pages 158–167.

Siekmann, J. and Wahlster, W. (2008). *Simulation, Modeling, and Programming for Autonomous Robots - First International Conference, SIMPAR 2008, Proceedings*, volume 5325 LNAI.

Singhoff, F. and Plantec, A. (2007). Aadl modeling and analysis of hierarchical schedulers. volume XXVII, pages 41–50.

Sorell, T. (2022). Cobots, "co-operation" and the replacement of human skill. *Ethics and Information Technology*, 24(4):1–12.

Stankovic, J. A. (1988). A serious problem for next-generation systems. *Computer*, 21:10–19.

Stricker, N. and Thiele, L. (2022). Accurate Onboard Predictions for Indoor Energy Harvesting using Random Forests. *2022 11th Mediterranean Conference on Embedded Computing, MECO 2022*, pages 7–10.

Thomas D, Woodall W, F. E. (2014). Next-generation ROS: Building on DDS. In *ROSCon Chicago 2014, Open Robotics, Mountain View, CA*.

Tokekar, P., Karnad, N., and Isler, V. (2011). Energy-optimal velocity profiles for car-like robots. *Proceedings - IEEE International Conference on Robotics and Automation*, (1):1457–1462.

Touzout, W., Benazzouz, D., and Benmoussa, Y. (2022). Mobile Robot Energy Modelling Integrated Into Ros and Gazebo-Based Simulation Environment. *Mechatronic Systems and Control*, 50(2):66–73.

Valero, F., Rubio, F., and Llopis-Albert, C. (2019). Assessment of the Effect of Energy Consumption on Trajectory Improvement for a Car-like Robot. *Robotica*, 37(11):1998–2009.

Vaussard, F., Rétornaz, P., Liniger, M., and Mondada, F. (2013). The autonomous photovoltaic marXbot. *Advances in Intelligent Systems and Computing*, 194 AISC(VOL. 2):175–183.

Vaussard, F. C. (2015). *A Holistic Approach to Energy Harvesting for Indoor Robots*. PhD thesis, Thesis of EPFL (Switzerland).

Vitolo, F., Rega, A., Di Marino, C., Pasquariello, A., Zanella, A., and Patalano, S. (2022). Mobile robots and cobots integration: A preliminary design of a mechatronic interface by using mbse approach. *Applied Sciences*, 12(1).

Wägemann, P. (2020). Energy-Constrained Real-Time Systems and Their Worst-Case Analyses.

Wägemann, P., Dietrich, C., Distler, T., Ulbrich, P., and Schröder-Preikschat, W. (2018). Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. *Leibniz International Proceedings in Informatics, LIPIcs*, 106.

Wagemann, P., Distler, T., Honig, T., Janker, H., Kapitza, R., and Schroder-Preikschat, W. (2015). Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. *Proceedings - Euromicro Conference on Real-Time Systems*, 2015-Augus:105–114.

Wahab, M., Rios-Gutierrez, F., and El Shahat, A. (2015). Energy modeling of differential drive robots. *Conference Proceedings - IEEE SOUTHEASTCON*, 2015-June(June).

Wang, B., Wang, T., Wei, H., Wang, M., and Shao, Z. (2008). Power-aware real-time task scheduling with feedback control for mobile robots. *2008 3rd IEEE Conference on Industrial Electronics and Applications, ICIEA 2008*, pages 1240–1245.

Wenzel, I., Kirner, R., Rieder, B., and Puschner, P. (2008). Measurement-based timing analysis. *Communications in Computer and Information Science*, 17 CCIS:430–444.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., and Stenström, P. (2008). The worst-case execution-time problem-overview of methods and survey of tools. *Transactions on Embedded Computing Systems*, 7(3):1–47.

Xiao, C., Naclerio, N. D., and Hawkes, E. W. (2019). Energy Harvesting across Temporal Temperature Gradients using Vaporization. *IEEE International Conference on Intelligent Robots and Systems*, pages 7170–7175.

Xiao, X. and Whittaker, W. L. (2014). Energy Utilization and Energetic Estimation of Achievable Range for Wheeled Mobile Robots Operating on a Single Battery Discharge. (August).

Xie, L., Henkel, C., Stol, K., and Xu, W. (2018). Power-minimization and energy-reduction autonomous navigation of an omnidirectional Mecanum robot via the dynamic window approach local trajectory planning. *International Journal of Advanced Robotic Systems*, 15(1):1–12.

Xie, L., Herberger, W., Xu, W., and Stol, K. A. (2016). Experimental validation of energy consumption model for the four-wheeled omnidirectional Mecanum robots for energy-optimal motion control. *2016 IEEE 14th International Workshop on Advanced Motion Control, AMC 2016*, pages 565–572.

Yacoub, M. I., Necsulescu, D. S., and Sasiadek, J. Z. (2016). Energy Consumption Optimization for Mobile Robots Motion Using Predictive Control. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 83(3-4):585–602.

Yamin, N. and Bhat, G. (2021). Online Solar Energy Prediction for Energy-Harvesting Internet of Things Devices. *Proceedings of the International Symposium on Low Power Electronics and Design*, 2021-July.

Yang, C. F. and Shinjo, Y. (2020). Obtaining hard real-Time performance and rich Linux features in a compounded real-Time operating system by a partitioning hypervisor. *VEE 2020 - Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 59–72.

Yang, G. J. and Choi, B. W. (2014). Implementation of Joint Space Trajectory Planning for Mobile Robots with Considering Velocity Constraints on Xenomai. *International Journal of Control and Automation*, 7(9):189–200.

Yang, J., Qu, Z., Wang, J., and Conrad, K. (2010). Comparison of optimal solutions to real-time path planning for a mobile vehicle. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, 40(4):721–731.

Yang, Y., Guo, X., Zhu, M., Sun, Z., Zhang, Z., He, T., and Lee, C. (2023). Triboelectric Nanogenerator Enabled Wearable Sensors and Electronics for Sustainable Internet of Things Integrated Green Earth. *Advanced Energy Materials*, 13(1):1–55.

Yun, X. and Yamamoto, Y. (1993). Internal dynamics of a wheeled mobile robot. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, volume 2, pages 1288–1294 vol.2.

Zhang, W. and Hu, J. (2007). Low power management for autonomous mobile robots using optimal control. *Proceedings of the IEEE Conference on Decision and Control*, pages 5364–5369.

Zhang, W., Lu, Y. H., and Hu, J. (2009). Optimal solutions to a class of power management problems in mobile robots. *Automatica*, 45(4):989–996.

A.1 ISO 25010 standard

Product Quality

Figure A.1 Product quality model proposed by the ISO 25010 standard. It evaluates the *intrinsic quality* (static and dynamic properties) of the software or computing system.

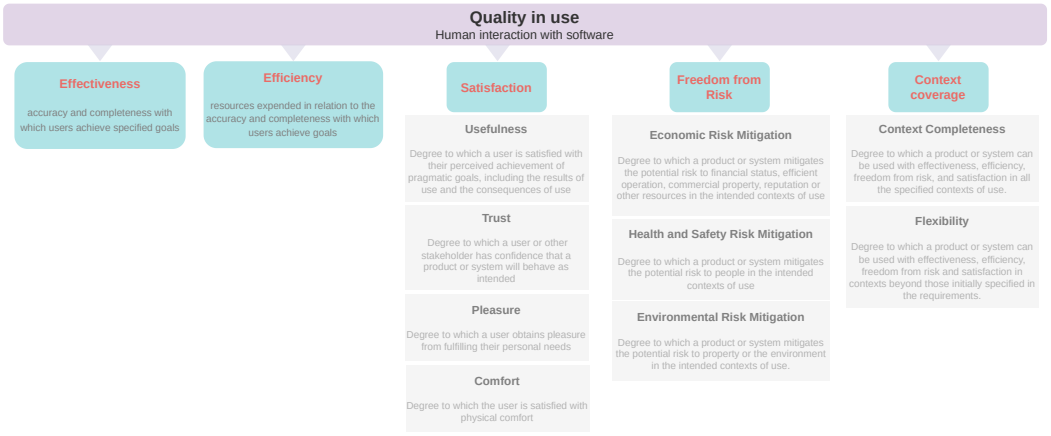


Figure A.2 Quality in use model proposed by the ISO 25010. It evaluates the quality characteristics of the software when it is used under specific context.

B.1 POSIX Thread

```c
#define _GNU_SOURCE
#include <pthread.h>

// Thread function
void* thread_function(void* arg) {
    // code here

    return NULL;
}

int main() {
    pthread_t thread_id;

    // Create a new thread
    int result = pthread_create(&thread_id, NULL, thread_function,
    NULL);
    if (result != 0) {
        // Handle error
        return 1;
    }

    // Wait for the thread to finish
    pthread_join(thread_id, NULL);

    return 0;
}
```

Listing B.1 POSIX Thread

B.2 POSIX Thread With `SCHED_FIFO`

```c
#include <pthread.h>
#include <sched.h>
#include <stdio.h>

void* thread_function(void* arg) {
```

```
    // code here
    while(1){

    }
    return NULL;
}

int main() {
    pthread_t thread_id;
    pthread_attr_t attr;
    struct sched_param param;

    // Initialize thread attributes
    if (pthread_attr_init(&attr) != 0) {
        perror("pthread_attr_init");
        return 1;
    }

    // Set scheduling policy to SCHED_FIFO
    if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0) {
        perror("pthread_attr_setschedpolicy");
        return 1;
    }

    // Get max priority for the SCHED_FIFO policy
    int max_priority = sched_get_priority_max(SCHED_FIFO);
    if(max_priority == -1) {
        perror("sched_get_priority_max");
        return 1;
    }

    param.sched_priority = max_priority;

    // Set the priority in the attribute
    if(pthread_attr_setschedparam(&attr, &param) != 0) {
        perror("pthread_attr_setschedparam");
        return 1;
    }

    // Create a new thread
    if (pthread_create(&thread_id, &attr, thread_function, NULL) !=
   0) {
        perror("pthread_create");
        return 1;
```

```
49      }
50
51      // Destroy thread attributes
52      pthread_attr_destroy(&attr);
53
54      // Wait for the thread to finish
55      pthread_join(thread_id, NULL);
56
57      return 0;
58  }
```

Listing B.2 POSIX Thread With SCHED_FIFO

B.3 POSIX Thread With SCHED_DEADLINE

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <sched.h>
6  #include <sys/syscall.h>
7
8  int sched_setattr(pid_t pid, const struct sched_attr *attr, unsigned
       int flags) {
9      return syscall(__NR_sched_setattr, pid, attr, flags);
10
11 void* thread_function(void* arg) {
12     // code here
13     while(1){
14
15     }
16     return NULL;
17 }
18
19 int main() {
20     pthread_t thread_id;
21     struct sched_attr {
22         uint32_t size;
23         uint32_t sched_policy;
24         uint64_t sched_flags;
25         int32_t sched_nice;
26         uint32_t sched_priority;
27         uint64_t sched_runtime;
```

```
28          uint64_t sched_deadline;
29          uint64_t sched_period;
30      } attr;
31
32      attr.size = sizeof(attr);
33      attr.sched_flags = 0;
34      attr.sched_nice = 0;
35      attr.sched_priority = 0;
36      attr.sched_policy = SCHED_DEADLINE;
37      attr.sched_runtime = 10 * 1000 * 1000;
38      attr.sched_period = attr.sched_deadline = 30 * 1000 * 1000;
39
40      if (pthread_create(&thread_id, NULL, thread_function, NULL) != 0)
        {
41          perror("pthread_create");
42          return 1;
43      }
44
45      sleep(1);
46
47      if (sched_setattr(pthread_getthreadid_np(thread_id), &attr, 0) !=
        0) {
48          perror("sched_setattr");
49          return 1;
50      }
51
52      pthread_join(thread_id, NULL);
53
54      return 0;
55  }
```

Listing B.3 POSIX Thread With `SCHED_DEADLINE`

B.4 Jetson Linux-tegra-evl kernel installation guide

Note: This guide is a procedure to compile and install custom kernel on Jetson AGX Xavier. The kernel version referred in this guide will enable the EVL core kernel on Jetson, however many drivers of Jetson cannot be used, as the kernel does not contain these specific drivers.

Prerequisite

We need to install Jetson SDK Manager on **HOST PC** (x86 machine with ubuntu). Download the Latest SDK manager https://developer.nvidia.com/sdkmanager_deb

- Install the gcc/g++ cross compiler for aarch64 and other utilities

```
\$ sudo apt install gcc-aarch64-linux-gnu
↪  g++-aarch64-linux-gnu build-essential git bison flex
↪  libncurses-dev libssl-dev libelf-dev
```

Versions

1. Host PC : Ubuntu 20.04
2. SDK Manager version : 1.7.3.9053
3. Jetpack version : JetPack 5.0.2
4. Linux Mainline version : linux 5.15-85
5. Xenomai version: EVL linux5.15-85

Kernel Building on Host PC

These steps have to be done on the Host machine (x86 PC) and not on Jetson PC. Download the `linux-evl` kernel source and `libevl` library source from the EVL project repository.

The Evl project repositories can be found in this site https://evlproject.org/. `linux-evl` repository contains mainline kernel with evl sources. `libevl` repository contains the C language API for programming real-time applications which needs to call the EVL core services.

- Open a new working directory to build the kernel source.

```
\$ mkdir linux_evl_kernel
\$ cd linux_evl_kernel
```

- Download the `linux evl` kernel and `libevl` from the EVL project git repository

```
\$ git clone --depth 1 --branch v5.15.85-evl2-rebase
https://source.denx.de/Xenomai/xenomai4/linux-evl.git
\$ git clone --depth 1 --branch r42
https://source.denx.de/Xenomai/xenomai4/libevl.git
```

- Two folders `linuxevl` and `libevl` will be cloned with the kernel source and evl library source code.

- Download the linux evl kernel with sched-edf patch files from the git repository `EVL-linux-kernel-patches` (tag `linux-evl-edf-v1.0`) and the respective evl library patch file from the git repository `EVL-library-patches` (tag `libevl-v1.0`).

- Apply the patches to the linux kernel source and libevl source

```
\$ cd linux-evl
\$ git apply ../evl-linux-kernel-patches/linux-evl-edf-v1.patch
\$ cd ../libevl
\$ git apply ../evl-library-patches/libevl-v1.0.patch
```

Build Linux-EVL

- Kernel compilation

```
\$ cd linux-evl
\$ cp /boot/config-`uname -r`* .config
\$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
↪  UAPI=~/linux_evl_kernel/linux-evl menuconfig
```

- Check if the following configurations are enabled, if not change the configuration accordingly.

```
1. General setup -> Local version - append to kernel release
↪  -> EVL-EDF
2. General setup -> Kernel .config support
3. Platform selection -> NVIDIA Tegra SoC platform (enable)
4. Kernel Features ->  EVL real-time core
5. Kernel Features ->  Dovetail interface
6. Device Drivers -> Out-of-band device drivers -> Timer
↪  latency calibration and measurement
7. Device Drivers -> Out-of-band device drivers -> OOB
↪  context switching validator
8. Device Drivers -> SOC (System On Chip) specific Drivers ->
↪  NVIDIA Tegra194 SoC (enable)
9. Device Drivers -> Thermal drivers -> NVIDIA Tegra thermal
↪  drivers -> Tegra SOCTHERM thermal management (Enable
↪  module)
10. Cryptographic API -> Certificates for signature checking
↪  -> Additional X.509 keys for default system keyring ->
↪  (clear)
11. Cryptographic API -> Certificates for signature checking
↪  -> X.509 certificates to be preloaded into system
↪  blacklist keyring -> (clear)
12. Kernel hacking -> Compile-time checks and compiler
↪  options -> Generate BTF typeinfo (disable)
```

- Compile the kernel

```
\$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
↪  UAPI=~/linux_evl_kernel/linux-evl -j`nproc`
```

Kernel Installation on Jetson PC

There are two methods to install mainline kernel on Jetson PC

Method 1 - Easy and efficient method

Using the ./flash command of nvidia sdk manager

- Install the modules of the compiled kernel into the rootfs of the Jetpack.

```
\$ cd linux_evl_kernel/linux-evl/
\$ sudo make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
↪  INSTALL_MOD_PATH=~/nvidia/nvidia_sdk/
JetPack_5.0.2_Linux_JETSON_AGX_XAVIER_TARGETS/Linux_for_Tegra/rootfs
↪  modules_install
```

- Before flashing the Jetson PC. Make sure to put the Jetson PC into recovery mode.

```
1. Jetson PC should be in power off state.
2. Connect the Power source to Jetson PC.
3. Press the ON button and recovery button (next to power
↪  button) simultaneously.
4. Connect a USB Type-C cable to the Jetson Type-C port (port
↪  facing the led power on) and to
the HOST PC.
```

- Flash the jetson PC using the command below.

```
\$ cd ~/nvidia/nvidia_sdk/
JetPack_5.0.2_Linux_JETSON_AGX_XAVIER_TARGETS/Linux_for_Tegra
\$ sudo ./flash.sh -K
↪  ~/linux_evl_kernel/linux-evl/arch/arm64/boot/Image -d
↪  ~/linux_evl_kernel/
linux-evl/arch/arm64/boot/dts/nvidia/tegra194-p2972-0000.dtb
↪  jetson-agx-xavier-devkit internal
```

Method 2

Compiled kernel module can be directly installed on Jetson PC and flashed with the dtb generated by the kernel build

- Copy the build and compile kernel folder from Host PC to Jetson PC.
- Extract the linux kernel folder into new folder
- Install the kernel

```
~/newDev/linux-5.15.28\$ sudo make modules_install
~/newDev/linux-5.15.28\$ sudo make install
```

- Modify the extlinux to boot the correct kernel

```
~\$sudo gedit /boot/extlinux/extlinux.conf
```

Building EVL library

EVL library uses meson build to install the library in the system. This has to be done on the Jetson AGX.

- Library compilation (fill the system path in <path_to>)

```
\$ cd libevl
\$ mkdir build
\$ meson setup -Dbuildtype=release -Dprefix=/
-Duapi=<full_path_to>/evlkernel/linux-evl
. build
\$ cd build
\$ meson compile
\$ meson install
```

B.5 ROS Graph - Simulation Setup

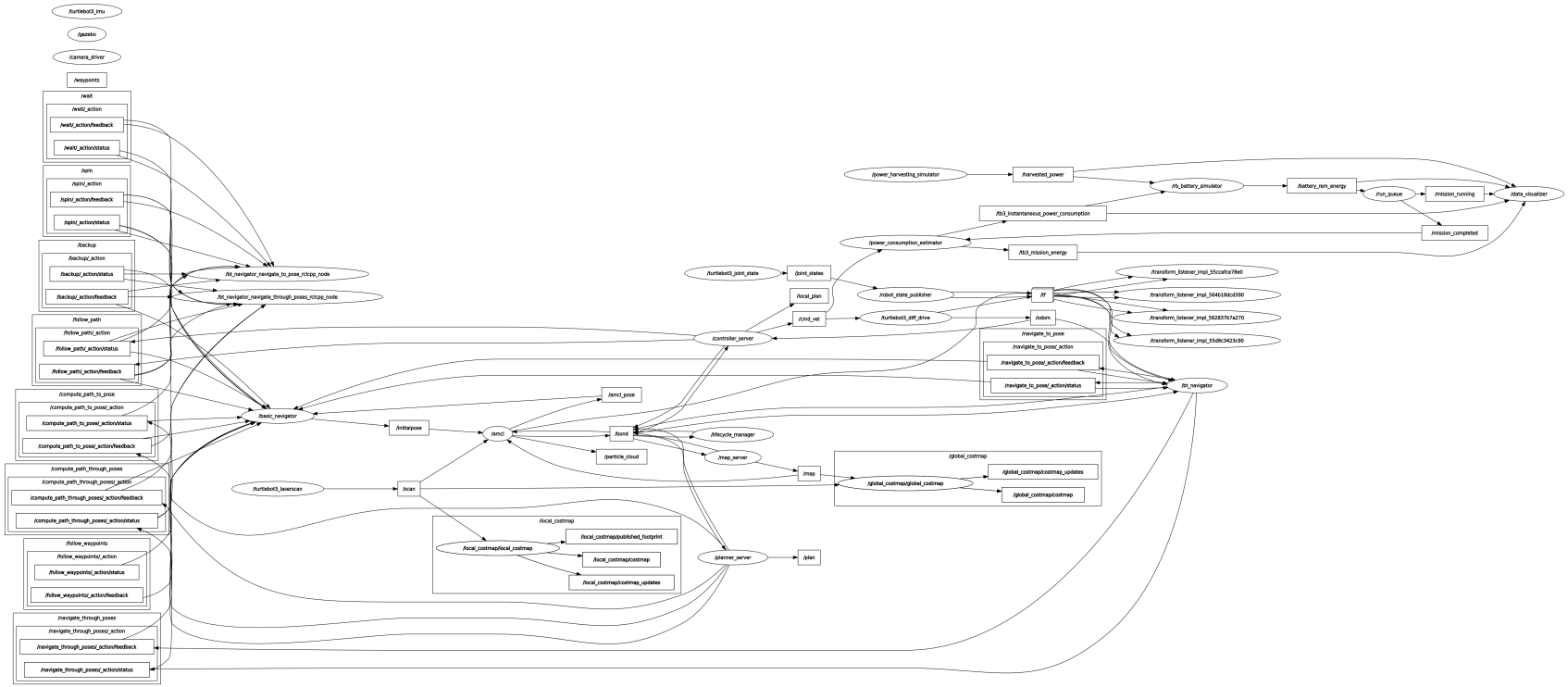


Figure B.1 ROS2 rqt graph of the simulation.

C.1 Jetson AGX Characteristics

	AGX Xavier[1]	AGX Orin[2]
		
AI Performance	32 TOPs	200 TOPs
GPU	512-core Volta GPU with Tensor Cores	1792-core NVIDIA Ampere architecture GPU (with 56 Tensor cores)
CPU	8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3	12-core ARM v8.2 64-bit 3MB L2 + 6MB L3 CPU
Memory	16GB 256-bit LPDDR4x | 137GB/s	32 Go 256 bits LPDDR5 | 204,8 Go/s
Peripherals	UART, SPI, CAN, I2C, I2C, DMIC, GPIOs	UART, SPI, CAN, I2C, I2C, DMIC, GPIOs
Consumption	15W | 30W	15W | 40W

Table C.1 Comparison of AGX Xavier and AGX Orin

The proposed architecture was developed using the Jetson AGX Xavier Developer Kit. For industrial production, the industrial version of the AGX Xavier could be utilized, but it would necessitate the design of a separate carrier board to utilize the peripherals. It should be noted that the Jetson AGX Xavier Developer Kit is no longer available. For ongoing development and research, the Jetson AGX Orin Developer Kit could be considered as a suitable replacement. However, it is important to note that the Linux-Tegra kernel, of the AGX Orin's software stack, would need to be modified to incorporate Linux-EVL features.

[1] https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-agx-xavier/
[2] https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-orin/

C.2 Other Components

Figure C.1 Industrial shield to interface I/O sensors.

The Industrial Shield PLC ARDUINO ARDBOX Analog[3] can be utilized to interface with I/O sensors. This shield can be connected to the Jetson PC via either UART or SPI communication protocols.

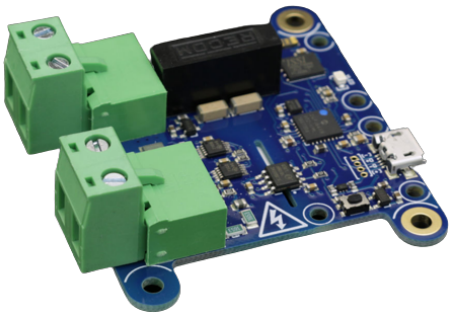


Figure C.2 Yocto-Watt to measure the power consumption.

We utilized Yocto-Watt[4] for power consumption measurements. This tool is provided with libraries in various languages, including C, Python, and C++, offering ease of use for monitoring the energy consumption of the device.

[3]https://www.industrialshields.com/shop/is-ab20an-hf-plc-arduino-ardbox-analog-17
[4]https://www.yoctopuce.com/EN/products/usb-electrical-sensors/yocto-watt

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

Titre : Systèmes cobotiques temps réel sous contraintes d'énergie

Mots clés : Cobots Mobiles, Gestion de l'énergie, Ordonnanceur temps-réel, Récolte d'énergie, Contrôle d'accès aux ressources, Noyau Xenomai, ROS2.

Résumé : Les cobots mobiles sont appréciés dans l'industrie pour aider l'opérateur humain et améliorer la productivité. Nous avons contribué à leur conception en développant d'une part une nouvelle architecture matérielle pour plus de performance et d'autre part un système d'exploitation combinant Linux et Xenomai pour une exécution déterministe du firmware. Pour pallier aux défauts de ROS2, nous proposons l'intégration d'un d'ordonnancement temps réel conduit par la priorité, au sein de Xenomai, ce qui permet aussi de minimiser la latence. Notre approche respecte les normes de qualité ISO 25010. Au coeur de cette thèse se trouve aussi la problématique de l'autonomie énergétique du cobot que nous cherchons à solutionner grâce à la récupération de l'énergie environnementale. Pour ce faire, nous préconisons d'utiliser ED-H, un ordonnanceur de tâches optimal qui assure la neutralité énergétique chaque fois que possible tout en garantissant le respect des contraintes temporelles du cobot.

Une contribution de cette thèse a donc été d'adapter l'ordonnanceur ED-H, initialement conçu pour des tâches indépendantes, à un ensemble de tâches dépendantes accédant à des ressources partagées en exclusion mutuelle. Une nouvelle condition d'ordonnançabilité a été proposée et la performance de ED-H a été évaluée en simulation avant son déploiement dans le noyau Xenomai. Cette preuve de concept nous a conduit à conclure que l'ordonnanceur non oisif EDF reste l'ordonnanceur de tâches temps réel à privilégier y compris sous contraintes énergétiques. Une autre contribution est de proposer l'ordonnanceur ED-H non pas au niveau des tâches logicielles mais des missions du cobot. Nous montrons comment ED-H, sensible à l'énergie, permet de planifier les missions pour gagner en autonomie énergétique. Enfin, nous avons créé une plateforme expérimentale, visant la conception d'un cobot de transport énergétiquement autonome par récupération d'énergie photovoltaïque et embarquant cette nouvelle architecture matérielle et logicielle.

Title : Real-time and energy constrained cobotic systems

Keywords : Mobile Cobots, Energy Management, Real-time Scheduling, Energy Harvesting, Resource Access Control, Xenomai Kernel, ROS2.

Abstract : Mobile cobots are popular in manufacturing to help human operators and improve productivity. We have contributed to the design of cobots by developing both a novel hardware architecture to enhance user interaction, and a dual-kernel OS that combines Linux and Xenomai, ensuring deterministic firmware execution. To overcome the limitations of ROS2, we have integrated a real-time priority-driven scheduling framework with Xenomai to reduce latency. Our methodology adheres to the ISO 25010 quality standards. A crucial challenge this thesis addresses is the issue of energy autonomy. We propose a solution based on harvesting the ambient energy. Our solution involves using ED-H, an optimal real-time computing task scheduler that guarantees energy neutrality whenever possible while satisfying the timing requirements. One contribution of this thesis was therefore to adapt the ED-H scheduler, initially designed for independent tasks, to a set of dependent tasks that access shared resources in mutual exclusion. We proved a new sufficient schedulability condition. We tested the actual performance of ED-H via simulation and then we implemented it in the Xenomai kernel. This proof of concept led us to conclude that the non-idling scheduler EDF remains the preferred real-time task scheduler even under energy limitations. Another major contribution of this work is the proposal to apply the ED-H algorithm, not at the task level, but for managing the cobot missions. We show how ED-H allows for mission planning while leading to increase the duration of the application. Finally, we have created an experimental platform that targets the design of an energy-autonomous cobot for transport. This involves scavenging photovoltaic energy and implementing our novel hardware and software architecture.