eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Department of Automatic Control and Systems Engineering
Faculty of Engineering in Bilbao

Ph.D. Thesis

A GENERIC MULTI-ROBOT ARCHITECTURE FOR MOBILE MANIPULATOR ROBOTS AND ITS INTEGRATION IN THE SMART FACTORY

AUTHOR

Jon Martin Garetxana

SUPERVISORS

Aintzane Armentia Díaz de Tuesta

Oskar Casquero Oyarzabal

Bilbao, July 2023

Abstract

Modern manufacturing is challenged by the dynamic demands of the global market, necessitating flexibility, adaptability, and digitalization in manufacturing systems. In fact, it is evolving towards the manufacture of products with shorter life cycles, mass customization, high quality and shorter, cost-effective delivery.

To address these needs, current production systems leverage the development of various technologies that have converged over the last decade in what is known as the fourth industrial revolution, or Industry 4.0. In this context, hierarchical control systems are giving way to heterarchical systems that allow great adaptability to changes. These new control systems consist of highly distributed and self-organized modules represented as individual Cyber-Physical Systems (CPSs) with social capabilities, comprising both physical and virtual components. Among the different physical assets found in a manufacturing system, a crucial component of the future factory is the Mobile Manipulator Robot (MMR), replacing traditional conveyor belts and enabling quick and efficient reconfigurability of production and increased flexibility.

In this context, this doctoral thesis focuses on solving two of the main challenges of flexible manufacturing systems in the field of MMRs, which constitute its main objective and are translated into the two main con-

tributions of this thesis. On the one hand, to offer a generic framework that facilitates the development of specific robotic applications in different fields, allowing the easy integration of the different robotic skills. On the other hand, to provide this solution with the necessary socialisation capabilities to be able to interact intelligently with other robots or any other asset in a smart factory.

As a solution to the first challenge, this PhD thesis proposes Robotframework, a generic control architecture for MMRs that helps to integrate the necessary navigation, manipulation or perception abilities for the execution of a robot's tasks. The modular design of the architecture, the implementation of skills through the pluginlibs concept and the the integration with high-level decision systems in a standardised way (guided by meta-model and plan format), lead to faster and simplified development of new robotic applications. The architecture is based on ROS, providing a generic and standardized approach for working with different hardware and software and includes generic real-time data collection tools, diagnosis and error handling modules, and user-friendly interfaces. Realistic use cases corresponding to applications that are part of European projects, such as industrial aileron inspection (CRO-INSPECT project) and pest inspection and treatment (Greenpatrol project), validate the generalisability and efficiency of the architecture, and demonstrate its potential for future applications.

It is worth noting that the versatility of Robotframework goes beyond industrial applications, so it is also applicable to a variety of non-industrial scenarios. However, Robotframework alone does not provide the necessary social capabilities for the integration of MMRs in the connected factories of the future.

In order to improve the socialisation capabilities of MMRs, the sec-

ond challenge, this doctoral thesis proposes the integration of Robot-framework in a decentralized flexible manufacturing system following the precepts of the RAMI reference architecture, which proposes the virtualisation of assets as a set of functionalities offered in the form of services. To this end, RAMI identifies the concept of an I4.0 component consisting of an asset and its corresponding Asset Administration Shell (AAS). This work focuses on the MMR asset and its AAS which is in charge of offering the rest of the entities of the manufacturing system (machines, intelligent product, operators...) both the manufacturing services (mainly related to transport tasks) and information about its status stored in models (battery status, location...).

The proposal is validated through a series of incremental use cases representing real plant situations. It begins with a use case based solely on a simulation in ROS, serving to define and evaluate mechanisms of distributed decision-making between machines and robots. The second use case presents a real scenario with machines (controlled by PLCs simulated in CODESYS), and robots represented by Kobukis, communicating through the industrial OPC UA standard. In this case, robots perform material replenishment and transportation of finished parts on-demand. The third and final use case proposes a methodology to integrate Robot-framework with RAMI concepts (based on JADE), abstracting the social capabilities from control functionalities and promoting a more organized and coherent approach to interaction among various agents in the smart factory.

Resumen

La industria manufacturera moderna se enfrenta al reto de las cambiantes exigencias de un mercado global que demanda flexibilidad, adaptabilidad y digitalización en los sistemas de fabricación. De hecho, se evoluciona hacia la fabricación de productos con ciclos de vida más cortos, una masiva personalización, gran calidad y tiempos de entrega más cortos y rentables.

Para hacer frente a estas necesidades, los sistemas de producción actuales aprovechan el desarrollo de diferentes tecnologías que han convergido, durante la última década, en la llamada cuarta revolución industrial, conocida como la Industria 4.0. En este contexto, los sistemas de control jerárquicos están dando paso a sistemas heterárquicos que permiten una gran adaptabilidad a los cambios. Estos nuevos sistemas de control están formados por módulos altamente distribuidos y autoorganizados, representados como Sistemas Ciberfísicos (Cyber-Physical Systems, CPSs) individuales con capacidades sociales, que constan de una parte física y otra virtual. De entre los diferentes activos físicos que se pueden encontrar en un sistema de fabricación, un componente esencial de la fábrica del futuro es el Robot Móvil de Manipulación (Mobile Manipulator Robot, MMR), que sustituye a las cintas transportadoras tradicionales, permitiendo una reconfigurabilidad rápida y eficiente de la producción y una mayor flexibilidad.

En este contexto la presente tesis doctoral se centra en dar solución a dos de los principales retos de los sistemas de fabricación flexibles en el ámbito de los MMRs, que constituyen su objetivo principal y se traducen en las dos principales contribuciones de esta tesis. Por un lado, ofrecer un framework genérico que facilite el desarrollo de aplicaciones robóticas concretas de diferentes ámbitos, permitiendo la fácil integración de las diferentes habilidades necesarias para la ejecución de las tareas correspondientes, cuya implementación es totalmente dependiente de los controladores de bajo nivel del robot utilizado. Por otro lado, dotar a dicha solución de las capacidades de socialización necesarias para poder interactuar de forma inteligente con otros robots o cualquier otro activo de una fábrica inteligente.

Como solución al primer reto, esta tesis doctoral propone Robotframework, una arquitectura de control genérica para MMRs que ayuda a integrar las habilidades necesarias para la ejecución de las tareas de un robot. Pueden ser habilidades comunes, como determinadas tareas de navegación, o habilidades más específicas, como el caso de tareas de manipulación o percepción, que demandan un alto nivel de destreza y precisión. Estas habilidades, encapsuladas en módulos, se integran con con los sistemas de decisión de alto nivel de forma estandarizada, conduciendo a un desarrollo más rápido y simplificado de nuevas aplicaciones robóticas. La arquitectura se basa en ROS, proporcionando un enfoque genérico y estandarizado para trabajar con diferentes elementos hardware y software.
El diseño modular de la arquitectura, la implementación de habilidades mediante el concepto de pluginlibs y la propuesta de unas directrices para la generación de los planes del robot (en forma de meta-modelo y formato del plan) son los pilares sobre los que se construye Robotframework. Casos de uso realistas correspondientes a aplicaciones que

forman parte de proyectos europeos, como la inspección industrial de alerones (proyecto CRO-INSPECT) y la inspección y tratamiento de plagas (proyecto Greenpatrol), validan la generalización y eficacia de la arquitectura, y demuestran su potencial para futuras aplicaciones.

Cabe destacar que la versatilidad de Robotframework va más allá de las aplicaciones industriales, por lo que también es aplicable a diversos escenarios no industriales. Sin embargo, Robotframework por sí solo no proporciona las capacidades sociales necesarias para la integración de los MMR en las fábricas conectadas del futuro.

Con el fin de mejorar las capacidades de socialización de los MMR, el segundo reto, esta tesis doctoral propone la integración de Robotframework en un sistema de fabricación flexible y descentralizado siguiendo los preceptos de la arquitectura de referencia RAMI, que propone la virtualización de activos como un conjunto de funcionalidades ofrecidas en forma de servicios. Para ello, RAMI identifica el concepto de componente I4.0 consistente en un activo y su correspondiente Asset Administration Shell (AAS). Este trabajo se centra en el activo MMR y su AAS que se encarga de ofrecer al resto de entidades del sistema de fabricación (máquinas, producto inteligente, operarios...) tanto los servicios de fabricación (principalmente relacionados con tareas de transporte) como información sobre su estado almacenada en modelos (estado de la batería, ubicación...).

La propuesta se valida a través de una serie de casos de uso incrementales que representan situaciones reales de planta. Comienza con un caso de uso basado únicamente en una simulación en ROS, que sirve para definir y evaluar mecanismos de toma de decisiones distribuidas entre máquinas y robots. El segundo caso de uso presenta un escenario real con máquinas (controladas por PLCs simulados en CODESYS) y robots representados por Kobukis que se comunican a través del estándar industrial OPC UA. En este caso, los robots realizan la reposición de material y el transporte de piezas acabadas bajo demanda. El tercer y último caso de uso propone una metodología para integrar Robotframework con conceptos RAMI (basado en JADE), abstrayendo las capacidades sociales de las funcionalidades de control y promoviendo un enfoque más organizado y coherente de la interacción entre varios agentes en la fábrica inteligente.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

Modern manufacturing is constantly changing in order to adapt to the fluctuations of the current global and competitive market demands. Moreover, products evolve or get obsolete rapidly, making difficult to foresee the sales and leading to an on demand production model. Products require shorter life cycles, which means lower delivery-times, mass customization and high quality, all at reduced costs. These challenges are transforming the way production systems are designed and deployed. Intelligent and customizable manufacturing systems are needed, capable of ensuring efficient management of manufacturing resources, and adapting their production to demand and plant incidents dynamically.

The need to meet these demands has led to a new industrial paradigm usually identified as the *Fourth Industrial Revolution* or *Industry 4.0* (I4.0) [5] which can be defined as *"the advent of cyber-physical systems involving entirely new capabilities for people and machines"* [6], where a cyber-physical system (CPS) is a horizontally and/or vertically connected factory entity consisting of a physical and a virtual part. Numerous government institutions and business organizations have considered the I4.0 paradigm as a key factor in their industrial development strategies. This has given rise to initiatives such as Plattform Industrie 4.0 in Germany and Industry IoT Consortium (IIC) in the USA, to name the most cited ones. Their common denominator is a coordinated effort between government, industry and academia to support innovation in manufacturing processes, based on the total interconnection between different manufacturing assets in a technological context of big data capturing and processing. In this sense, all initiatives are working on the standardization of their reference architectures (e.g., RAMI 4.0 in the case of Plattform Industrie 4.0 and IIRA in the case of IIC) and on the analysis of their confluence. However, standardization and correlation of reference architectures is often a slow process. While moving in this direction, it is important to provide the industry with supporting elements that allow to move forward in the digital transformation. This circumstance raises the following questions regarding this issue:

- **RQ1:** What are the challenges that companies must face in order to address the transition to Industry 4.0?

- **RQ2:** What are the paradigms that can contribute to facilitate the transition to Industry 4.0?

The smart factory concept can be built on a CPS network to create a flexible and autonomous manufacturing grid [7, 8]. The abstraction or virtualization of manufacturing assets in terms of their functionalities, provided as network services, is a necessary condition for materializing the CPS concept. This is the approach followed by RAMI 4.0, which defines the concept of the *I4.0 Component* as a participant of an *I4.0 System* consisting of an asset and its *Asset Administration Shell* (AAS). In turn, the AAS is defined as a virtual representation of the I4.0 Component in the I4.0 System, i.e., a management interface that exposes the information submodels and the manufacturing operations of the asset. Since reference architectures aim to identify the concepts and technologies to be used in the design and development of Industry 4.0 compliant solutions, it is important to understand how they approach I4.0 paradigm. This leads to the following question:

- **RQ3:** What are the key concepts considered by the main reference architectures in the interpretation of Industry 4.0?

Within the great diversity of physical assets that can be introduced in a factory, Mobile Manipulator Robots (MMRs) play a fundamental role to achieve the flexibility and autonomy attributed to a CPS network or an I4.0 System. Replacing the traditional fixed conveyor belts by MMRs enables the quick reconfigurability of the factory layout, increasing its flexibility and scalability [9, 10]. Thus, innovative CPS or I4.0 Components based on mobile robotic solutions that provide flexible navigation and manipulation strategies can help automating the transportation in the factory. Research in this context has been mainly focused on robotic frameworks (RFs) that facilitate information ontologies, algorithms for performing specific tasks, libraries for component integration and tools for development and simulation support, among others. The industrial mobile manipulation task can be solved by combining three main functionalities: 1) navigation, to transport loads from one point to another; 2) manipulation, to perform pick-place or dexterity tasks; and 3) perception, to avoid obstacles on the way or recognize specific targets for manipulation. Although RFs usually provide modules that can be integrated and extended for different applications, combining the different robotic skills is an error prone work that requires experience in many robotic fields, usually deriving on domain specific solutions that are not reusable in different contexts. Thus, every time a new project starts, the state machine that embeds the logic of the robotic applications, the relationship between the different robotic modules, or the user interfaces to externally interact with them, must be usually built from scratch. These issues bring up the following questions:

- **RQ4:** What are the requirements for the design and development of MMRs in the context of Industry 4.0?
- **RQ5:** What information models and functionalities should RFs provide to meet those requirements?
- **RQ6:** How could these information models and functionalities be organized in architectural terms?

Once the single MMR operation is achieved, its integration within a more complex system, such as a CPS network or an I4.0 System, should be considered. New trends in the digitalization of manufacturing systems (such as high connectivity, service orientation and data analytics, among others) caused production control paradigms to evolve from hierarchical control systems that assume a fixed and deterministic context, to heterarchical topologies that show greater adaptability to changes [11]. However, several authors have reported that purely distributed systems present performance problems [12, 13]. The solution to this involves the development of hybrid solutions that combine the benefits of centralized systems and distributed systems [14]. But even hybrid control approaches in distributed and changing environments do not focus on the interaction that arises between multi-robot systems and other manufacturing assets. In this sense, the literature reflects that the main RFs lack the necessary socialisation features to support the rich interaction between robotic and non-robotic assets. In addition to this problem, there is the need to provide distributed intelligence to robots so that they can resolve conflicts collaboratively (e.g., negotiation and decision making) to generate standardized, quality information that can be consumed by analytical applications at higher levels of the control hierarchy, and to develop new skills as services in a standardized way, so they can be used by other management or manufacturing entities in the factory. In order to address these issues, this work explores already proven and reliable paradigms and technologies, such as OPC Unified Architecture (OPC UA) and Multi-Agent Systems (MAS), that have showed a great potential to develop: 1) distributed manufacturing control systems with autonomy and intelligence capabilities; 2) an agile and fast adaptation to the environment changes; 3) an increased robustness against disturbances; and 4) an easier integration of new manufacturing resources and legacy systems. For this, the following questions are considered:

- **RQ7:** What are the requirements and methodologies for the integration of multi-MMR systems in smart factories?
- **RQ8:** Which paradigms and technologies fit those requirements and methodologies?
- **RQ9:** How can an integration architecture fit into the single MMR architecture?

This scenario and the research questions that have been derived from it, serve as a framework for the development of this thesis.

1.2 Goals

The main objective of this work is to develop a generic architecture for autonomous MMRs that facilitates the integration of different robotic skills, and provides MMRs with socialization capabilities that allow them to interact with other manufacturing assets in a smart factory. To achieve this general goal, the following partial goals are proposed:

- **O.1.** Analyse the initiatives and roadmaps that are being proposed in the most prominent industrialized countries to carry out the digitalization process of the factory (RQ1-RQ3).
- **O.2.** As for the generic architecture for autonomous MMRs:
  - **O.2.1.** Define the MMR tasks through industrial use cases and establish the main requirements for a single MMR (RQ4).
  - **O.2.2.** Develop a generic robot architecture that, based on the single-robot requirements, considers complex MMR functionalities, and provides reusable high-level robot management, monitoring and diagnosis tools (RQ5, RQ6).
  - **O.2.3.** Validate the proposed architecture with single-MMR realistic use cases, in both industrial environments and non-industrial ones (RQ4-RQ6).
- **O.3.** Regarding the integration of the proposed generic MMR architecture within a smart manufacturing system:
  - **O.3.1.** Identify the heterogeneous production entities that take part in a distributed manufacturing control system (RQ7).
  - **O.3.2.** Define the communication mechanisms to enable the interaction among robots and with other intelligent manufacturing entities (RQ7, RQ8).

– **O.3.3.** Define a decentralized architecture to integrate MMR services within the flexible manufacturing system (RQ9).

1.3 Phases

This thesis covers the results of six years of research work (2017-2023), performed on a part-time basis, on MMRs in the field of Industry 4.0. This work was framed in different projects, both industrial and academic, performed at the Technische Hochschule Nürnberg, the University of the Basque Country (UPV/EHU), and Tekniker research center dedicated to technological and industrial research. This trajectory has allowed the development, optimization and validation of this work.

Technische Hochschule George Simon Ohm

The participation period in the Technische Hochschule George Simon Ohm in Nuremberg, Germany, overlaps the first years of the thesis, from 2017 to 2018. This phase allowed to delimit the current knowledge related to the problem domain, to outline the research questions, and to identify the possible implications of this research for practice. This made it possible to address the first objective of the thesis **(O.1)**. The main activities performed during this period constitute an important legacy for this thesis, specifically: developing an Autonomous Transport Vehicle (ATV) for Bosch GmbH, and playing the team leader role of a newly built RoboCup@Work team. Both projects envision an autonomous MMR performing logistic tasks on a industrial scenario requiring specific navigation, manipulation and perception skills that showed certain common aspects in different applications. Thus, this phase comprised the search, review and analysis of the scientific literature and engineering solutions regarding the specific topics of the problem domain, as well as the links among them. As a result, this period provided a glimpse of the difficulties and the work that remained to be done, to develop robotic skills in a standardized and reusable manner, and to integrate a multi-MMR fleet into a factory by decentralizing the control at plant level.

University of the Basque Country (UPV/EHU)

With this previous experience, in 2017 the thesis was formally registered within the Doctoral Program in Control Engineering, Automation and Robotics of the University of the Basque Country, and drawn on the experience of the Systems Control and Integration research group (GCIS, for its acronym in Spanish), focused on the use of distributed intelligence technologies for the management of manufacturing systems in accordance with the Industry 4.0 paradigm. This made it possible to address the third objective of the thesis **(O.3)**. As a result, a multi-layer approach for the integration of MMRs as I4.0 Components was proposed in response to that objective and published in [15]. The purpose of this approach is to ease MMR integration into a manufacturing system and provide I4.0 Components in a factory with access to MMR-related services, i.e., manufacturing services that provide MMR-related functionalities. This phase benefited from the involvement in the *Smart Distributed Architecture for Enabling the Fog-in-the-Loop in i4.0* research project (SMARTFOG, reference RTI2018-096116-B-I00), financed by Ministerio de Ciencia, Innovación y Universidades (MCIU), Agencia Estatal de Investigación (AEI), Fondo Europeo de Desarrollo Regional (FEDER), Unión Europea (UE).

Tekniker

From 2019 onwards, the thesis was developed at Tekniker technology research center, which offered the chance to collaborate on the development of a generic robotic framework for multipurpose use cases. This experience combined with the previously collected one at the Technische Hochschule Nürnberg, made it possible to address the second objective of the thesis **(O.2)**. The Department of Autonomous Intelligent Systems in Tekniker is strongly engaged with the setup of a neutral experimentation infrastructure for intelligent automation applications for collaborative robotics. This workspace offers robotised mechanisms to implement fully manual assembly processes as collaborative assembly processes, with the robot performing the majority of the work, and the human operator helping out with specifically difficult subtasks. These efforts are being conducted in different European projects. The *AUTOWARE* [16] project is rooted in the ICT Innovation for Manufacturing SMEs (I4MS2), an initiative to enhance the digital transformation of the European manufacturing sector. Tekniker provides the mobile platform working on a neutral experimentation environment to evaluate heterogeneous communication and networking architecture supporting connectivity and data management in CPSs. The *A4Blue* [17] project proposes the development and evaluation of a new generation of sustainable, adaptive workplaces. Dealing with evolving requirements of manufacturing processes, flexible and efficient automation mechanisms are introduced to optimize human-machine interaction by personalized and context-aware assistance capabilities. Finally, other European projects such as *Cro-Inspect* and *Greenpatrol* have been used to develop and validate some of the contributions of this thesis. As a result, the generic robot architecture *Robotframework* was contributed and reported in [18]. The purpose of this architecture is to ease the integration and extension of MMR functionalities to create high-level and reusable robot management services.

The aforementioned research phases offered different benefits. From a scientific point of view, they gave the opportunity to deal in depth with the MMR research strand and to get into the Industry 4.0 paradigm. In addition, the experimental part of this thesis extends the body of studies about MMRs towards industrial applications. From an applied point of view, they gave the opportunity to include various relevant case studies that provided real implementation and feedback about the proposed architectures and the problems examined. This was essential to understand some of the limitations of the present work. From the diffusion of research point of view, they allowed to gradually improve the quality, quantity and range of the publications derived from it. The findings have been presented in various international peer-reviewed conferences and published in several JCR peer-reviewed scientific journals. Finally, from a social point of view, this work allowed the author to greatly expand his network of scientific contacts. This fact will hopefully allow the author to collaborate and participate in different national and international research projects and events in the future.

1.4 Structure

Chapter 2

Once the motivation and objectives of the research work have been defined, in the second chapter - State of the Art - different approaches that try to solve the demands of MMRs and their integration in flexible manufacturing control systems in the context of Industry 4.0 are reviewed. In this chapter, first, the evolution of manufacturing paradigms is described; Second, the growth of robotic applications during the last decade is presented, analysing its influence on different domains; third, the trend of flexible manufacturing and the latest efforts to integrate MMRs in the factories of the future are analysed.

Chapter 3

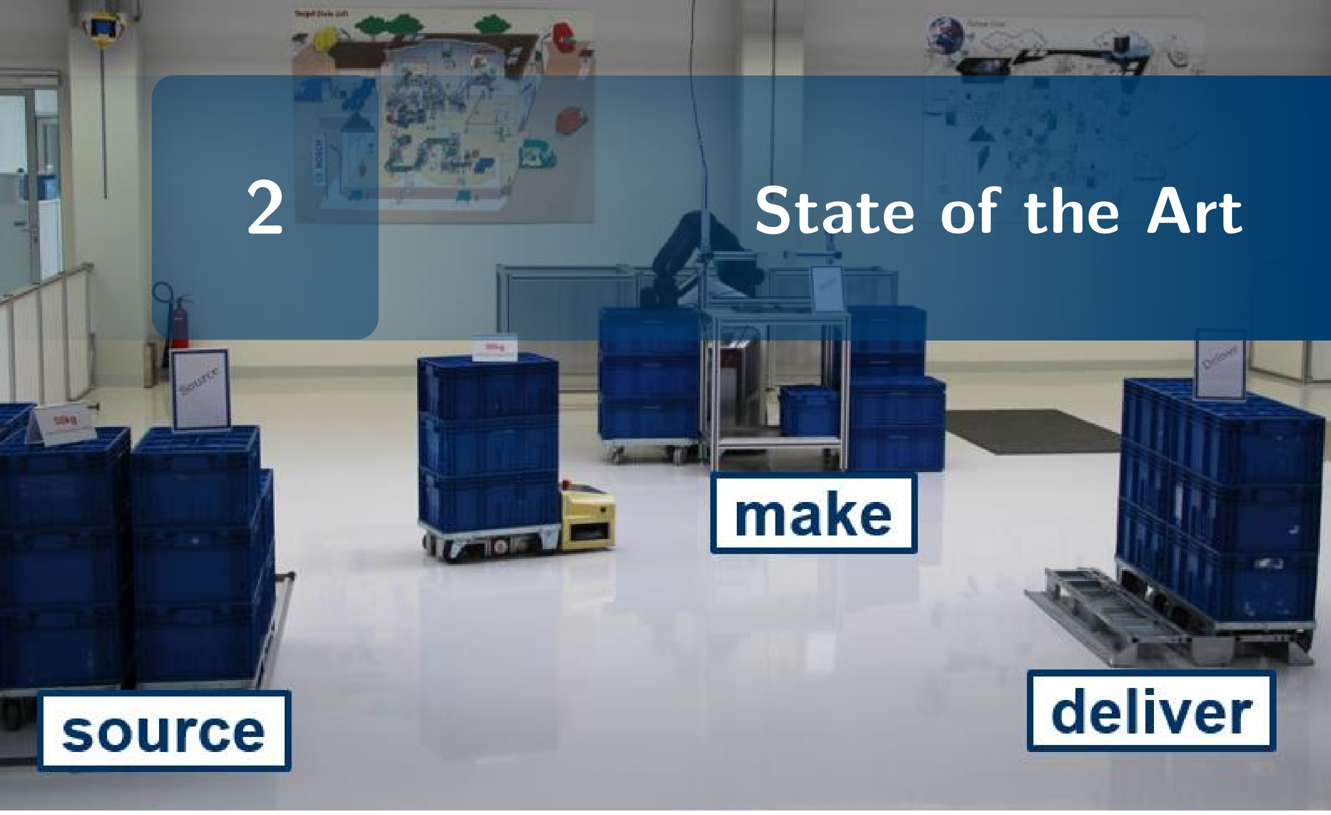
Here, the state of the technology is reviewed, focusing on robotic frameworks for mobile manipulators, multi-agent frameworks for distributed industrial applications, and industrial communication standards. After detecting the weaknesses and strengths of these frameworks standalone, the benefits obtained by integrating them together are analysed and the most suitable technological frameworks are presented.

Chapter 4
Using the information provided in Chapters 2 and 3, this chapter analyses relevant industrial MMRs in order to detect similarities in their hardware and software modules and to define generic framework requirements for single robot use cases. Then, the development of a generic robot architecture for MMRs, the so-called Robotframework, is presented and evaluated through an industrial inspection use case and a more generic agriculture use case.

Chapter 5
Introduces the entities of a distributed manufacturing system represented by physical resources such as MMRs, machines or edge computing devices, and application resources such as production management, monitoring activities or smart production orders. Here, multi-agent frameworks and industrial standards are used to develop distributed task allocation and traceability mechanisms, and to integrate the previously presented single-robot architecture within a distributed multi-robot flexible manufacturing. To evaluate the distributed task allocation mechanisms, simulated and on-field use cases will be presented, where robots transport products from one machine to another, and replenish the machines with new raw material.

Chapter 6
Finally, the conclusions and contributions of this work are summarized, and future work is presented.

2 State of the Art



This chapter is devoted to the analysis of the literature related to the main objective of this thesis: the design of a generic architecture for autonomous MMRs that allows their integration in flexible manufacturing systems in the context of Industry 4.0. As a result, it is expected to obtain a better knowledge of the state of the art in this field, and to identify the gaps and limitations of the solutions proposed in the current literature with respect to that objective. The volume and variety of related works makes it essential to consider limiting and organizing the state-of-the-art in order to facilitate the review process. For this purpose, research context has been divided into three blocks:

- Section 2.1 describes the evolution of manufacturing paradigms. To that end, the main ongoing reference architectures and international roadmaps proposed by globally relevant institutions over the last decade are reviewed, since they provide the conceptual and normative framework for Industry 4.0.

- Section 2.2 describes the growth of robotic applications during the last decade, analyzing its influence on different domains. The design of a MMR requires efficient and reliable procedures to describe the vital aspects of its development by means of RFs that provide tools to develop generic robotic skills, standardized interfaces to provide robotic services, and models to support correctness of the robotic applications. Thus, this section presents an analysis of works oriented to the design and/or development of robotic control systems for transportation and manipulation.

- Section 2.3 focuses on the domain of flexible manufacturing, and describes the latest efforts to integrate MMRs in the factory in order to achieve interoperability between those and the rest of the participants in a factory. Thus, this section analyses how to integrate manufacturing assets into Industry 4.0, but particularized for the case of an MMR. The generality of possible solutions is a key aspect in addressing this issue, so the main characteristics of the distributed nature of the current manufacturing systems and the available technologies for this purpose are considered.

This chapter concludes with a corollary including the main requirements for a set of MMRs in a smart factory.

2.1 Industry 4.0: a planned (r)evolution

Industrial revolutions constitute great social transformations derived from technological advances that have marked clear milestones in the scientific development of humanity. Over the last 250 years, the world has experienced three industrial revolutions [5]. The first industrial revolution took place between the 18th and 19th centuries, and involved the mechanization of tasks, mainly through the use of steam-powered machinery, which resulted in the rise of the factory system. At the end of the 19th century and the beginning of the 20th century, the second industrial revolution took place, where new advances related mainly to electricity and fossil fuels, and methods for manufacturing standardized and interchangeable parts, led to the appearance of factories capable of mass production. In the second half of the 20th century, the advent of electronics and information technology made it possible to automate production lines, leading to what is known as the third industrial revolution [19]. This revolution resulted in the era of high-level automation in production thanks to two major inventions: Programmable Logic Controllers (PLCs) and robots.

Nowadays, the entire scientific community and society in general accepts and recognizes these three industrial revolutions have taken place to date. However, defining a set of events as an industrial revolution is relatively easy with the perspective given by history: once the transformation of the economy and society becomes manifest and consolidated, it is easy to go the other way around to find the root causes. However, it is much more difficult to identify a revolution from a present perspective, since it is necessary to predict the future, imagining the level of rupture necessary in social and economic structures to make it a reality. In this sense, industry is undergoing a profound and dizzying transformation that has led to a torrent of initiatives and technological advances that many institutions have agreed to identify as the fourth industrial revolution [20]. This paradigm lies mainly in the distribution of intelligence in systems, understood as the capacity of the entities that make them up to interact by interpreting information, and making decisions to achieve their objectives. The transversality of this paradigm allows its application in multiple fields, e.g., smart cities, smart grids or smart factories.

2.1.1 Reference architectures for Industry 4.0

For the first time in history, there has been the possibility to anticipate the industrial transformation that is to come and identify the associated change vectors. This has allowed governments and institutions of the major industrialized economies to prepare for a new paradigm and present their strategies to meet the challenges of the new industrial revolution.

Reference Architectural Model Industrie 4.0 (RAMI4.0)
The first of these strategies saw the light at the 2011 Hannover Fair, where the German government presented its *Industrie 4.0* initiative [21]. This initiative stated the obsolescence of current production models and placed the irruption of the fourth industrial revolution in the imminent future. In order for German industry to adjust to the future manufacturing paradigm, the Industrie 4.0 initiative proposes the intelligent interaction of humans, machines and processes through the integration of CPSs and the Internet of Things (IoT) [22] to reach flexible and product oriented manufacturing [23]. To that end, a reference framework is needed to identify each entity of the system and provide a common language among them. Thus, in 2015, Plattform Industrie 4.0 presents the Reference Architectural Model Industrie 4.0 (RAMI 4.0) [24].

RAMI 4.0 is a service-oriented architecture that provides a structured description of a manufacturing system in the context of Industry 4.0 (hereafter referred to as *I4.0 System*). It presents a cubic model that provides a framework for a common understanding among the entities of an I4.0 System with respect to three axes:

- *Layers* axis: It describes the six functional levels into which manufacturing systems for Industry 4.0 can be divided. In each of these layers, service definitions abstractly describe the functionality provided to a layer N by a layer N-1.

  - *Business* layer: It orchestrates the high-level services to determine the status of the processes at a factory level.
  - *Functional* layer: It provides a definition of the high-level services offered by an asset and manages their access remotely.
  - *Information* layer: It acquires, processes and adapts the data from assets while ensuring its integrity and persistence.
  - *Communication* layer: It establishes architectural styles, message patterns and data formats to ensure interoperability.
  - *Integration* layer: It offers low-level services that enable access to the data and functionalities of the asset.
  - *Asset* layer: It represents the physical or logical entities with value for a company, including human beings.

- *Life Cycle Value Stream* axis: Supported by IEC 62890 [25], it describes the operational status of the product, differentiating between product type and product instance:

  - *Product type*: A product in development stage constitutes a product type.
  - *Product instance*: A manufactured product constitutes an instance of a product type.

- *Hierarchy Levels* axis: It adopts some of the factory hierarchy levels of ISA 95 [26] and ISA 88 [27] (such as *Enterprise*, *Work centers*, *Stations* and *Control device*), while adding additional levels to make the factory hierarchy consistent with Industry 4.0 [28]:
  - *Connected world*: It represents a group of companies collaborating above the *Enterprise* level.
  - *Field device*: It represents devices that are directly involved in the manufacturing process below the *Control device* level.
  - *Product*: It represents the product in the factory hierarchy.

Together with this cubic model, RAMI 4.0 also introduces the *I4.0 Component* as a participant of a manufacturing system [29]. An I4.0 Component consists of an asset (physical part) and an *Asset Administration Shell* or AAS (virtual part). I4.0 Components are service-oriented: the AAS provides the asset an interface through which service requests from other I4.0 Components are channeled. Within the AAS, service requests are handled by a *Component Manager* that manages the *AAS submodels* (also called *Manifest*), made up by the set of properties that describe the data and functionalities of the asset. Services provided by a I4.0 component can be grouped into two categories as seen in Figure 2.1:

- *Submodel Services:* They provide access to the information submodels of an I4.0 component without interacting with the asset.
- *Asset Related Services:* They involve an interaction with the asset to execute some functionality or operation, or to manage its state.

Services provided by I4.0 Components can be combined to compose manufacturing applications that are offered to other components, resulting in *Application Relevant Services* [30].

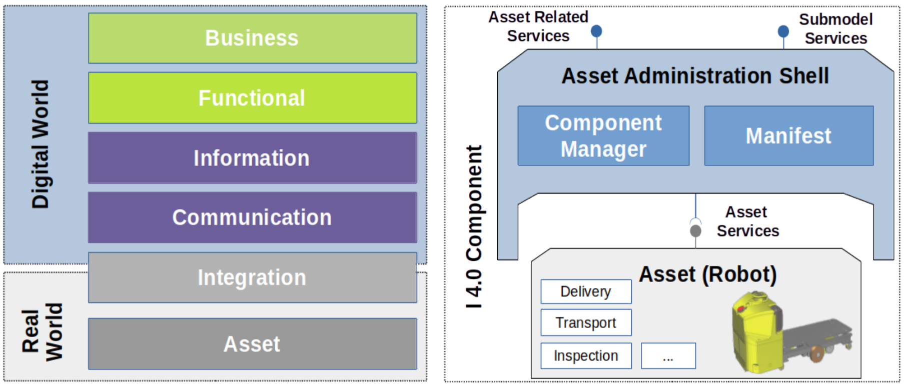


Figure 2.1. I4.0 Component representing a Robot-asset and its Asset Administration Shell (AAS) with respect to the RAMI layers.

However, I4.0 Systems also require *Infrastructure Services* to manage I4.0 Components and support them in the execution of *Application Relevant Services*. *Infrastructure Services* are classified into two categories:

- *AAS Services:* they manage the information and functionalities (i.e., *Application Relevant Services*) of the I4.0 Components, and can, therefore, be performed by the AASs themselves (e.g., to manage AAS-related information or to control the access to Application Relevant Services)
- *AAS Infrastructure Services:* they manage the AASs in the system as a whole (e.g., to register and create them, or to make them reachable to each other). These services cannot be performed by the AASs themselves. Instead, they are offered by platforms in charge of managing the I4.0 Systems.

Industrial Internet Reference Architecture (IIRA)
The next initiative, in terms of both age and relevance, is *Industry IoT Consortium* (IIC) [31], which was constituted as a non-profit consortium in 2014 among industry, academia and the U.S. Government around the

Industrial Internet concept promoted by the General Electric in 2012 [32]. This concept promotes the application of ICT into industry to enable interconnected and intelligent systems capable of processing and analyzing large amounts of data. In 2015, the IIC released the Industrial Internet Reference Architecture (IIRA) to guide the development of those interconnected and intelligent systems.

IIRA is based on the ISO/IEC/IEEE 42010 standard on description of architectures in systems engineering [33], which organizes software architectures in *Viewpoints*. These Viewpoints frame the description and analysis of specific problems, named *Concerns*, in the system. These Concerns can refer to any relevant aspect of the system. System participants, or *Stakeholders*, may show interest in different Concerns, and, by extension, in different Viewpoints of the system [34]. Specifically, IIRA is structured in four Viewpoints:

- *Business Viewpoint:* It deals with Concerns at the enterprise level (e.g., calculation of maintenance costs and expected benefits) and identifying the capabilities of the system to meet those objectives. Stakeholders interested in this Viewpoint are typically business decision makers, product managers and system engineers.

- *Usage Viewpoint:* It addresses Concerns related to basic manufacturing services that provide functionality that can be combined to provide the system capabilities identified in the *Business Viewpoint*. Stakeholders interested in this Viewpoint are typically system engineers and product managers.

- *Functional Viewpoint:* Its main Concern is related to functional components of the IIoT system (i.e., their structure, and interfaces both with other system components and with external elements). Stakeholders interested in this Viewpoint are typically system and component architects, developers and integrators.

- *Implementation Viewpoint:* It focuses on Concerns involved in the implementation of the functional components of the system (e.g., the selection of technologies and devices). Stakeholders interested in this Viewpoint are typically system and component architects, developers and integrators, and system operators.

Cooperation has been established between Plattform Industrie 4.0 and IIC. In 2017, a comparative analysis between RAMI 4.0 and IIRA [35] led to the conclusion that a relationship can be established between the layers defined in the Layers axis of RAMI 4.0 and the domains considered by IIRA in its Functional Viewpoint. In 2020, both organizations published a new joint paper comparing the IIC understanding of the Digital Twin (DT) with the AAS concept of RAMI 4.0 [36]. It was concluded that the AAS provides full support for the key requirements of DTs described and characterized by the IIC.

Intelligent Manufacturing System Architecture (IMSA) & Industrial Value chain Reference Architecture (IVRA)

Similarly, 2015 saw the appearance of the *China Manufacturing 2025* [37] and the Japanese *Industrial Value Chain* initiative [38]. Both pose their own reference architectures: Intelligent Manufacturing System Architecture (IMSA) in the chinese case and Industrial Value chain Reference Architecture (IVRA) in the Japanese case. Both architecture references are strongly influenced by the RAMI 4.0 and IIRA:

- IMSA is clearly influenced by RAMI [39], as it proposes a cubic model organized in three dimensions : *Life Cycle*, representing the chain of activities that add value to a product from design to commissioning; *System Hirarchy*, which takes its four lower levels (Equipment, Control, Workshop and Enterprise) from IEC 62264 standard [26] and adds the *Cooperation* level at the top of the hierarchy to illustrate the collaboration between companies throughout the life cycle of a product; and *Inteligent Functions*, which are organized in five levels (*new business patterns*, *information fusion*, *interconnection*, *system integration* and *resources*).

- IVRA introduces the concept of the Smart Manufacturing Unit (SMU), which defines any participant in a manufacturing system from three viewpoints: *Asset View*, which shows the assets that are valuable to the company; *Activity View*, which covers the activities performed by the assets comprising the SMUs; and *Management View*, which focuses on the different management areas that can be worked on to optimize the performance of a SMU.

2.1.2 Confluence of Industry 4.0 roadmaps

Despite the fact that some of the aforementioned initiatives were launched almost a decade ago, the degree of adoption of the precepts of the Fourth Industrial Revolution by many companies is remaining moderate or nonexistent [40]. The lack of technical content and the apparent vagueness with which these initiatives have been disseminated have been the subject of debate, assessing the balance between the actual degree of contribution of the proposal and its lack of support for the benefits it theoretically reports [41]. However, this has not prevented *Industrie 4.0* from being adopted as a reference initiative at the European level (Factories of the Future [42]), with replicas at state [43] and regional [44] levels. The same happens to *Industrial Internet* at state level (Manufacturing USA [45]). Although each roadmap has a special emphasis on certain characteristics related to their local situation, all of them show broadly similar characteristics. The following is a summary of the common characteristics of German [21, 46, 47], European [48, 49] and American [45] roadmaps relevant to this thesis:

- **Adaptable Production:** Continuous reduction of product and innovation cycles requires an adaptable manufacturing that permits the quick reconfigurability of production capacities and capabilities. Modular plug and play components can allow a quick, inexpensive and reliable expansion, reconfiguration and reusability, enabling last-minute changes, and permitting a flexible response to disruptions and failures on behalf of suppliers.

- **Self Organizing Adaptive Logistics:** The transportation of goods, from the factory material supply to the delivery of the product to a client, are needed to be managed by self-learning systems that react flexibly to system failures and customer priorities. This will enable shorter delivery times, lower inventories, and improve the use of the available infrastructure.
- **Data Collection:** IT platforms collect data from production systems, logistics or delivered products, and use them as raw material for data mining processes. This way, both products and production processes could be optimized, which, in turn, could lead to the creation of new products.
- **Operator Support in Production:** This feature includes the use of technology to help the humans to perform a better work, and physical assistance by using equipment for hazardous or monotonous tasks. This context-related assistance can provide support in analysis and decision-making during diagnosis. These aspects will allow the operator to focus on value-added tasks, while benefiting from technological support for repetitive or hazardous tasks.
- **Smart Product Development for Smart Production:** Virtual products allow for new types of teamwork in engineering processes and automation of engineering activities. This involves methods and tools for modeling, simulating and forecasting the behaviour of production processes. As a result, it is expected that producers,

system suppliers and customers will work together to define requirements and coordinate functionalities, providing for additional benefits in subsequent process and value-added steps.

- **Model Driven Engineering:** Models and standard methodologies are basic prerequisites to bring together the different groups engaged in the development of the smart product, reducing the risks through early detection of errors or early verification of the demands placed on the system. Explanatory models that describe interactions and behaviours in the real world can be useful for validation purposes during the development and design stages.

- **Traceability:** The information provided by IEC 62264 and IEC 61512 standards does not have attributes to represent the state of the manufacturing system at all times. Full traceability of products or factory components life-cycle is needed to record the evolution of the current manufacturing process (i.e., when and where operations are performed on the product), which might not conform to the original planning in case of unforeseen events. To that end, it is necessary to extend the *Process* model, as well as the *Operations* definition model and the *Operations Schedule* model.

- **Standardization:** Standardized, multi-vendor and modularized production systems are needed as sample references for Industry 4.0 [50]. On the one hand, this will permit the interoperability of vendor-specific hardware and software, enabling the plug-and-play connectivity of components with the same functionality. On the other hand, it requires the definition of standard communication interfaces such as RFID and OPC UA, web server technologies using the REST architectural style, and common data formats such as JSON and XML.

2.2 Evolution of autonomous mobile manipulator robots

The interest in mobile manipulation applications has grown remarkably during the last years in multiple domains: industrial manufacturing logistics, inspection or assembly, precision agriculture, exploration or rescue, healthcare or domestic assistance. Robots offer efficient solutions for industrial tasks with minimal interaction (e.g, inspection) or with complex interaction (e.g., material transportation). This flexibility, offered by the redundancy of different robot skills and capabilities, needs to be accurately orchestrated to ensure enhanced performance. Therefore, decision-support methodologies and frameworks are required for successful mobile manipulation in (semi-)autonomous working scenarios. Due to the extensive literature available on this subject, this work focuses on wheeled mobile manipulators, with special attention to its industrial applications.

2.2.1 Application domains

This subsection presents the main aspects related to MMRs and the common denominator of the technologies that enable their practical use. These aspects are organized by the improvements in the main application areas in which the MMRs have acquired a key role.

Healthcare

The usefulness of mobile manipulator service and assistance robots is quite apparent at home and hospital environments, where safe human collaboration is a very important aspect [51]. Robots can be used to transport and administrate medication [52, 53], assist elder or disable patients [54, 55], or provide therapeutic assistance in hospitals [56]. They can also be used at home for cleaning [57], cooking [58] or organizing library shelves [59]. The dynamic and continuously changing environments in which robots have to work hinder the navigation and safety pick and place operations tasks as demonstrated in [60–63]. These uncertainties need to be reflected in the robot's behaviour as discussed and studied in [55, 64].

Precision Agriculture

Digital farming is considered a key aspect that must be addressed to increase the production and the efficiency of farming in order to provide food for the exponentially growing human population [65–67]. Sensors, robotics and data analysis for automatically maintaining and monitoring greenhouses can help to the transformation of farms into intelligent systems, making cropping system smarter and, thus, enhancing the agricultural productivity. In this sense, research has been focused on two main areas: weed inspection and fruit and vegetables harvesting.

Weed inspection is mostly represented by outdoor robots for weed detection and spraying: the Graph Weeds Net [68] formulates multi-scale graph representations for weed classification using deep learning techniques; the RHEA project is centered on both agriculture and forestry [69]; the BoniRob project is dedicated to multipurpose farming [70]; the CROPS project is focused on precision spraying in vineyards [71]. The navigation of these outdoor robots is largely based on the use of satellite localization systems. However, the signal used by these systems is much weaker and imprecise in indoor environments, making them less suitable for greenhouses [72].

The unstructured environments with constantly growing plants have led most of the harvesting robots found in greenhouses to use rails to navigate them [73–76]. With the aim of avoiding the setup of additional and expensive infrastructure in the greenhouse, latest robotic solutions have proven to successfully use Galileo satellites combined with Inertial Measurement Units (IMU), odometry and Light Detection and Ranging (LiDAR) sensors to provide a more flexible autonomous navigation in greenhouses [77].

The use of fixed paths, such as rails, for navigation in greenhouses has lead to decoupling navigation from manipulation and inspection tasks. This has resulted in overlooking the navigation functionalities while developing the CROPS robot framework [78], where the control architecture covers only the fruit localization and arm control functionalities. FroboMind [79], another interesting open source control architecture, demonstrates that a common and reusable architecture, tailored to precision agriculture robots, significantly decreases the development time and resources due to efficient reuse of existing work across projects. In this sense, BoniRob [70] may be considered outdated as it does not integrate the state-of-the-art accepted *navigation_stack* for navigation or *MoveIt!* for manipulation.

Exploration and Rescue

Other application areas with mobile manipulators working on unstructured environments are exploration and rescue in catastrophic areas (e.g., in an earthquake [80, 81]), nuclear examination [82] or underground scenarios [83, 84]. The surroundings in these contexts are frequently poorly mapped, with significant uncertainty, resulting on challenging navigation and dexterity controls [85–87].

Competitions

MMRs are also an important part of several robotic competitions, being probably RoboCup the most popular one [88]. Established in 1997, RoboCup Federation is an international scientific initiative that aims to advance the state of the art in robotics. RoboCup, as an event where mobile robots compete to face different state-of-the-art research challenges, is organized in different categories: home assistive [89, 90], rescue [91, 92], soccer [93, 94], logistics [95, 96] and industrial [97, 98]. Both RoboCup@Work [99] and its counterpart European Robotics League [100] push the idea of a mobile platform integrated with an industrial arm that autonomously schedules a series of picking and delivery orders, planned by a high-level Manufacturing Execution System (MES).

Industry

Nowadays, robotic systems are common elements throughout the entire manufacturing plant. Its implementation in the industry is in constant growth, being a technological key for the Industry 4.0 [101]. Among the different application scenarios, logistics, manufacturing and assembly are the areas that benefit most from the application of robotic systems in industry, since they offer the required flexibility and scalability to integrate modular production and assembly lines [102]. Among the various projects and initiatives in this area, the following should be highlighted: Kitting-bot [103] helps the car manufacturing with collaborative autonomous kitting robots; in TAPAS [104], a mobile robot with a torque-controlled manipulator fetches and transports a rotor to a working station, where it receives help in dealing with real-world uncertainties from an intelligent sensor assistant [105]. Other examples focus on human assistive robot co-workers such as [106], where the cobot is used in screwdriving tasks by means of a skill-based approach.

Logistics

Although it has its own entity as a domain, logistics can be considered transversal and, therefore, with applicability in the previous domains. During the last decade, several commercial solutions for autonomous transportation systems have been developed. The first prototypes used to have movement limitations as they followed magnetic bands, lines or QR codes on the floor. In addition, aside from the time and cost of implementing and maintaining these approaches, robots may have to stop and take a different path, when possible, if an object or operator gets in their way. Newer autonomous transport vehicles, which are usually based on LiDAR sensors to perceive their environment and self localization, can freely navigate without the need of expensive modifications in the working environment. Some examples are given below:

- Amazon robotics, former KIVA [10], which may be the most remarkable autonomous ground vehicle. Successfully implemented in different companies, it is capable of lifting and carrying shelving units in warehouses. The localization has been historically based on QR code markers on the floor and cameras facing them. The multi-robot management is based on a hierarchical traffic-light like control. However, the new prototipe, Proteus, claims to be fully autonomous without the need of any environmental marker.

- KARIS [107, 108], developed at the Karlsruhe Institute of Technology, is an extension of a flexconveyor system [109]. Its goal is to partially replace the current rigid and inflexible roller conveyors and conveyor belts. The localization is based on LiDAR scanners which allow it to drive freely on the factory environment, without the need of additional infrastructure.

- Adept's Lynx OEM, available to developers for custom applications and payloads, uses a LiDAR scanner-based localization with optional sensing systems to improve the obstacle avoidance. The upper section can be replaced depending on the use case and the material to be transported [110].

- Multishuttle Move (MSM) [9], fusion of conventional shuttle and automated guided vehicle (AGV), uses two LiDAR scanners and dead-reckoning for the global navigation, and a rail-guided system in the racking system for load handling. It requires a specific layout of the supermarket area based on a racking system.

- Bosch ActiveShuttle, started under the name Autobod as a co-operation between the company and the Technische Hochschule Nürnberg [111]. The ActiveShuttle Management System presents a partially centralized system for multi-robot coordination.

Many start-ups have also appeared fighting for their own place in this growing market. Companies such as Evocortex [112], for heavy load transportation with an own patented localization system, or Symovo [113], for low load cargo and intra-logistics in small and medium-sized enterprises, offer custom solutions to new or established companies to build their own distributed transportation logistics. Other companies rolling ahead that are developing and offering autonomous transport vehicle solutions are: Omron Adept Technology [114], the TUG of Aethon Robotics [115], IAM Robotics [116], Locus Robotics [117], Mobile Industrial Robots (MIR) [118], Fetch Robotics [119], Magazino's TORU and SOTO [120] and Arculu's AMR arculee [121]. Despite being all excellent transportation systems, these are proprietary commercial solutions and therefore not accessible to the public domain.

2.2.2 Research gaps

The analysis of the different application domains of MMRs makes it clear that they are going to play an important role in the close future, not only in the industrial area, but also in many other fields as seen above. Despite working in more or less structured scenarios and pursuing heterogeneous goals, all the different MMR types share a bunch off common abilities and requirements such as localization, perception of their working environment, navigation and some kind of manipulation skills. Many works are focused on a single objective, either a novel control algorithm or the integration of specific elements. They are also normally focused on the context of the application and, thus, usually generate ad-hoc solutions difficult to extrapolate to other use cases. For this reason, more generic architectures are needed to simplify the integration of the different abilities seamlessly, allowing to execute low-level commands but also high-level plans in a standardized way. These architectures should promote the modularization, reusability and scalability of components, while promoting the use of the latest standard technologies and data information that helps with the digitization.

Recent reviews on system architectures and applications for mobile manipulator robots [122, 123] show that current research is mainly focused on low-level controls for localization, navigation or dexterity tasks, and not that much on the interoperability and reusability of these control and robotic skills. In addition, these reviews remark the need of generic rules to define single robot skills that are reusable, robust to non-deterministic events, and intuitive to non-expert users under a wider context than the proposed use case. Encapsulating robot tasks into skills was initially presented in [124] and proposed as a skill based control architecture with parameterized robot abilities in [125–127].

To design the skills, two main opposite approaches have been followed: bottom-up approach, which focuses on the control and aims at capturing the continuous aspects of a task [128, 129]; and top-down approach, which focuses on the semantic coordination level, while usually abstracting the low level control [130–132]. The latter seems to be suitable for creating generic architectures controllable in a more standardize way. However, after defining and developing these skills for specific contexts, there is still a need to reuse and coordinate them in a wider spectrum applications. For these robot skills to become useful, heterogeneous domain application users, who may not be robotics experts, must be able to instruct them to perform a variety of tasks. With this goal in mind, CoSTAR [133] developed a robust task plan creator for collaborative robots based on behavior trees. Other works in this context are: the proposal made in [134] and validated on simulated humanoid robot platform; the task programming framework proposed in [135], with powerful logging, debugging and profiling capabilities in the context of space missions and terrestrial applications; the Affordance Template task description language presented in [136], to provide high-level augmented reality capabilities to facilitate human-in-the-loop simulation; the ROS Commander (ROSCo) [137], which enables expert users to construct, share and deploy robot behaviors for home robots; AutoRobot [138, 139], that offers a series of reusable packages, templates and tools to help with the integration of new agents within their framework; ALLIANCE [140], a fully distributed framework that delivers abstract function units to facilitate robot behaviour description, and fault-tolerant multi-robot task allocation. An interesting observation is that all these proposals have been built on top of the Robot Operating System (ROS).

In summary, there is a wide range of MMRs that possess diverse abilities but encounter similar challenges. With the increasing demand for this type of robots in the market, numerous robotic platforms and start-ups have emerged, aiming to provide on-demand solutions. However, despite ongoing efforts to develop a comprehensive and standardized framework that facilitates integration and the reuse of various skills, a universally accepted solution is yet to be established. Moreover, most of the existing solutions mainly focus on single robot applications, neglecting the crucial aspect of integrating them into a multi-robot industrial ecosystem, that requires seamless communication with other non-robotic components. Consequently, autonomous transportation systems face new scientific challenges when operating in real industrial environments with heterogeneous sensors and actuators. One of the notable research gaps lies in the field of multi-robot socialization, decentralized multi-robot task allocation, and their integration with other non-robotic Cyber-Physical Production Systems (CPPSs), as demonstrated below.

2.3 Integration of multi-robot systems in a smart factory

The Smart Factory proposes the transition from traditional automation, where a MES centrally governs all the devices in the factory, to manufacturing systems with distributed intelligence that provide a degree of total interconnection. This encompasses a technological development in different areas that include process simulation, unit traceability, predictive maintenance, supply chain integration, product design optimization and intelligent manufacturing, among others. Specifically, the concept of intelligent manufacturing focuses on the development of manufacturing systems capable of dynamically adapting their production to demand, responding to any change derived from incidents in the plant or modifications in the manufacturing plan.

Multi-Robot Systems (MRSs) is a robotics field that studies the cooperation between a set of robots that collaborate or compete in order to perform a task. Its application generally considers problems that need more than one robot to be solved, or even problems that may be resolved with a single robot, but where benefits are obtained through the cooperation of a group of robots as a team. The underlying hypothesis is that MRSs can enable the fulfillment of current market demands, by providing plants with the necessary flexibility to manage production through real-time decision making. Providing architectures, methodologies and resources as a standardized basis for the integration of a MRS in a smart factory can simplify the design and implementation process of manufacturing applications, boosting the adoption of Industry 4.0 solutions in the real settings.

2.3.1 Smart factory beyond Industry 4.0 initiatives

The scientific community's interest in the development of flexible and fully connected systems existed long before the emergence of Industry 4.0 initiatives [13, 141]. However, despite the proliferation of work in academia proposing advanced manufacturing solutions based on reference architectures [142] or proprietary developments [143], the barriers detected by the authors at the beginning of the century [144] and today [145] remain stable: the investment cost required to implement plant transformation, the lack of professionals with experience in the necessary technologies, or the lack of design methodologies, among other causes, continue to hinder the adoption of these principles by the industry.

The emergence of initiatives such as Plattform Industrie 4.0 have sought to reduce this gap by proposing reference architectures based on standards. The authors in [146] compile all the standards that can be applied to ensure the different characteristics that are presupposed to CPS. The authors in [147] defend that the basis for the industry to adopt these new manufacturing systems is that they are smart and sustainable, providing a literature review that remarks that safety is an aspect little worked in the current literature.

The proliferation of similar concepts (Smart Manufacturing, Industry 4.0, Intelligent Manufacturing, etc.) has been identified as a source of noise and confusion that prevents the definition of clear objectives and progress. In this sense, different authors have reviewed these concepts, as well as the main reference architectures, with the intention of unifying the characteristics that they should present, the technologies that enable their development, and the pending challenges to be solved in advanced manufacturing systems [46, 148, 149]. In their conclusions, some of these works point out that reference architectures are incomplete in some aspects and only provide a starting point, although they are fundamental because of the foundation they provide through standards.

Although ad-hoc architectures are still being presented today [12, 143], some authors have started to develop their architectures based on reference models, mainly RAMI 4.0, from different approaches: the architecture presented in [150] was developed by adopting the features of different European projects; the works in [151] and [142] developed two architectures from scratch based on RAMI 4.0; a previously designed advanced manufacturing system was adapted in [152] to make it compatible with RAMI 4.0. The limitations that can be attributed to these works are twofold: on the one hand, some of these works present superficial approaches, where the procedures followed to adapt to the RAMI 4.0 architecture are not described in detail, apart from the mention of a series of technologies; on the other hand, the adoption of the RAMI 4.0 architecture by these works is partial, because the proposed solutions focus on developing and detailing the operation of one of the axes of the cubic model in particular.

There is a certain consensus in considering the distribution of intelligence as a common procedure for adopting the most important concepts of the reference architectures. Thus, there have been significant efforts to shift manufacturing control structures from centralized, hierarchical or modified hierarchical architectures towards heterarchical architectures where the information flow is absolutely horizontal, as a result of the direct communication between entities in the same level. However, despite their apparent high flexibility and adaptability against disturbances, fully distributed decision-making based on the limited information handled by system entities makes heterarchical control systems prone to myopic behavior, because entities have a limited view of the system progress towards its overall objective [153]. Empirical evidence suggests that a hybrid approach, based on a compromise between centralized and distributed control, can improve the control of myopic behavior while maintaining flexibility [154]. In this sense, the authors in [155] propose a distributed system in which the agents are monitored by a system supervisor who ensures that the overall objectives are met, and who only yields control in case of disturbances. In the same vein, the CASOA architecture [156] has a lower layer based on multi-agent systems and an upper layer in the cloud, linked through a specific agent, to optimize the manufacturing plan. As a final example, in the hybrid system presented in [157], the centralized part depends on two agents: the system agent, which centralizes communications, and the directory agent, which exposes system information.

Finally, in [148] it is emphasized that the full development of a CPS requires the principles of service-oriented architectures (SOA). [158] also identify the combination of SOA with holonic systems as the ideal solution for the control of advanced manufacturing systems. In this regard, one of the main initiatives for the adoption of SOA principles in manufacturing systems is the European Arrowhead project [159]. This project presents a framework for the interaction of cloud-based systems. Other efforts include the concept of Cyber-Physical Manufacturing Services (CPMS) [160], consisting of the implementation of CPS using SOA to provide a framework for modeling these entities.

2.3.2 Integration of assets in Industry 4.0

The integration of manufacturing assets has been studied in the context of holonic systems [161, 162] and, more recently, in the context of CPS [163, 164] and I4.0 Components [165]. Some works are aimed at solving specific integration problems and, thus, propose specific implementations for particular case studies [166, 167]. Others, although not specific, focus on a concrete type of asset. In this regard, two types of assets attract most of the attention of researchers: PLCs and robots.

- PLC integration approaches differ on the standards used to perform the integration: some are based on IEC 61131-3 [168–170], whereas others are based on IEC 61499 [171, 172]. These works also present differences on where to deploy the AAS: inside the PLC [170] or on an external node [168, 172].

- ROS is commonly used to integrate robots [15, 173]. However, while some researchers use ROS as middleware between the asset and its corresponding AAS [15], others propose to use ROS for the interactions between AASs, choosing the most suitable communication protocol to interact with each asset [173].

In other works, the focus is not on the type of asset to be integrated, but on which technologies to use for this purpose. In this regard, two approaches stand out above the rest: OPC UA and the industrial agent paradigm.

- The use of OPC UA is promoted in different documents published by Plattform Industrie 4.0. The authors in [174] propose the use of OPC UA for communication between the asset and the AAS, and a combination of AutomationML and OPC UA supported by the IEC 62769 standard for communications between AASs. The proposal in [165] presents a three-layer architecture (field assets, edge AAS deployment, and cloud AAS management) where OPC UA is used to integrate manufacturing assets and to implement their AASs.

- Industrial agents naturally extend MAS with asset integration capabilities and practices [175]. In the agent-based AAS proposed in [176] functionalities are coded as microservices, including an agent-based AAS interface (to interact with other agent-based AASs) and an asset interface (to interact with the asset). The authors in [177] proposed an agent-based solution to establish standard interfaces based on the use of the ISO 9506 Manufacturing Message Specification international standard.

OPC UA allows transferring different types of data structures with ease, but does not innately offer the autonomy and decision-making capabilities inherent to agents. Thus, a large number of authors agree on the use of MAS over OPC UA as a technological resource for deploying industrial agents due to their ability to provide flexibility, reactivity, adaptability and distributed intelligence [178–180]. MAS have been widely used for the development of flexible manufacturing systems with distributed intelligence throughout the last two decades, presenting some reference architectures such as PROSA [162], [161] and, more recently, CASOA [156] or the agent-based architecture proposed by [142]. Most of these proposals are fundamentally based on two agents: product agent (PA) and resource agent (RA) [181–183].

Despite the clear benefits that this technology could bring into production environments, the utilization of MAS technologies in manufacturing is not yet established. Studies about the maturity and utilization of MAS technologies [184] demonstrate that the maturity level for industrial use cases is still low. Despite having developed many prototypes to study the potential of the technology there are only a few real implementations.

The authors in [144] introduce several of these works and summarizes the barriers for the current implementation in production processes of the technology.

- **Integration of assets:** Although MAS naturally stand out for addressing asset integration, they still depend on the characteristics of the technology used for this purpose. Besides, industrial controllers rarely include support for running MAS platforms, although this situation is changing with the emergence of open controllers that bundle a software PLC and an operating system where agents can be deployed.

- **Convergence with standards:** The Foundation for Intelligent Physical Agents (FIPA) standard does not support full interoperability in real-time, distributed control, diagnostics or production management. Additionally, its Agent Communication Language (ACL) contemplates the use of ontologies to ensure interoperability, but improvements in knowledge ontology representation standards are needed.

- **Reliability:** Most demostrators are too simple and do not provide the required reconfigurability that manufacturing processes require. Besides, prototypes use too few agents in compare to what it would be used in a real setting (current platforms do not offer robustness for such number of agents).

- **Cost:** Higher investment needed for implementing agent-based solutions compared to classical centralized solutions. Moreover, the implementation of a distributed and modular system implies the complex task of redesigning the interactions between components for the integration of MAS technologies.

In conclusion, MAS-based approaches show great potential to integrate assets in an easy, fast and reusable way, and to improve the adaptability of industrial dynamic systems. However, the lack of standard methodologies and uses cases in real industrial environments are an obstacle for their immediate implantation. As industry distrusts emergent solutions that have not been proved in real environments, it is important to demonstrate that MAS meet all the requirements attributed to them.

2.3.3 Multi-Robot Systems

The integration of a MRS on a smart factory faces a variety of scientific problems ranging from the definition and assigment of tasks to a set of available robots, to the interaction with heterogeneous CPSs in real industrial environments. Specifically, the coordination and cooperation among robots and other non-robotic agents, also known as *Multi Robot Task Allocation (MRTA)*, is a complex and yet not completely solved research field among robotics that deals with the efficient assignment of a set of tasks to a set of robots [185]. Tasks can be discrete (e.g., deliver-this-there) or continuous (e.g., monitoring a building) and may require single or multiple robots to be executed (i.e., single robot task or ST vs multi robot task or MT). Similarly, each single task robot (SR) can execute as most one task at a time while multi task robots (MR) are capable of executing multiple tasks simultaneously. The present work focuses on the decentralized version of the *Single-Task Robots, Single-Robot Tasks (STSR)* classification.

Traditional control proposals were used to apply the centralized task allocation approaches as they provide optimal solutions to the MRTA problem. However, robustness and scalability problems have made the trend to turn towards more decentralized solutions. In [186], a decentralized market-based approach is presented, where robots bid for tasks based on their capabilities. For the bidding process the Contract Net Protocol (CNP) [187] was used, in which the machine fulfills the role of the coordinator and the MMRs participate as bidders. This multi-agent task-sharing protocol is divided into 4 stages: 1) task announcement by an agent that takes the role of the coordinator, 2) bid submission by individual agents, 3) evaluation and winner selection by the coordinator, and 4) contract stage of the winning agent. In order to increase the robustness, other solutions assign an additional secondary robot to supervise the primary robot and assume its task in case of failure. In order to increase scalability, the auction process can be run locally, taking into account only nearby MMRs, thus, improving networking performance and reducing data processing performed by the MMRs.

Being robotic devices, MMRs can be developed using available RFs, as they provide hardware abstraction and software components for the creation of complex and robust robot behaviors in diverse applications and across a wide variety of robotic platforms. Nevertheless, RFs have been focused on developing single robot functionalities and have not included social abilities within their inherent characteristics. RFs must now evolve aimed at improving social abilities among robots and allowing the interaction of robots with non-robotic entities. To overcome the lack of social abilities inherent to RFs, the work in [1] proposed the combination of RFs with already proven and reliable MAS to create the so-called Multi-Agent Robotic Systems (MARS). MAS technology has been proved as a natural way to meet the socialization requirements among different manufacturing entities in many industrial domains [188]. In fact, the concept of Industrial Agent is related to the implementation of CPPSs as agents [189, 190]. The application of the MAS paradigm to RFs contributes to the agentification of an MMR, a process by which an MMR would become an agent that can socialize with other agents in the factory. The combination of RFs and MAS can lead to a fast developing of reusable individual robotic systems that are coordinated by already proven multi-agent technology. This results in a suitable and modular multi-agent transportation system that is capable to communicate not only among them but also with other components in the production system.

The creation of generic multi-layer architectures for multi-robot collaboration based on RF and MAS integration has been of interest to researchers for decades [1, 191]. AutoRobot [138, 139] combines ROS and JADE to enable support to autonomous and rational service robots. It offers a series of reusable packages, templates and tools to help with the integration of new agents (robots or sensors) within their framework. However, it mainly focuses on single robot requirements, without addressing socialization issues. The MAS2CAR architecture [192] proposes an architecture for controlling and coordinating robots working in teams, where the coordination among agents occurs over a central supervisor component which detects and avoids collision conflicts. Another three-layered architecture is presented in [193] for enabling robotic services in intelligent environments. Specifically, they present the interaction of a mobile robot with smart light and door agents. A centralized component acts both as a global knowledge container and as a central coordinator of other system components.

In summary, the introduction of intelligent machines and transportation vehicles increases the automation and the system reaction to disturbances at the shop floor level, such as changes in the production plan or possible faults in the resources, thus increasing the robustness and adaptability of the factory. The use of intelligent product and order agents increases the traceability and available information at any time which can be used for process optimization or failures forecasting. Despite the noticeable efforts on the research of RF-MAS integration architectures, the decentralization issue remains unsolved, limiting one of the main benefits of using MAS. It is therefore necessary further research to develop a generic RF-MAS architecture to enable intelligent multi-agent networks composed by robots and smart sensors.

2.4 Summary: Requirements for a MRS in a smart factory

As mentioned above, the latest proposals for adaptive and product-oriented manufacturing systems replace centralized control in favor of autonomous, intelligent and heterogeneous production entities capable of cooperating to solve problems. The MMR plays a key role with on-demand material transportation between machines and warehouses, enabling system reconfigurability and supporting operators with complex or hazardous dexterity abilities. Having reviewed the related literature and considering the objectives of the work, it is possible to define the main design requirements of a generic robotic system in a flexible manufacturing process as shown in Table 2.1 and described below.

Table 2.1: Main Requirements of a flexible manufacturing process.

Identifier	Brief description
R1	Alignment with RAMI 4.0.
R2	Distributed control architecture.
R3	Modular design development.
R4	Monitoring and data management.
R5	Socialization among I4.0 Components.
R6	Efficient communication management.
R7	Fault detection and recovery.
R8	Reconfiguration.

Alignment with RAMI 4.0 (R1)

Reference architectures propose a contextualization of Industry 4.0 to ensure an understanding among its participants, providing a starting point for the design and development of solutions on a common basis. The analysis of the related literature shows that RAMI 4.0 is the reference architecture to follow at plant level. However, asset integration in Industry 4.0 is a complex task that involves, asset-related automation, information management and service implementation. This includes the definition of the *Asset Related Services* regarding the MMR (i.e., *Application Relevant Services* that provide mobile and/or manipulation functionalities). Once the characteristics of MMRs as I4.0 Components in the factory have been defined, methodological support is needed to guide MMR integration into an I4.0 System. To that end, the concerns related to MMR integration must be abstracted into layers, so that they can be addressed both generically (i.e., regardless of the type of MMR to be integrated) and independently (being decoupled from each other). In addition, technological support is needed to facilitate the development of AASs for MMRs based on patterns that include the necessary *AAS services* that can be extended and customized according to application requirements.

Distributed control architecture (R2)

Distributed systems are designed to manage changing environments where intelligent and loosely-coupled software components do have to interact with each other to perform tasks. In a MRS, it is important that the control architecture distances itself from the traditional ones linked to the state of a single component or controller (such as centralized, hierarchical or modified-hierarchical). In this sense, each robot should be able to perform its tasks autonomously or quasi-autonomously from the exchange of information with other robots. Furthermore, it is important to note that robotic processing needs can be complex, being able to perform low-level control and high-level social communication in parallel is also considered necessary, so that there are no delays and problems in functionality. Nevertheless, the complete distribution of the control leads to a local information-based decision making, resulting on a locally optimized solutions and a non-predictable global behaviour. The solution to this problem involves the development of hybrid solutions that combine the benefits of centralized systems and distributed systems, embodied in a SOA that reflects the service model defined in RAMI 4.0 (R1).

Modular design development (R3)

The design of modular and structured software systems allows highly complex problems to be solved by dividing them into smaller parts that can be developed, tested and modified much more easily without the danger of affecting the rest of the application. For this, it is necessary that the software units or modules have high cohesion and low coupling, that is, they have the least possible dependencies. In the case of MARS, the generation of modules that avoid having to adapt the functionalities to the hardware, would allow the functionalities of the system to be portable. In this way, it is only necessary to adapt or rectify the hardware access units, which must share a common interface. As a consequence, the scalability, extensibility, maintainability and portability of the system are improved. The addition or removal of modules on the system does also not affect other modules. Also updates can be done individually for testing purposes or as a group to include new functionalities or solve known bugs on the whole fleet. Finally, being modules independent from each other, a failure of a module must not affect any other module, increasing its robustness.

Monitoring and data management (R4)

Agents must update information about their operation and events occurred. This information is monitored by the system and used by operators for visualizing the state of the system, for historical traceability, for rapid detection of failure situations and reaction, and, finally, for data analysis such as optimization and forecasting.

Socialization among I4.0 Components (R5)

Sociability is the ability of two or more systems, applications or components to exchange information or manage services that they both know. For this, it is necessary that all of them agree on a method from which its components can be understood. Based on this information exchange, I4.0 Components make autonomous decisions, allowing a decentralized control architecture. This requirement involves the definition of the interface or API of the MMR as an I4.0 Component, through which it interacts with the rest of the I4.0 Components in a I4.0 System to perform requests for its Application Relevant Services. In the case of robotic systems in an industrial environment, it is important to be able to use a language that allows robots to communicate with each other, but also that they can communicate with machines, orders, operators, etc. This implies that interoperability standards in socialization must go beyond robotic platforms.

Efficient communication management (R6)

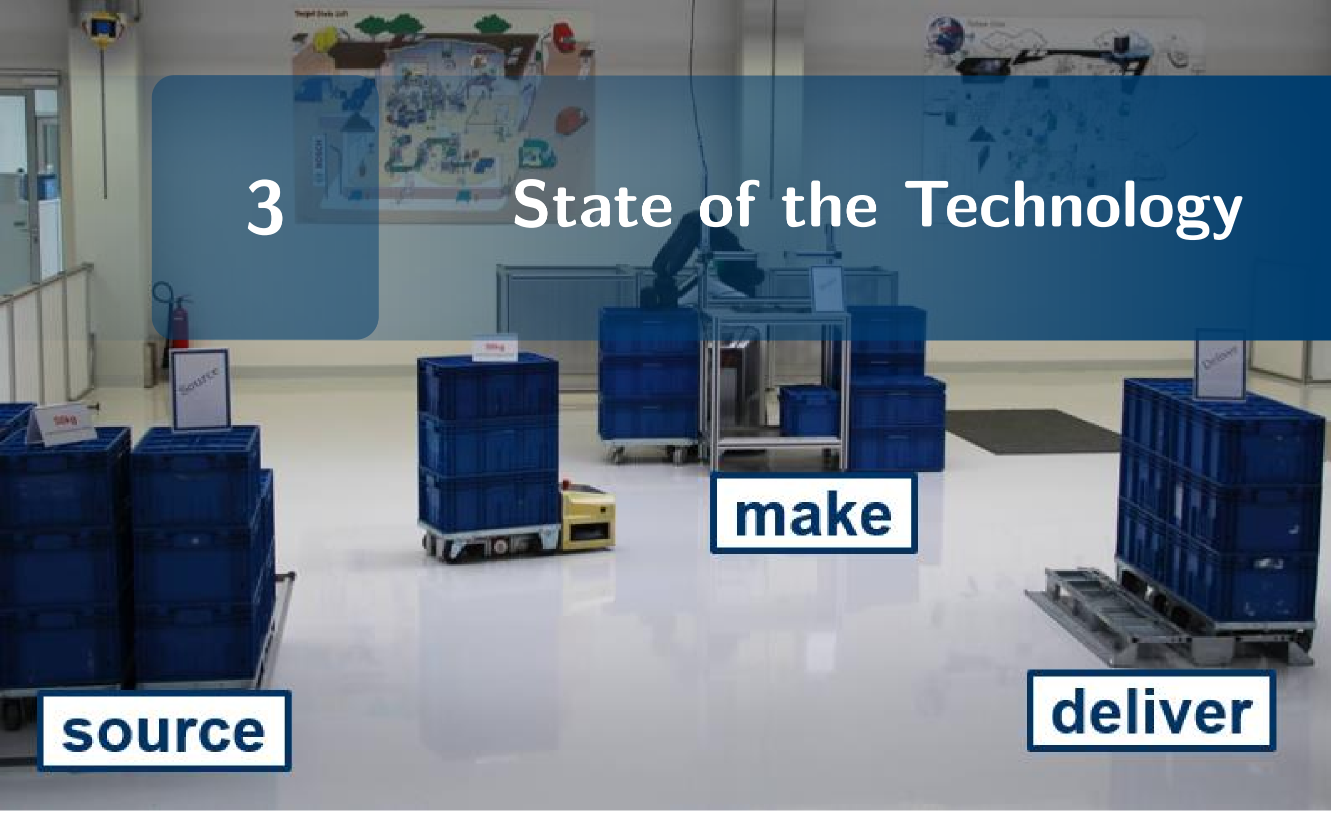
In a manufacturing system we have a large number of components that must communicate over the network. If the interaction does not have established rules so as not to saturate the system, it is possible that the functional time restrictions cannot be met due to the long delays that the system accumulates.

Fault detection and recovery (R7)

When failures occur in the software or hardware of any component, the system may loose part of its information or receiving it erroneously. The design must be robust and fault tolerant, including error detection and recovery or, at least, operating in degraded mode in unforeseen situations.

Reconfiguration (R8)
Also, changes in the manufacturing plan or the inclusion of new products must be faced by a rapid and efficient reconfiguration of robotic components, machines or additional software modules. Although a fault or an unforeseen reconfiguration implies delays, robots must minimize the effects that these may have on the global system.

In order to adapt the thesis to actual research problems and necessities, this work presents two main contributions: First the development of a generic multi-robot architecture that facilitates the integration of single robot capabilities to work on a wide range of mobile manipulation robotic applications. The architecture will be validated on both industrial and non-industrial robotic use cases, within different European projects. Second, its integration on the flexible manufacturing, using the latest IT technologies and industrial standards.



3 State of the Technology

Taking into account the aforementioned requirements for the design of MRS systems in Industry 4.0 environments, this chapter identifies the available frameworks that support the required technologies, and analyses their characteristics for this purpose. During the analysis, the weaknesses and strengths of these frameworks are identified, and the benefits that can be obtained by integrating them with each other. In addition, this analysis focuses on those particular concepts that are considered essential to understand the design proposed and tested in this thesis.

3.1 Analysis and selection of tools

There are many and different frameworks that allow creating new robotic applications in a fairly simple way. Over the years, there has been a shift from developing software completely dependent on hardware to seeking alternatives, not yet standardized, that provide solutions to the most recurrent problems in this area of knowledge. In this sense, a large number of environments and libraries have emerged. In the following, RFs, OPC UA and MAS will be analysed.

Component-based frameworks are the most popular robotic software development approach as they simplify the development process by modularization and heterogeneity. Some have been popular in the past but are already deprecated such as ORCA [194], Microsoft Robotics Studio (MSRS) or Player/stage [195]. Other popular and still maintained component-based frameworks are Yet Another Robot Platform YARP [196], ArmarX [197], OROCOS, Open-RTM [198] and ROS [199]. Simultaneously, modelling and simulation frameworks permit their collaboration with some RFs. That is the case of Webots [200], V-REP (Virtual Robot Experimentation Platform) [201] or Gazebo [202]. In addition, frameworks such as the Open Robot Middleware Framework (RMF) [203] enable interoperability between multiple fleets of robots and physical infrastructure, like doors or elevators, and are also seamlessly integrable with the previously mentioned RFs.

In modern manufacturing facilities, PLCs play a central role as low-level intelligent devices in production machines [204]. To permit the communication of robots with manufacturing machines, the OPC UA standard has been considered as it is the de-facto standard middleware for seamless control of PLCs from diverse vendors [205].

In order to enable the use of the OPC UA system architecture with as many devices, components, tools and manufacturers as possible and outside of classic industrial automation area, the OPC Foundation decided to make its middleware specifications and stacks available as open source. Since then, several libraries that provide OPC UA functionalities have gradually appeared on the network. Some companies offer their professional paid products for experienced developers who want to develop their system independently [206, 207]. However, in addition to the paid products, there are also many different free open source middleware architectures such as node-opcua, EclipseMilo, Opcua4j. These cannot usually compete with larger companies in terms of functionality and support, but they can still offer the opportunity to successfully implement smaller OPC UA solutions and start in the OPC UA area. An extended list of different architectures can be found in [208] and be used by interested people who are considering entering this market. Because of its documentation, ease of use and popularity, the open62541 [209] and FreeOpcUa [210] implementations are the ones that will be considered in this work.

Finally, MAS frameworks are standard tools created by researchers, developers and companies to facilitate the implementation of heterogeneous agents. Although MAS technology is not new, its implementation is still in its early stages. In this sense, it was not until the beginning of the 2000s that this type of platforms began to be developed, highlighting names such as AgentBuilder [211], JADE [212, 213], Jason [214] or MaDKit [215].

The diversity of alternatives and application scenarios makes it difficult to choose the optimal one for each aspect. Thus, the selection is supported by studies that evaluate and detect the desirable characteristics (C) of available RFs [1, 216–218] and MAS frameworks [219, 220]:

- **Support Sensors/Actuators (C1)**
  The management of the low-level hardware interfaces and drivers cannot be ad-hoc for each application. It is desirable the support for a large number of sensors and actuators, as well as their innocuous interchangeability without affecting higher level functionalities.

- **Provide Robotic algorithms (C2)**
  Existence of generic robotic functionalities that allow managing low-level actions (control of actuators and sensors) and high-level actions (task planning, navigation, etc.).

- **Provide simulation and modelling tools (C3)**
  Availability of model generation tools and executable specifications. Simulation can help saving time and costs by training and testing in advance, without damaging the working space.

- **Based on a standard (C4)**
  Standards are crucial as they facilitate interoperability, compatibility, and seamless integration between technologies. They encourage innovation, competition, and best practices while enhancing reliability.

- **Supports platform security (C5)**
  Choosing frameworks with built-in security features or the flexibility to develop them on-demand is crucial for deploying these technologies in industrial environments, ensuring protection against threats and vulnerabilities.

- **Open Source and active community (C6)**
  Free frameworks with active user community promotes the engagement of the research community and much faster evolution or problem solving.

- **Multi-platform (C7)**
  Platform abstraction layer that allows specific communications from peripherals, work in real time, in multitasking, etc.

Table 3.1 groups the most popular, open source (C6) and multi-platform (C7) robotic, machine and MAS frameworks, identifying which of the mentioned requirements and desirable characteristics do they cover. In the following, an analysis of the selected tools is presented.

Among RFs, it is clear that all of them provide mechanisms related to the support of physical sensors and actuators (C1) from a reusable perspective. Being component-based, they also follow a distributed and modular architecture, ensuring compliance with the requirements (R2) and (R3), respectively. Most of them provide certain level of robotic algorithms support (C2) and simulation tools (C3). They also provide mechanisms for monitoring and visualization (R4), despite they need to be further enhanced in order to save data on databases and detect desired anomalies or events. These monitoring tools should be enhanced by developing quality of service network management or overall system diagnostics and error management tools (R7). Factory set up reconfiguration (R8) or software updates are also possible thanks to the modular, plug-and-play design, replacing one robot by another or including new ones on demand without affecting the rest of the system. However, none has been developed considering intrinsic social capabilities that allow robots to establish cooperative solutions (R4), making it clear that it is not possible for a robotic platform to solve the complete problem on its own. This lack of socialization complicates their direct integration on the factories of the future (R1) and new mechanisms need yet to be developed. Finally, despite some of them are based on the Common Object Request Broker Architecture (CORBA) specification, it cannot be considered as the overall robotic communication standard as there are other middlewares more popular and with more active community than OpenRTM-aist and OROCOS.

Table 3.1: Robotic Machine and Multi-agent frameworks comparison

	(R1) RAMI Alignment	**(R2)** Distributed	**(R3)** Modular	**(R4)** Monitoring tools	**(R5)** Enable Socialization	**(R6)** Efficient communic.	**(R7)** Fault Detect.&Recov.	**(R8)** Reconfiguration	**(C1)** Support Sens.&Act.	**(C2)** Robotic Algrtihms	**(C3)** Simulation/Modelling	**(C4)** Standard based	**(C5)** Security	**(C6)** Open Source	**(C7)** Multi-platform
ROS	L^2	✓	✓	✓	NO	L^3	M^3	H^1	✓	M^1	Gazebo, Stage	NO	NO	✓	Linux,OSX, windows..
YARP	L^2	✓	✓	✓	NO	L^3	L^3	M^1	✓	M^2	iCub, Gazebo	NO	NO	✓	Linux,OSX, windows...
OROCOS	L^2	✓	✓	✓	NO	L^3	L^3	L^1	✓	M^2	NO	OMG-CORBA	NO	✓	Linux,OSX, windows...
OpenRTM-aist	L^2	✓	✓	✓	NO	L^3	M^3	M^2	✓	M^2	Open HRP3	OMG-CORBA	NO	✓	linux windows
Agent Builder	H^2	✓	✓	✓	✓	M^2	H^1	H^1	NO	NO	NO	KQML	L^3	✓	linux, windows
JADE	H^2	✓	✓	✓	✓	H^1	H^1	H^1	NO	NO	NO	FIPA-ACL	H^1	✓	any with JVM
Jason	H^2	✓	✓	✓	✓	M^2	M^2	H^1	NO	NO	NO	FIPA-ACL	M^2	✓	any with JVM
Madkit	H^2	✓	✓	✓	✓	M^2	M^2	H^1	NO	NO	NO	UML	H^1	✓	any with JVM
open62541	H^2	✓	✓	✓	NO	L^2	M^2	H^1	✓	NO	NO	OPC UA	H^1	✓	Linux,OSX, Windows
FreeOpcUa	H^2	✓	✓	✓	NO	L^2	M^2	H^1	✓	NO	NO	OPC UA	H^1	✓	Linux Windows

1 High Support.
2 Medium Support.
3 Low Support.

An evaluation of existing RFs and the need of robotic benchmarks and standards is done in [221]. Here, ROS appears as the most popular and best suited framework to become the standard and YARP, which is more focused on humanoid robots, as the second most popular one. ROS is the framework with the brightest present with an increasing number of citation of over a 25% yearly, and an accumulated increase of 250% since 2017 (9260 citations in 2021 against the 3704 citations in 2017) [222]. Also, many companies have trusted on ROS with over 200 robots integrated on the ROS infrastructure, including industrial robots manufacturers that actively support the *ros industrial* flavour of ROS [223]. This allows them to ROSify their industrial robots and components, and re-use all the software created in the ROS ecosystem.

Taking into account that several of the RFs have very similar characteristics, it has been decided to analyse and use ROS as the reference RF for development in this work. Its popularity and the enormous variety of available add-on packages make ROS a highly flexible and interesting middleware for a wide variety of robot applications. With the development and release of ROS2, this middleware will be able to assert itself on the market in the long term.

With regard to MAS frameworks, the intrinsic characteristics of practically all the frameworks analysed and the standards on which they are build upon, meet the requirements R2-R5 without the need for any additional development. Besides, all of them have some kind of network analyser and manager to react in case of network overload (R6). In addition, the requirement R7, despite not being specific to multi-agent architecture, is partially met by platforms that, based on MAS frameworks, add the necessary mechanisms to include this fault detection and recovery characteristic [224]. Being designed upon industrial standards (C4), they also support some kind of security directives (C5). On the other hand, since C1-C3 are strongly related to the hardware of physical systems, the multi-agent paradigm does not naturally support them.

Thus, it can be concluded that the addition of the multi-agent paradigm in RFs could be a step forward when it comes to assist socialization, network management, recovery and security situations [1]. However, for both frameworks to cooperate, it is necessary to define an integration architecture that allows, on the one hand, to solve the communication problems between them, and on the other hand, to allow the separation of concerns between them, allowing specialists in different areas to work together.

Specifically, in the present thesis it has been decided to use the JADE multi-agent framework. Its high performance, ease of use, robustness and use of the most widespread standard norms, makes it the most suitable solution for the problem under study. Its architectural design has been articulated mainly on the basis of two important decisions: choice of programming language and standard reference norm. With regard to the programming language, cross-platform support has been sought. To that end, Java was chosen, an object-oriented programming language that, when compiled, generates code that must be interpreted during execution time by a JVM (Java Virtual Machine) dependent on the target platform. In this sense, any operating system capable of running a JVM virtual machine is able to implement a MAS in a simple way. With regard to the standard reference norm, the framework is based on FIPA. In this way, JADE can communicate its agents with others implemented on another platform that follow the same standard.

The integration of the different robotic and MAS frameworks into a RAMI-based flexible manufacturing leads to an agent-based robotic architecture where robots or robotic components appear as agent assets

(R1) that interact or cooperate with other non-robotic agents, providing larger support for autonomy and increasing their socialization capabilities. As seen in figure 3.1, RFs provide the support to hardware drivers, simulation and development tools and robotic algorithms, while MAS frameworks support the needed socialisation mechanisms. [1] explains the characteristics and necessities for MARS and compares available and mature Robotic Software Frameworks and Multi-Agent System Frameworks. Among the agent-based robotic architectures, the following stand out: PANGEA [225], an organizational-based platform for sensing devices with resource constrains; COROS [226], a multi-agent software architecture for cooperative and autonomous service robots; a three layer architecture for service robots in intelligent environments [193]; and the AutoRobot framework [139], that supports autonomous and intelligent service robots. Unfortunately, these architectures are not yet well established and accessible to be used. In addition, the architecture has to be completed with a careful analysis of the application domain to achieve stable interfaces and component re-usability [227]. These works are mainly designed for service robotics, overlooking some important industrial requirements and recommendations such as integrating the robots in the CPPSs concept [21, 163, 164] or the I4.0 Component [165], or the use of standard communication interfaces like OPC UA.

In this regard, OPC UA frameworks can be complementary enablers to communicate machines, usually controlled by PLCs, with the robots and other manufacturing systems. Among the frameworks evaluated, even if open62541 and FreeOpcUa satisfy the same range of requirements and characteristics, they differ significantly in their implementation. Due to its simplicity, easy of use and object-oriented development, FreeOpcUa is better suited for beginners than open62541. Another important point against open62541 is that the library may only be used in projects that do not belong to the open source category. Finally, it already exists a ROS

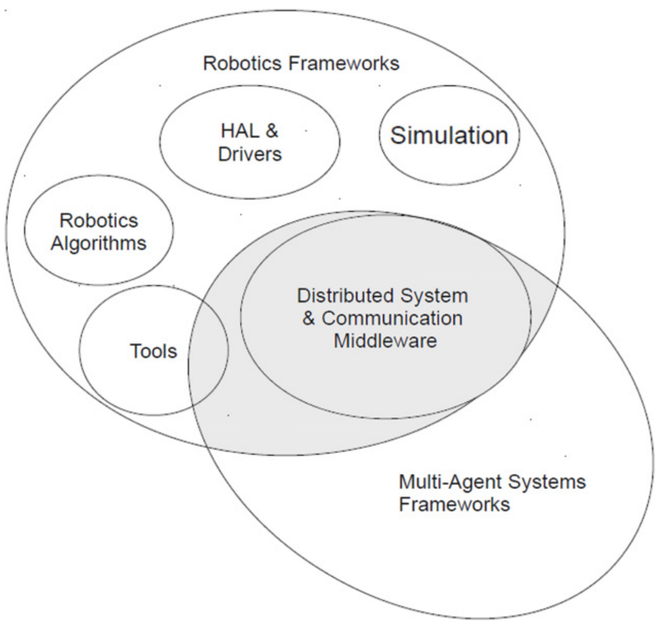

Robotics Frameworks
HAL & Drivers
Simulation
Robotics Algorithms
Tools
Distributed System & Communication Middleware
Multi-Agent Systems Frameworks

Figure 3.1. Robotic and multi-agent system frameworks synergy [1].

package that implements bindings for FreeOpcUa Server and Client Libraries supporting C++ [210] and Python [228] programming languages. For these reasons, the selected OPC UA middleware is FreeOpcUa, enabling the distributed interaction among robots and machines. Nevertheless, it should be noted that the middleware open62541 shows more potential in the long term: since several companies are involved as cooperation partners in its architecture model, this system will be constantly further developed, and various improvements and extensions will be implemented. In any case, the integration of a different open source OPC UA-based framework should not involve much work.

3.2 ROS (Robot Operating System)

ROS [199] is one of the most widespread and popular RF with a global and active community. It is considered an open source meta-operating system that provides the minimum services of an operating system: hardware abstraction, low-level device control, implementation of common utilities, message passing between processes, and package management. Among its features, it contains a large number of tools and libraries that facilitate its use in various applications, without the need for additional efforts. In this sense, the operating systems supported by ROS are those most widely used for computer processors (Linux, OSX, Windows, as well as various embedded platforms) and the programming and development possibilities are also varied (C++, Python, Octave, LISP and even JAVA), facilitating the integration work. It is also commercial friendly, as it is distributed under permissive open source licenses, with Apache 2.0 being the default one. All these issues are what make it one of the most used robotic frameworks for the development of applications of this type.

The ROS design is completely distributed, isolating small processing units that exchange information. To do this, it has various levels of concepts. The *Filesystem concepts level* mainly covers ROS resources that are encountered on disk, such as: packages, repositories, message (msg) types and service (srv) types. The *Computation Graph level* is composed by the ROS network processes that are processing data together allowing the correct functioning of the whole. The most important ones are illustrated in Figure 3.2 and briefly described below:

- **Node:** Nodes represent modular computation processes. A robot control system is usually composed by many nodes. For example, one node controls a laser sensor, another one controls the localization algorithms to define the robot position, another one calculates the best route on a map, and a final one controls the wheels of a robot to follow the path. The ROS nodes communicate with each other by sending messages by the means of message patterns defined by the platform.

- **Parameter server:** Global information server that allows the processing units to be parametrized in a generic way. In this sense, the parameters stored in this central location can be modified during execution, changing certain variables or operating modes of whoever uses them. Although ROS proposes that it is an independent entity, nowadays it is part of the ROS Master node.
- **Message:** Messages are used by nodes to communicate with each other. A message is simply a data structure, comprising typed fields. The information stored in each message is configurable, with simple messages (integers, character strings, etc.) and complex messages (structures, other messages, etc.).
- **Topic:** Publisher-subscriber is the message pattern per excellence of ROS due to the speed and simplicity it entails. It is a unidirectional message transmission channel opened for writing (publishers) or reading (subscribers). This way, it presents an asynchronous solution for updating the information of many processing units at the same time, without the need to open and close the connection between them on each occasion. The name of the channels and the messages they transmit have to be correctly defined by design, since they are not subject to be modified during execution.
- **Service:** Request-reply is the message pattern used for those interactions that cannot be resolved through topics. It is a synchronous model through which two nodes interact based on events. The identification of the services in the ROS system must be unique, and there cannot be two collaborative nodes that offer a service with the same name. A connection is established between nodes

each time a service is requested, following the next steps: connection request, sending of messages by agreed protocol, and connection close. As long as a request/response model or direct communication with the other node or processing unit is not necessary, services are less efficient than topics.

- **Action:** A client/server model that allows services to be carried out asynchronously. It is not a native ROS communication, although the implementation and use of the *actionlib* library among the developer community is very extensive, and today it is considered the third method of communication on the platform. The asynchronous implementation is based on the use of bidirectional topics, ensuring the existence of a response from the server in all cases. This solution was created to allow execution units to use services without blocking their tasks to completion.

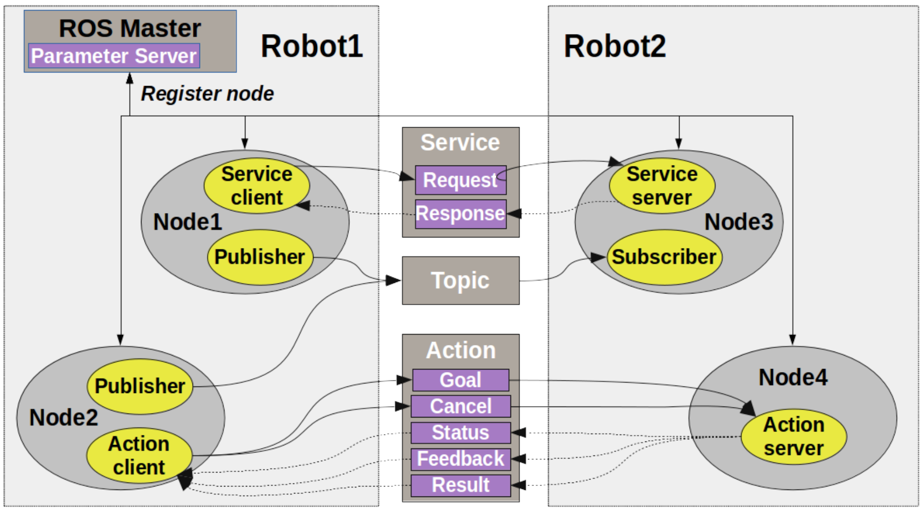


Figure 3.2. ROS Computation Graphs Components and interaction mechanisms

After briefly analysing the most important components and utilities of this distributed architecture, it is important to analyse the possibilities offered by this robotic framework in multi-robot or robot-human applications. For this, the native characteristics of ROS and the possibilities generated by some developers have been analysed.

ROS itself allows multiple nodes to be run on different computers or machines, as long as they share the same ROS Master node. In this way, there is a general system address book that can be accessed by all the nodes, establishing unique communication paths between them. This allows a robot to interact directly with its sensors, an external computer to monitor information from a robot at runtime, and even multiple robots to interact using the same Master node. However, there are also external alternatives that propose multimaster architectures to solve the problem of multi-robot communication. Next, the multi-robot possibilities that can be implemented on ROS are described:

- **Mono-master architecture:** Completely centralized control architecture in which a single ROS Master node acts as the address book for the entire system. In case of multi-robot communication, this Master node usually runs on a workstation known to all robotic systems. This multi-robot design is very inefficient and unscalable. The dependence of the system on a single node implies a traditional centralized control, with all the problems that this entails. Besides, since all the nodes of all the robots communicate through the network, it is possible that there may be saturation problems in wireless and even wired networks.

- **Multi-master architecture:** The ROS user community has sought to provide a solution, based on external libraries, to the problems that the use of a single ROS Master node had. These architectures define their own Master node in each robot and generate communication methods between these system nodes. Although there are multiple ROS packages that allow you to customize this architecture, the most prominent is *multimaster_fkie* [2], whose node architecture is illustrated in Figure 3.3. This package supports its functionality in three of its own nodes:

- **master_discovery:**
  Node that seeks to find new Master nodes in the multi-robot network. To do this, it sends periodic multicast messages to a group, hoping to get a response.
- **master_sync:**
  Node that synchronizes the local Master with the remote ones detected from the master_discovery. The topics, services and parameters of the remote Master nodes are synchronized with the local one, generating a transparent communication between Masters.
- **node_manager:**
  Node that allows managing and configuring the ROS nodes of the different ROS Masters at the same time. Its main functionality is the facilitation of tasks for operators during the testing and commissioning phase of a piece of equipment.

The ROS master has a *registration* API which allows the registration of nodes, publishers, subscribers and service providers. The registrations are done individually, so if one publisher or service fails it does not affect the subscriber or client, or vice versa. When a new publisher-subscriber or service-client match happens, there is a XMLRPC interaction between the master and the interested nodes, where the latter are notified about information such as the IP and port of corresponding partners. The interested nodes can now establish a peer-to-peer communication with corresponding partners. In the publisher-subscriber case, this connection is held on until one of the participants interrupts it. However, in the service-client case the XML data exchange with the master must be repeated every time. Therefore, two main issues can be expected in a MAS with a single ROS master: first, all registrations and request are managed by a single API resulting on a bottleneck; second, agents running on an external device other than the master must register and call all their topics and services over the network, resulting in slower response times and network overload. To overcome these problems, a multi-master approach should allow every agent to poss its own master, distribute the registrations locally and still be able to exchange information between agents.

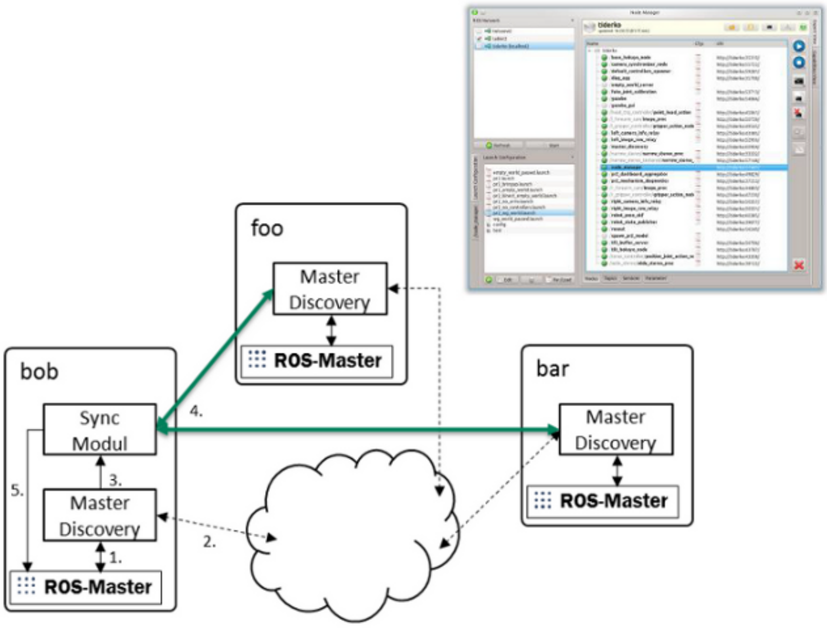


Figure 3.3. ROS multimaster configuration [2]

3.3 JADE (Java Development Framework)

JADE is a software platform that simplifies the implementation of any type of distributed application that is based on MAS technology. It can be considered as the most popular multi-agent platform as it offers the basic features expected in a middleware of this type and some added features that make its use, debugging and maintenance very easy. Among these added features, JADE allows creating graphical interfaces to monitor the operation of a specific agent, the messages that are exchanged between the agents, sending messages in a simple and graphical way, even creating an operator agent that interacts with the rest of agents from buttons [229].

JADE defines the distributed architecture of the Figure 3.4, where it is detailed how different hosts can contain agents, as long as there is one on the same platform that hosts the main container functionality defined by FIPA. The following specific entities must be run in this parent container on a mandatory basis:

- **Agent Management System (AMS):** It deals with access control and provides a white page service. To do this, it registers and monitors all the platform agents, that is, those belonging to the main container and the remote containers. This agent is automatically launched with the main container, since without it, no other agent would exist.

- **Directory Facilitator (DF):** It provides a yellow page service where the services offered by the different agents are registered, allowing any agent to quickly find the agents providing them. By default, a parent container must contain a DF as defined by the

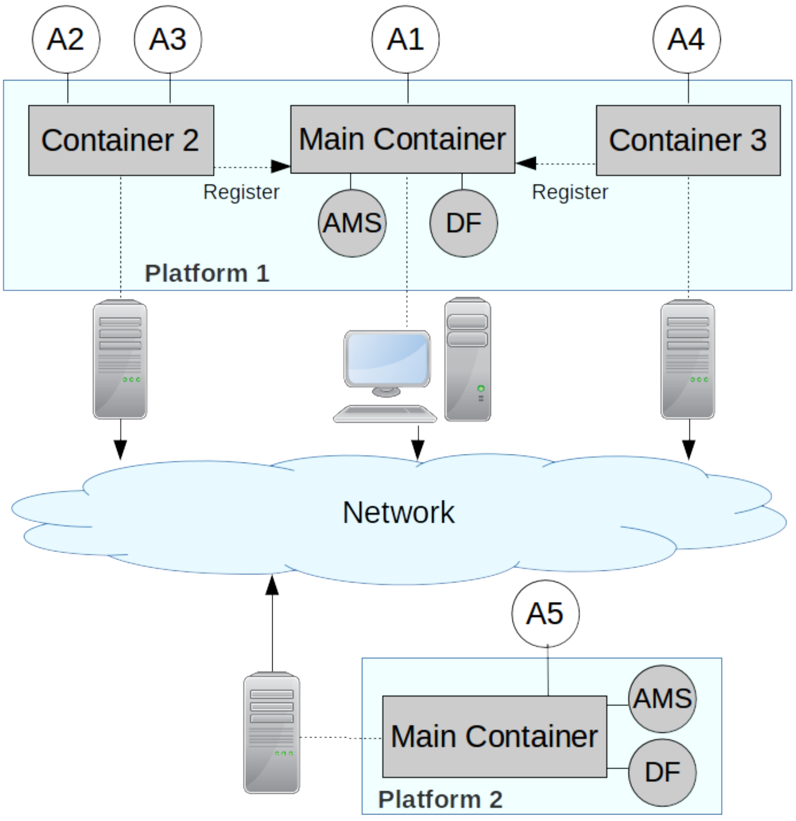


Figure 3.4. JADE architecture overview with containers and platforms

standard, but it is possible to launch more than one to distribute services over the network.

- **Agent Communication Channel (ACC):** The communication channel between agents that controls the exchange of messages (local and/or remote) within the platform. It is equipped with all the tools to manage asynchronous messages of the typologies defined by the FIPA ACL (Agent Communication Language) specification.

Once the elements that a FIPA-compliant MAS platform must contain are known, it is important to analyse the essential aspects that characterize a JADE agent (see Figure 3.5). An agent is actually a Java object that runs in a thread and is registered in the AMS with a unique and universal identifier. An agent is made up of different generic states that it can go through throughout its life-cycle. Agents take advantage of their social capabilities to communicate by exchanging asynchronous ACL messages. This language provides a standardized set of performatives or communicative acts that allow specifying predefined sequences of messages that can be applied in various situations that share the same communication pattern. In this sense, FIPA Contract Net Protocol (CNP) solves the "task delegation" problem with the following pattern: 1) task announcement by an agent that takes the role of the coordinator, 2) bid submission by individual agents, 3) evaluation and winner selection by the coordinator, and 4) contract stage of the winning agent.

The functional tasks of the agents in JADE are carried out from the definition of behaviours. These represent small tasks that allow the achievement of partial or complete objectives of the agent. Its management is carried out from a scheduler that orders the execution of the behaviour queue following a round-robin policy without priorities. The agent is, therefore, capable of performing its tasks autonomously thanks to the correct definition of its behaviours. Every time an agent receives a message, it is stored in the message queue. If the agent is active, the message is not processed until it reaches the appropriate code section for that purpose. If the agent is blocked, the entry of a message in the queue unblocks it by restarting the execution of the behavior queue.

Finally, JADE provides a gateway mechanism to communicate a MAS

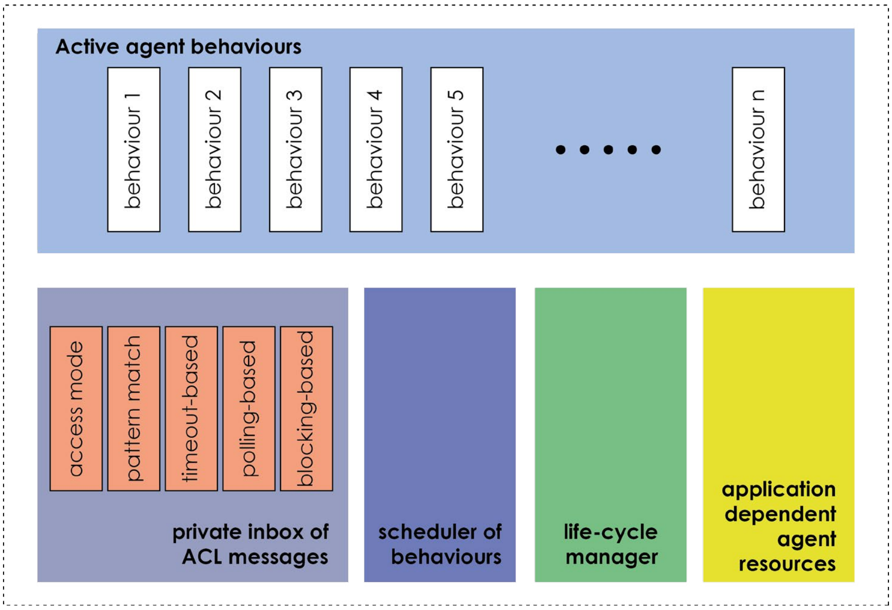


Figure 3.5. JADE generic agent

environment with other types of non-MAS environments. This gateway is composed of two elements: the *JadeGateway* class and the *GatewayAgent* class. The JadeGateway class is used in the non-MAS environment to instantiate an agent derived from GatewayAgent (*JadeGateway.init(myGatewayAgent)*) and to execute commands (*JadeGateway.execute(command)*) in that agent. To that end, the GatewayAgent defines a *processCommand()* method to execute the commands received from the JadeGateway. Commands are data structures that can be used in the non-MAS environment to send data to the MAS environment (i.e., from the program running the JadeGateway class to the GatewayAgent) and/or to receive data from it (i.e., from the GatewayAgent to the program running the JadeGateway class).

3.4 OPC UA (Open Platform Communication Unified Architecture)

In 1995, the "OPC Working Group", led by industrial companies, such as Siemens AG or Rockwell, developed a standard for real-time data access under Windows operating system. However, being limited to Microsoft's operating systems, the OPC Foundation considered adapting the middleware to create a standardized architecture: the "Unified Architecture", also known as OPC UA. This model standardizes the exchange of data between software applications, regardless of manufacturer, programming language, location or operating system. Therefore, OPC UA complements the existing OPC industry standard with essential properties such as platform independence, scalability, high availability, Internet capability, and others. Like its predecessor, the new OPC UA architecture was standardized and published under the IEC-62541 series of international standards [230]. Due to its cross platform communication functionality and multiple features, OPC UA has become an important middleware to integrate new technologies into industrial applications [205].

OPC UA is a unified interoperability model focused on data acquisition, information modeling and communication (between plant and applications). OPC UA is composed of multiple *clients* and *servers* that exchange information using either a client/server or a peer-to-peer distributed architecture model, or a publisher/subscriber or a request-response message pattern. Each client may interact concurrently with one or more servers, and each server may interact concurrently with one or more clients.

OPC UA Client

The Client Application implements the client's function within the client architecture shown in Figure 3.6. The Client API is used to send and receive OPC UA Service Requests and Responses to the OPC UA Server. In this sense, the API is an internal interface that separates the application code from the OPC UA *Communication stack*. This converts client API calls into messages, and sends them as a client request to the OPC UA server using the subordinate communication unit. Similarly, the communication stack receives response and *NotificationMessages* from the underlying communications entity, and delivers them to the Client application through the Client API.

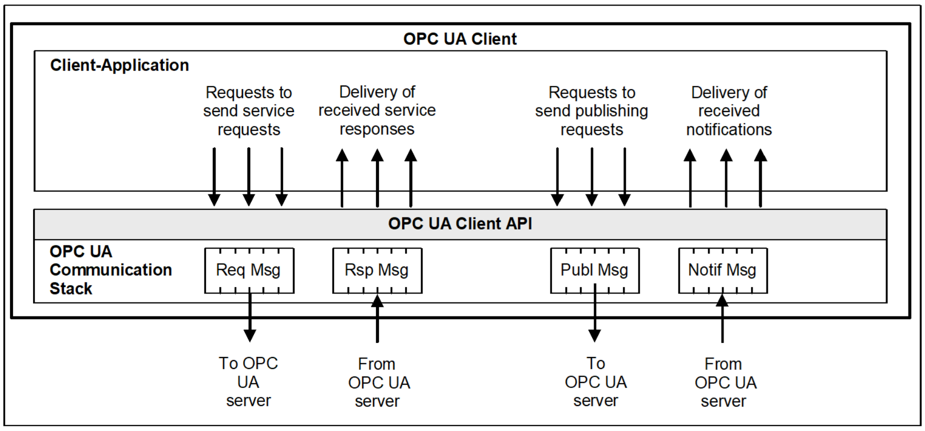

OPC UA Client
Client-Application
Requests to send service requests
Delivery of received service responses
Requests to send publishing requests
Delivery of received notifications
OPC UA Client API
OPC UA Communication Stack
Req Msg
Rsp Msg
Publ Msg
Notif Msg
To OPC UA server
From OPC UA server
To OPC UA server
From OPC UA server

Figure 3.6. OPC UA Client architecture [3]

OPC UA Server

Since an OPC UA server is the center of an OPC UA system, it contains a complex architecture that is designed to meet all requirements. In order to be able to understand this model (represented in Figure 3.7) it is necessary to describe important elementary contents from it in more detail.

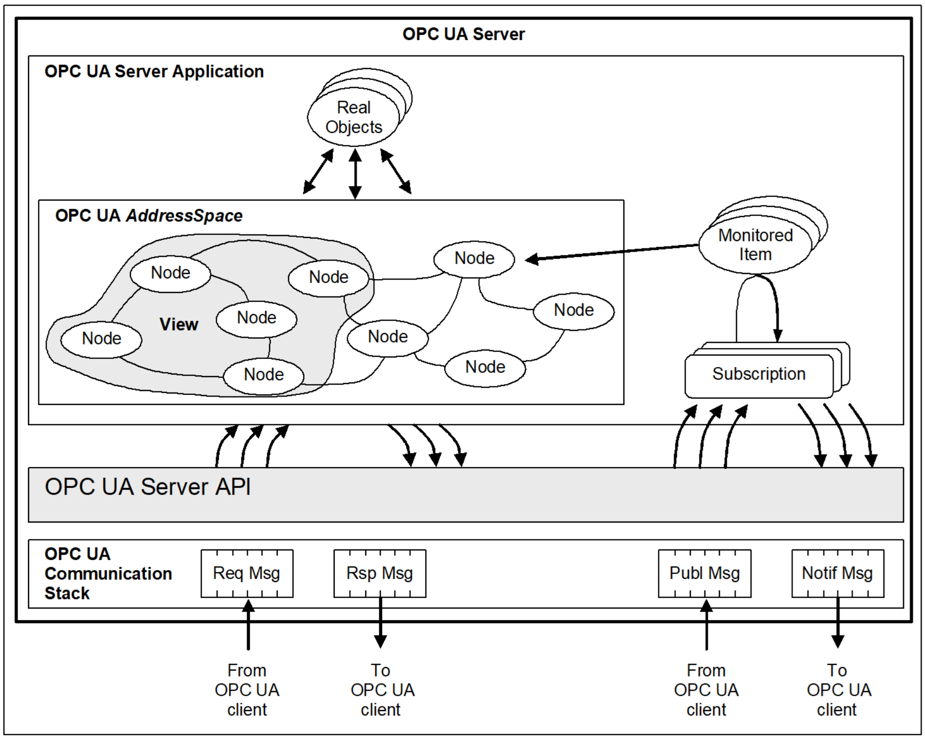


Figure 3.7. OPC UA Server architecture [4]

- **OPC UA Server Application:** The Server Application implements the actual functions of the server. To do this, it uses the OPC UA Server API to send and receive OPC UA messages to and from clients. As with the client, this is an internal interface that separates the communication stack from the server application.

- **Real Objects:** Real objects are physical or software objects that can be accessed via the OPC UA Server Application or that are managed internally by the server itself.

- **OPC UA AddressSpace:** The address space of an OPC UA server is one of the most important components of this architecture. It contains the so-called *nodes*, that the client can access. These nodes are representative definitions and indications, *references*, of real objects. Each node is identified with a unique NodeId and contains a set of Attributes. These attributes are used to identify, categorize, and name the node, and, despite the exact attributes of a node depends on its NodeClass, some are common to all nodes such as the DataType.

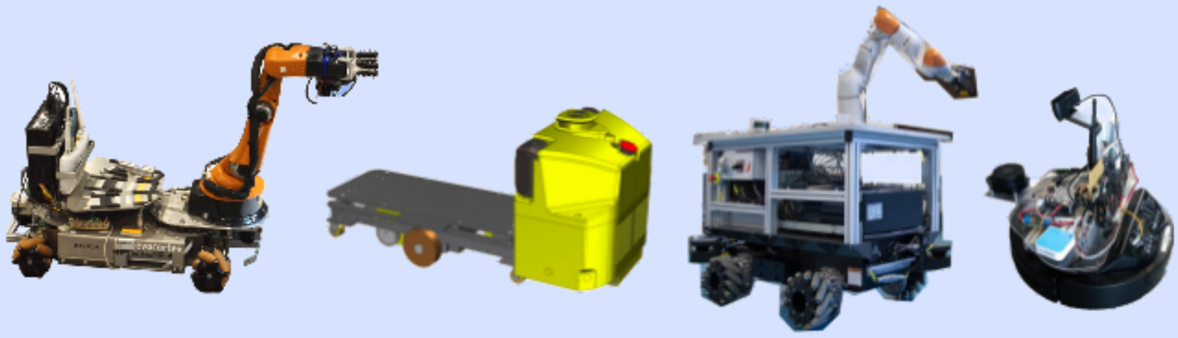
- **Subscription:** This is an endpoint within the server that publishes notifications to each client. Clients can subscribe to three different types of data changes: events, variables and aggregated data. Each data source is represented by a MonitoredItem that checks value changes according to a predefined sample interval. In this process, the clients themselves control the publication rate at which a message is published.

- **Variables:** Variables are Nodes that contain values. Each containing a DataType that depends on the Variable. A simple variable consists of a single node containing the current value of the node. Complex variables can contain child variables.

- **Events:** Events are punctual occurrences that are received via subscriptions. Events can be, errors, or address space configuration changes generated by a node specified as an EventNotifier.

- **Read and Write:** The nodes are first identified by NodeId in the address space, from that point on, its Attributes can be accessed using the Read and Write services. Read and Write will trigger value change notifications for Nodes with Subscriptions.

4 A Generic ROS based Architecture for Mobile Manipulator Robots



In the factory of the future different kinds of mobile manipulators will be found. Some might transport heavy loads, others might perform pick-place operations that demand dexterity or even complex inspection tasks. However, in spite of their very different uses, they all must face a number of common challenges such as recognising orders from high-level logistic systems, recognising their environment for navigation, interacting with the scenario for dexterity tasks or collecting data for both health and traceability monitoring. Most mobile manipulation applications are composed by navigation actions to a target position followed by an specific task, usually related to the robot manipulation skills. Robots could therefore share a bunch of common modules, such as a generic state machine, that, correctly parameterized, triggers domain application specific abilities that are already available and implemented on the robot. This chapter presents and analyses relevant industrial mobile manipulators in order to detect similarities in their hardware and software modules and to define generic framework requirements for single robot use cases. The three basic behaviours of mobile manipulators (navigation, perception and manipulation) are considered, but also other reusable modules such as monitoring, diagnostic and decision making. Based on the observed requirements, a generic framework for MMRs is proposed, the so-called *Robotframework*. The framework is then evaluated through industrial and agriculture use cases, which prove both the feasibility and the generality of Robotframework.

4.1 Requirements for Single Mobile Manipulator Robots

In order to illustrate the challenges that MMRs must face in different domains, the following sections present and analyse heterogeneous industrial robots working on different industrial use case scenarios. Namely, a robot performing navigation and precise pick-place logistic tasks within the RoboCup@work logistics league (YouBot, as part of the AutonOHM team from the Tecnische Hochschule Nurnberg) and a robot performing heavy load transportation for a car manufacturing facility (Autobod).

4.1.1 Logistics and precise pick-place for the RoboCup@work and ERL industrial leagues

RoboCup@Work [99, 231] and the European Robotics League-Industry [100] are competitions that target the use of robots in work-related scenarios, tackling open research challenges on mostly unsolved problems in robotics, artificial intelligence, and advanced computer science. In particular, they are mainly interested in perception, path planning and motion planning, mobile manipulation, planning and scheduling, learning and adaptability, and probabilistic modelling, to name just a few. The orders are generated by a central manufacturing system, represented by the Referee Box, and, in order to complete these tasks, the robot needs to master skills such as localization, navigation, object detection or object manipulation.

The KUKA omni directional mobile platform Youbot in Figure 4.1 has been used in both competitions. Regarding its hardware, at the end-effector of the manipulator, a primary Intel RealSense 3D D435 camera has been mounted for detecting objects. Additionally, a secondary one has been mounted on the back of the robot for helping with the detection of forbidden areas on the way. Two laser scanners, one at the front and one at the back of the YouBot platform, are used for localization, navigation and obstacle avoidance. The YouBot's default computer has been enhanced with a NUC7i7BNH computer for intensive tasks like 3D point cloud processing. Table 4.1 shows the main hardware specifications of the robot. The drivers for the robot mobile platform and manipulator as well as for the sensors like laser scanner or cameras are provided by the different hardware vendors, and are accessible within the ROS framework ecosystem.

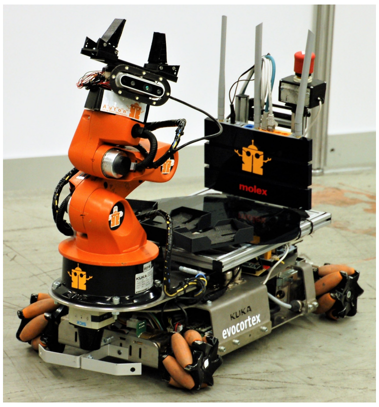


Figure 4.1. Customization of KUKA YouBot platform for team AutonOHM 2018.

Table 4.1. Hardware Specifications.

PC 1	
CPU	NUC7i7BNH
RAM	16 GB DDR4
OS	Ubuntu 16.04
Gripper	
Type	3D printed
Motor	Dynamixel AX-12A
Sensors	
Lidar Front	SICK TiM571
Lidar Back	SICK TiM571
3D-cam arm	Intel RealSense D435
2D-cam arm	Endoscope Cam
3D-cam back	Intel RealSense D435

The software modules are composed of a combination of pure-open-source, customized-open-source and self-developed packages. In fact, image processing is handled with OpenCV library (2D image processing and object recognition) and PCL (3D image processing). For mapping the arena and navigating on it customized gmapping [232] and navigation-stack [233] ROS-packages have been used. Self-developed packages include the state machine and different precise navigation, perception and manipulation modules. To perform the transportation logistics, a task planner node processes the high-level orders received from the Referee Box, calculates the best route considering the maximum transport capacity and distances between the workstations, and generates a robot-readable set of subtasks that are managed on the state machine.

The main control of the robot is coordinated following the state machine in Figure 4.2. It starts with an initialization state (*Init* state) where the robot receives the map and localizes itself on it. From there, it moves to the *Idle* state and waits for new tasks to perform. The Referee Box provides the high-level orders which are processed by the *Task Planner* node and passed to the state *Running*, divided into a vector of smaller subtasks. This state is divided into substates where the *Movement*, *Grasp*, *Delivery*, *Precise* delivery and *RotatingTable* subtasks are managed, as it is depicted in Figure 4.3. Once every subtask is finished it returns to the Idle state to wait again for new tasks to perform. It is important to notify failures, such as an unreachable navigation goal or an unsuccessful grasping of an object, and react to them. Therefore, most of the states have error handling behaviours (*Error* state) that manage recovery actions that imply repeating the action or triggering planning modifications. Next paragraphs describe how the different subtasks are accomplished, grouped by type of robotic skill.

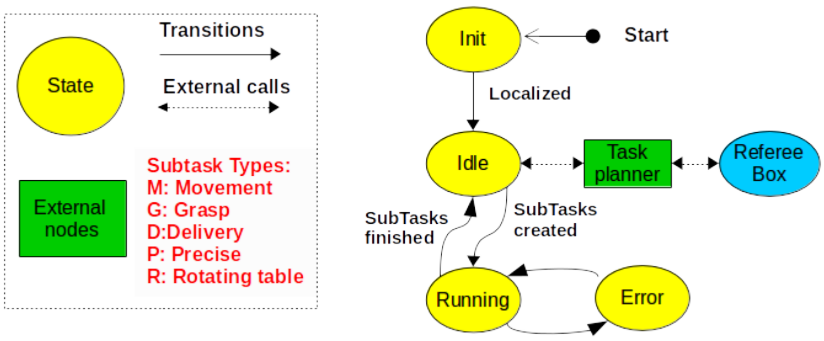


Figure 4.2. Global overview of the state machine for the RoboCup@Work robot control.

Mapping, Localization and Navigation

A very important task when facing a new competition is generating the map (Figure 4.4) of the working area (Figure 4.5). Mapping is a necessary task that needs to be done at least once, and then be updated every time the working environment suffers significant structural changes. The

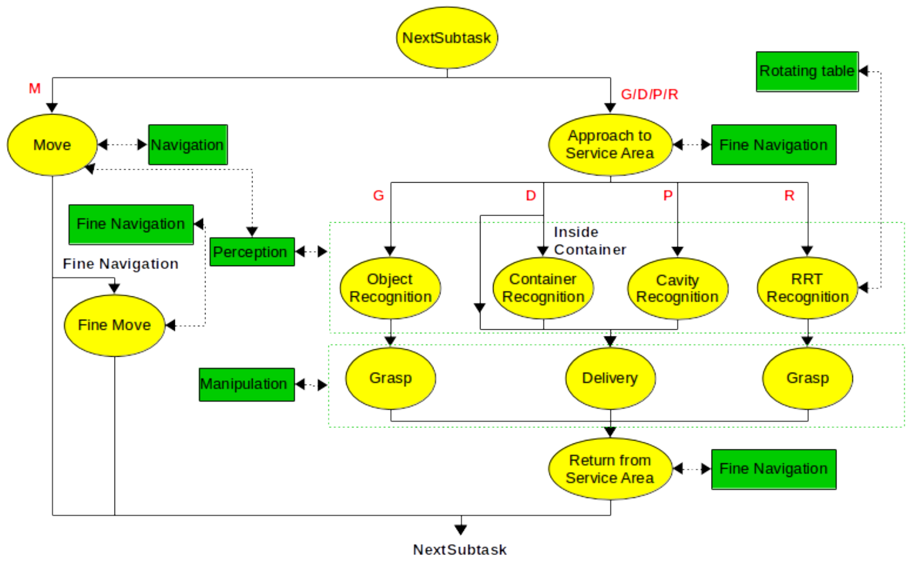


Figure 4.3. Detail of the substates of the Running state.

map is then loaded by the robot and used to localize itself, to generate global path planning plans to navigate, and to detect unexpected objects on the working area to avoid collisions.

For the global navigation (*Move* substate), the ROS navigation stack has been used. The localization is based on a Monte Carlo particle filter algorithm, close to the AMCL (Adaptive Monte Carlo Localization) localization described in [234]. The algorithm is capable of using two laser scanners and an omnidirectional movement model which provides useful positioning with an approximate error of about 6 cm, depending on the complexity and speed of the actual movement.

For the fine navigation (*Fine* substate), such as approximation to service areas, an approach based on the front laser scanner data and the RANSAC algorithm [235] is used to detect the workstation in the laser

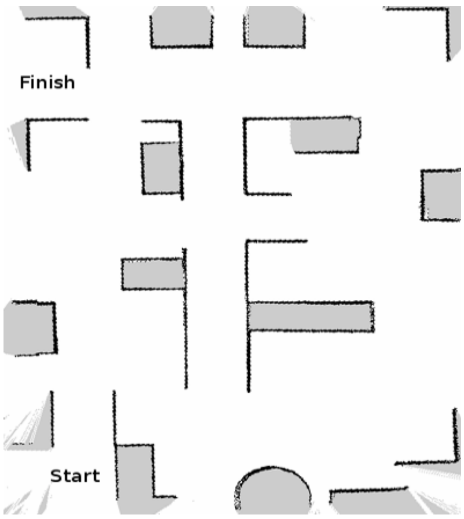


Figure 4.4. Map used for navigation.

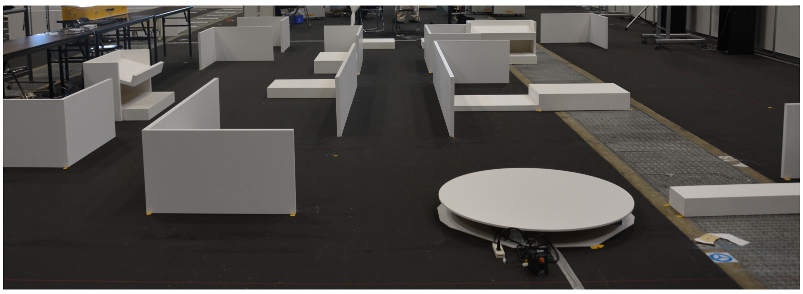


Figure 4.5. The @Work arena during the RoboCup world cup in Montreal.

scan. Out of this, the distance and angle relative to the area are computed. Using this information, the robot moves in a constant distance along the workstation. A mean positioning error of under 3 cm was achieved during a navigation benchmark tests performed in the European Robotics League local tournament in Milan.

Additionally, yellow/black barrier tapes are semi-randomly placed on the floor and used to mark restricted areas during the RoboCup@Work competition. The robot needs to detect them and prevent stepping into the restricted areas to avoid being penalized with point deduction. The process to detect and avoid crossing the barrier tapes is: The Front and Back Realsense cameras presented in Table 4.1 are used to obtain RGB images of possible barrier tapes placed on the floor (Figure 4.6-a). These images are transformed to obtain a top-down or bird-view perspective (Figure 4.6-b). The images are now color filtered and compared with the barrier tape pattern (Figure 4.6-c). If the barrier tape is detected, the point cloud information of the image is used to localize its position on the map, and include it as an obstacle. The local planner will then take it into account to plan new navigation trajectories that avoid crossing them.

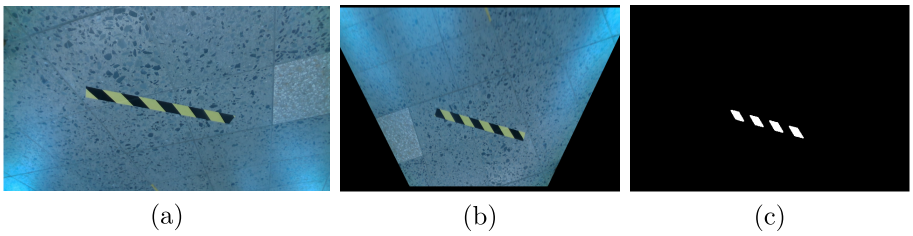
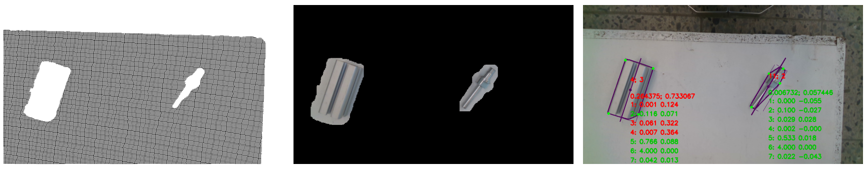


Figure 4.6. Barrier Tape Detection: (a) Camera image of the barrier tape; (b) Birdview; (c) Filter for yellow RGB and HSV values and HU-Moments.

Perception and Manipulation

In case of a Grasp subtask, it is required a stable object recognition (*ObjectRecognition* substate) that allows to grasp objects reliably (*Grasp* substate). For this purpose, the robot performs a *Fine Navigation* to a pre-grasp position. Once the mobile base reaches the position, the arm is extended and positioned above the service area, performing the object classification task using the RealSense camera as seen in Figure 4.7. The object recognition is performed in three steps: First, the 3D camera point cloud is used to segregate the flat points representing the workstation platform from the objects to be detected; Second, the segmented 3D points are projected to the cameras RGB image and several vision learning morphological operations are applied to them; Finally, the features are evaluated and used to classify the available objects on the RGB image, and to analyse the most appropriate grasping points for the gripper.



(a) (b) (c)

Figure 4.7. Segmentation mask: (a) The projected point cloud to cameras RGB image; (b) Filled border and morphological operations; (c) Classified objects.

Once the desired position is located, the arm is activated, whether for picking up and storing the object on the robot, or for delivering it (*Grasp* substate). The Manipulation module is responsible for the arm and gripper control, as well as for the inventory management. The picks are always performed from a flat surface, while the surface of the placements can remarkably change. Placing an object on a flat surface (*Delivery* substate) is a direct action. Placing it on a shelf, inside a container or through a cavity requires an additional set of perception and manipulation abilities. The robot must therefore recognize the exact position of containers (*Container Recognition* substate) or cavities (*Cavity Recognition* substate), respectively. For that purpose a container recognition module has been developed, that detects blue or red boxes (see Figure 4.8). Similarly, a cavity recognition module has been developed in order to classify the different holes and precisely deliver an object on it (see Figure 4.9). Finally, in case the object needs to be placed on a platform with a shelf, a fixed position and predefined movements are used to avoid colliding with the shelf, as seen in Figure 4.10.

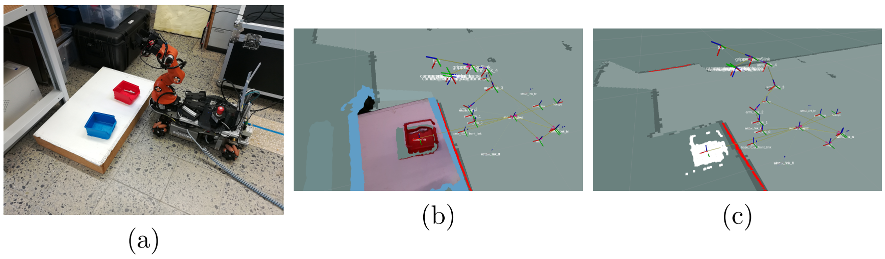


Figure 4.8. Box Detection: (a) RGB image with Blue and red boxes; (b) Point cloud image (c) Red filtered point cloud and mass center.

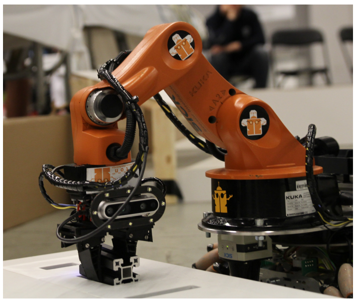


Figure 4.9. Precise placement of objects.

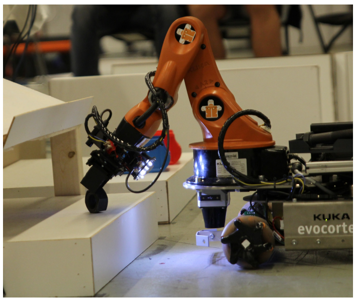


Figure 4.10. Placing an object below the shelf.

Rotating Table

In case of a RotatingTable subtask, the robot first navigates to the rotating turntable and performs a pre-processing step to determine the rotation velocity of the table and the objects' position on it (*RRT Recognition* substate), as seen in Figure 4.11. With the collected data points of each circular path, a RANSAC-based algorithm [235] calculates the rotation speed of the table, its center (blue marked in Figure 4.11-b), and the radius of each determined path. Having all necessary information and making use of the previously recorded time stamps, it is possible to estimate an approximate moment, when each object passes the object grasping position. To achieve an accurate grasping (*Grasp* substate), an additional endoscope RGB camera has been attached on top of the manipulator. A background change algorithm is now applied to the image in order to detect the object entrance in the camera view. The previously calculated circular path velocity is used to close the gripper at the right moment.

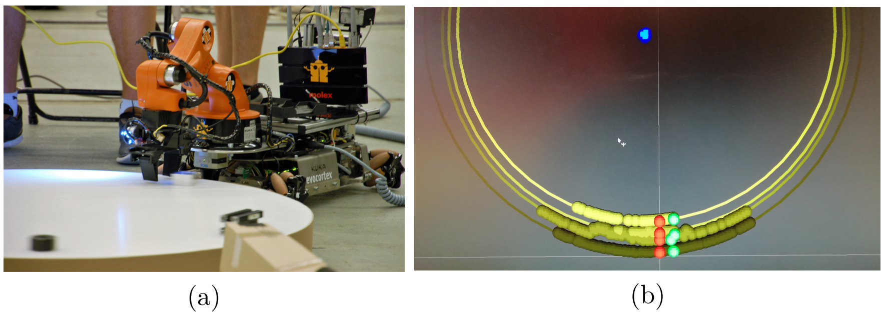


Figure 4.11. Rotating Turn Table: (a) Robot in front of the rotating turntable grasping an object; (b) All data points, given by the object recognition, and the result of the determined circular paths of all objects on the turntable with different grasp points (red marked).

More detailed information about the robot and the tasks performed by it are available in the winners papers of the RoboCup@Work World championships 2017 and 2018 [97, 236].

4.1.2 Precise heavy load transportation for a car manufacturing facility

Currently, load transportation tasks are mostly done either by complex conveyor belts (highly optimized but expensive systems where even minor changes require big investment), by guided milk-runs or by the workers themselves. In the future, mobile robots will be able to navigate the facility and perform on-demand deliveries of material boxes from the goods receipt to each station in the production line. Then, they will bring material from the pre-assembly to assembly lines, and, finally, carry the finished products to the delivery area. Using mobile robots for heavy load transportation has several advantages. Firstly, it increases the flexibility of the system by making each entity independent of the rest and, therefore, making it easier to replace entities in case of failure. Secondly, extra robots can be added at peak work times and be removed when they are no longer required. Thirdly, on-demand deliveries lead to a reduction of the warehouses at the factory (also called supermarkets).

This section presents a particular case of MMR, an autonomous transport vehicle (ATV) prototype for heavy loads transportation in the Bosch Production System (BPS), whose test area is presented in Figure 4.12. As well as in the previous section, the robot needs to create a map of the working area and navigate on it to move from one point to another. The modules used for this purpose are, as explained in the previous section based on standard ROS packages, and are therefore ignored here. This section is focused on the docking manoeuvre problem: the action of approaching the warehouse areas (the so-called supermarkets) and precisely entering the zones to load heavy packages on it.

The robot prototype has been developed by Bosch GmbH in cooperation with Technische Hochschule Nuernberg Georg Simon Ohm. The Bosch

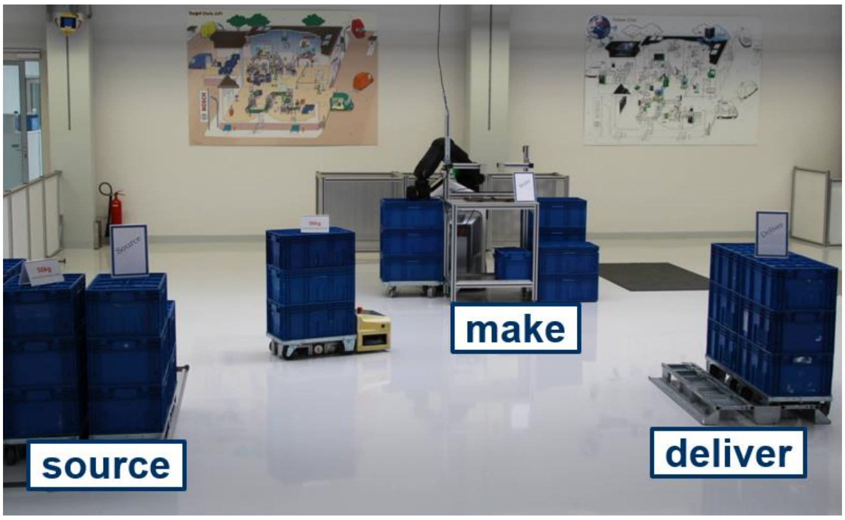


Figure 4.12. Test area at Bosch Nuremberg: The robot loads material from the input supermarket (*source*) and transports them to the work station (*make*). Then, the robot takes the finished product to the output supermarket (*deliver*).

ATV in Figure 4.13 seeks the development of a flexible transportation system that is able to work in the existing factories. In order to autonomously carry materials from the source supermarket to machines and finally to delivery stations, the robot must be able to carry out tasks such as navigation, obstacle avoidance or precise docking manoeuvres without the help of environmental markers.

The software architecture is implemented in a Pokini i2 industrial mini-computer, running on Ubuntu and using ROS as a middleware for the complete system. ROS packages have been used and improved for camera drivers, path planning, navigation, etc. To generate a map, localize the robot on it and navigate avoiding obstacles on the way, a safety laser scanner has been used. More technical specifications of the robot are shown in table 4.2.

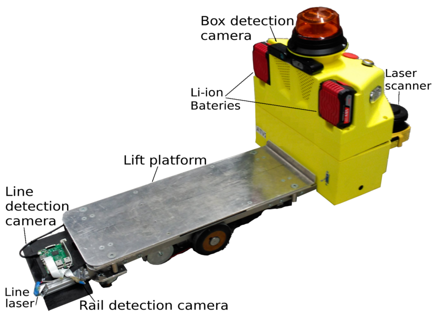


Figure 4.13. AutoBod prototype.

Table 4.2. Platform Specifications.

Maximum Speed	1 m/s
Accuracy Global Nav.	+/- 50 mm
Accuracy Docking	+/- 10 mm
Payload Capacity	200 kg

Similarly as in the previous Robocup@work use case scenario, the control of the ATV follows a singleton pattern state machine (see Figure 4.14). In the initialization state (*Init* state), the robot receives the map and localizes itself on it, waiting in the *Idle* state for new orders to perform. These high-level orders can be supplied either by a user manually or by a production system planner. The tasks are processed by the *Task planner* node and sent to the state machine, divided into a vector of smaller subtasks. The *Move (M)*, *LoadBox (L)*, and *DeliverBox (D)* subtasks are now managed in the *Next* state.

This use case focuses on three different docking manoeuvre algorithms to automatically perform the process of loading and delivering boxes from material supermarkets. Other important functionalities such as navigation, monitoring or error management are similarly used as in the previous RoboCup use case, but not covered in detail here.

The ATV first drives close to the supermarket using the state of-art ROS localization [237] and navigation modules [233] based on encoders, IMU and laser scanner sensors (*Move* state). The global localization has an

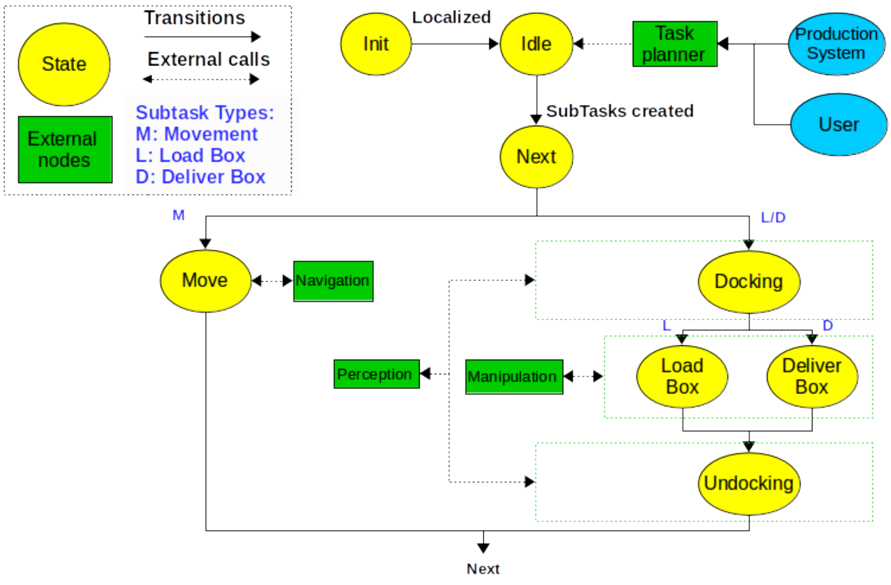


Figure 4.14. State machine for the Autobod robot control.

accuracy of about 5 cm, which is not sufficient to drive under the roller at which the load is located and performs a precise docking manoeuvre. By the use of low cost sensors, the robot follows characteristic features in the docking area such as material boxes, rails or additional lines in order to accurately move into the supermarket, and pick-up or delivery the material (*Load Box* and *Delivery Box* states). Once the load is lifted, the ATV leaves the supermarket and it is again localized using the laser scanner. Below, three different docking manoeuvre approaches making use of perception, navigation and manipulation abilities are described.

Line following

The line detection approach is a low cost, robust and repeatable method for the docking manoeuvre. First, a pre-step is necessary to place tape-lines with a color that contrast with the color of the floor. The image obtained from an ultra wide angle RGB camera, directed towards the floor (see figure 4.15), is processed to detect the line and its orientation. This information is used to guide the robot and to drive into the supermarkets. Then, the robot lifts the load and drives out by following the line again. In order to avoid external disturbances and obtain a constantly well distributed illumination, the camera has been integrated into a housing with low power LEDs. Despite being a robust method, it is desirable to avoid the markers in the future, thus, other possibilities have been researched.

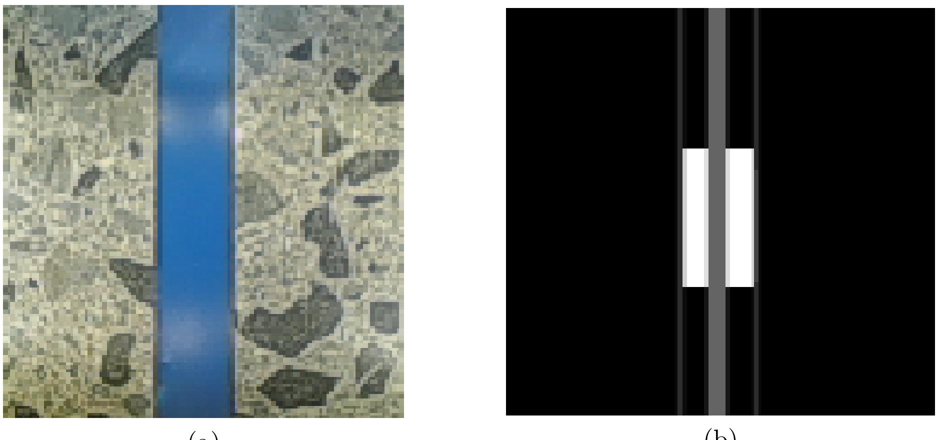

(a) (b)

Figure 4.15. Line detection: The camera's RGB image (a), processed image including detected line (b).

Box detection

To develop a marker-free fine localization approach, it has been first decided to use one of the most common blue boxes for material carrying at the Bosch Production Systems as seen in Figure 4.12. To detect and follow the box, a 3D camera is mounted on the yellow housing of the robot. The position of the camera allows the detection of the box until the roller is above the robot.

Regarding image processing, the floor (figure 4.16, mark 2*) and the background (figure 4.16, mark 1*) are first detected and removed from the RGB image. The floor gets extracted by comparing the height values from the depth sensor's point cloud to a precalculated lookup table. The background is removed by analyzing the depth values and erasing pixels exceeding a certain distance. This first filtering reduces the area in the RGB image to be processed by the box detection cascade algorithm and speeds up the process.

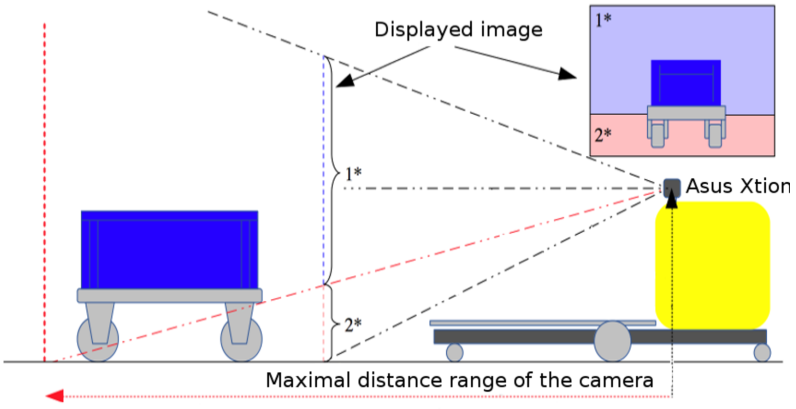


Figure 4.16. Drawing of the box detection setup with the Asus Xtion camera on the robot. Meanings: 1* represents the background; 2* represents the floor.

To detect the boxes, the OpenCV library's Haar Feature-based cascade classifier has been used. The training for the cascade classifier consists of 900 positive and 1,651 negative examples and 23 training levels.

Since many similar boxes can be found in the supermarket (see figure 4.17), an additional filter is applied to find the correct box. The 2D

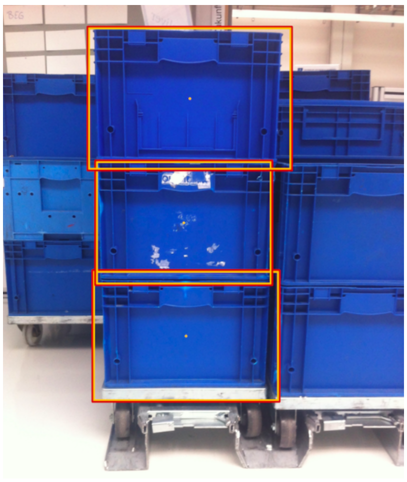


Figure 4.17. Box detection: Asus Xtion camera's RGB image with multiple boxes detected.

position information of the classifier is used in order to recognize the lowest and most centered box in the image. Once the right box has been recognized, the depth information is used to get the rotation of the box relative to the position of the camera.

Rail detection

Another approach without the need of additional markers consist on using the available rails on the supermarkets floor which are mounted permanently. Once the position of the robot relative to the rail is known, it is possible to control the ATV along the rails, until it reaches the roller for docking.

For the detection of the rail, a low-cost, laser-based triangulation method is used. A red line laser (power output 5 mW, opening angle 90 degrees) is mounted on the left side of the ATV, about 10 cm above the rails, facing vertically downwards. The camera, a low-cost Raspberry-Pi NoIR camera, is attached with a horizontal displacement of 10 cm to the line laser, facing towards the projected line (see figure 4.18).

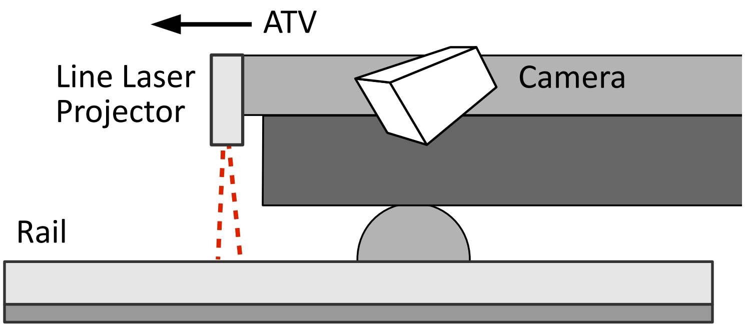


Figure 4.18. Diagrammatic drawing of the mounting of the line laser projector and the camera on the ATV (side view).

The rollers are mainly guided by one rail (either left or right). In the prototype setup, a camera and a line laser are mounted sinistrally allowing only docking to left-side-guided rollers. The limited opening angle of the camera and the line laser prevent a centered attachment to the ATV.

Due to the displacement between the laser line projector and the camera, different heights of the rail result in shifted pixels in the camera image (see figure 4.19). As a result, it is possible to determine the position of the rail relative to the robot.

Figure 4.20-a shows the RGB camera image of the Raspberry Pi NoIR camera. The edge of the detected rail is shown as a white cross in the image.

Regarding image processing, the OpenCV library is used. The red laser line gets extracted from the image using color filtering in HSV (hue, saturation and value) representation. Noise is removed with the help of

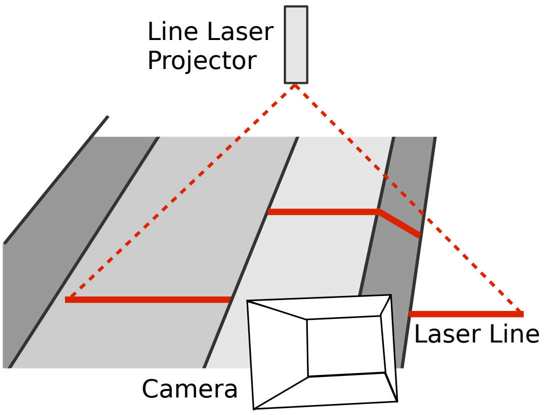


Figure 4.19. Diagrammatic drawing of the line laser projector, the rail and the camera (3D view).

OpenCV's functions erode and dilate. The resulting image (see figure 4.20-b) contains the extracted laser line.

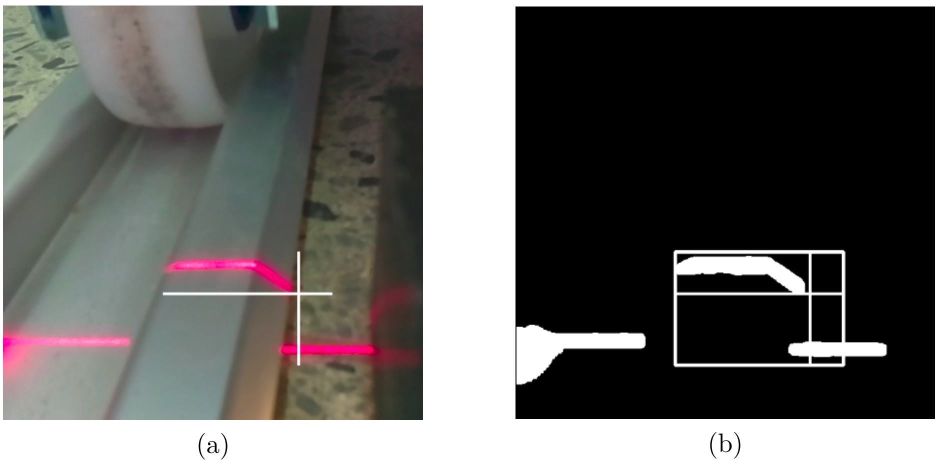


Figure 4.20. Rail detection: (a) Raspberry Pi NoIR camera's RGB image; (b) processed image including detected rail.

Template matching is applied to determine the exact position of the rail in the image. The template used is visible in figure 4.20-b as the surrounding rectangle. As a result, pixel coordinates of the rail's edge can be calculated. Based on this information, it is possible to control the robot along the rail.

Due to the distinct geometric shape of the rail resulting in a unique image, errors like interfering objects are detected and the robot is stopped.

The proposed approach offers the advantage of being more robust against sunlight than an infrared-based 3D depth camera. By using a line laser with a higher power output, the robustness could be further increased. Furthermore, the solution is remarkably cheaper than the application of a standard 2D laser scanner (LIDAR).

More detailed information about the robot, its modules and the accuracy evaluation performed are available in [111].

4.1.3 Requirements identification

It is possible to glimpse from the previous sections two strong tendencies in nowadays manufacturing. On the one hand there is an increasing need of developing new robotic applications or using existing ones to support the development of new technologies. On the other hand, these solutions consist of a set of co-related mobile, manipulation and perception robotic modules oriented to achieve concrete functionalities. To build the integration between the different modules, current RFs such as ROS, provide hardware abstraction and software resources to easy the set up of their basic functionalities. Available RFs provide drivers, communication mechanisms, simulators and higher level modules for localization, navigation, manipulation or perception. However, every time a new project starts the state machine that builds the logic of the robotic applications, the relationships between the different robotic modules or the mechanisms to externally interact with them are mostly built from scratch.

There is therefore a necessity to build a new robotic architecture that embraces high level robot functionalities on top of the already available RFS. The main requirements of such a robotic system are collected in Table 4.3. The architecture must provide reusable modules that most applications require. In concrete, it must include navigation abilities to perform global and and precise navigations (*SR1*). The architecture must also provide manipulation abilities to perform specific dexterity tasks such as precise drilling or inspection operations, or to support operators on the working space with complex or monotonous tasks (*SR2*). Additionally, it must consider perception abilities to detect obstacles on the way, markers that help localizing itself more precisely, and objects that need to be inspected or moved from one place to another (*SR3*).

The plans generated by high-level decision modules are normally based on domain expert knowledge and the application context. These plans must contain enough information for the robot to understand where to go and which actions to take at each place. It is important to establish a number of rules and generic key-words to convert complex high-level order information into robot-comprehensive multidisciplinary plans, increasing the extensibility of applications domains and the reuse of previously developed robot abilities, without the need to touch or recompile the core of the framework (*SR4*).

Previous experiences have also shown the importance of detecting and notifying critical errors and warnings on time as well as building strong recovery behaviours or, ultimately, triggering the complete application interruption. At the same time, latest digitization technologies should be used to increase the amount of information available to optimize the processes, trace the results and prevent future errors (*SR5-SR6*). During the development of previous works, it has also become clear that mobile manipulator related applications are complex and carry a steep learning curve for new users. To simplify the use of the framework for new and former users, the architecture must provide an easy-to-use graphical user interface (GUI) that allows a partial control of the robot application such as starting, pausing or stopping it on demand, monitor its progress and present the general system status (*SR7*).

Finally, when designing and developing a RF it is desirable its adaptability to permit the adjustment of previous abilities to new scenarios and scalability to permit the integration of novel abilities without needing to change the core architecture. It is important to remark that these requirements are based on single robot use cases, ignoring the necessity of building a multi-robot system to coordinate several robots, creating intelligent collaboration channels or permitting the robot interact with other non-robotic industrial systems. These aspects are later considered in Chapter 5.

Table 4.3: Main System Requirements for a single MMRs.

System Req.	**Description**
SR1	The robot can **localize** itself within the working space with or without the help of markers and can **autonomously navigate**.
SR2	The system must provide **precise manipulation capabilities** to support dexterity tasks.
SR3	The system must provide **perception capabilities** to allow a save and precise navigation or to enable precise dexterity manipulation and inspection tasks.
SR4	The robot can **execute parameterized high-level orders** increasing its adaptability to context changes, scalability and reuse on heterogeneous applications.
SR5	The system must **detect and notify critical errors and warnings**, triggering recovery behaviours or, ultimately, the complete application interruption.
SR6	The system must enhance the factory digitization, **logging historical events** such as the route (positions) of the robot, the actions occurred there and unexpected incidents.
SR7	The system must provide an **easy-to-use GUI** to start the application, monitor its progress and present the general system status.

4.2 Robotframework: a Generic ROS-based architecture for mobile manipulator robots

This Section describes the general control architecture proposed in this work, the so-called Robotframework (see Figure 4.21), which seeks the easy integration of the different robot functionalities ensuring the system requirements presented in Table 4.3. It follows a distributed computing design allowing several tasks to run in different computers while still appearing to its users as a single coherent system and allowing an easy extensibility.

ROS is proposed as core communication middleware among the different modules, as, during the last decade, ROS has become the de facto standard framework for the development of software in robotics. ROS is a flexible open-source framework for writing robot software that provides a collection of communication mechanisms, tools, libraries, and rules that aim to simplify the task of creating robot software for a wide variety of robotic platforms. Therefore, being a ROS-based architecture provides the generality needed to build solutions in different domains.

Robotframework consist of several modules which are grouped in four abstract layers according to their overall goal within the framework. The architecture follows a top-down approach in terms of the functionality covered by modules, from high-level decisions related to the desired plans depending on the application context, to the low-level drivers fully dependent on the robot used. Layers definition allows not only the separation of concerns between modules, but also decoupling from each other, for which the interfaces between them have been also defined.
The upper layer is responsible for generating the plans with the applica-

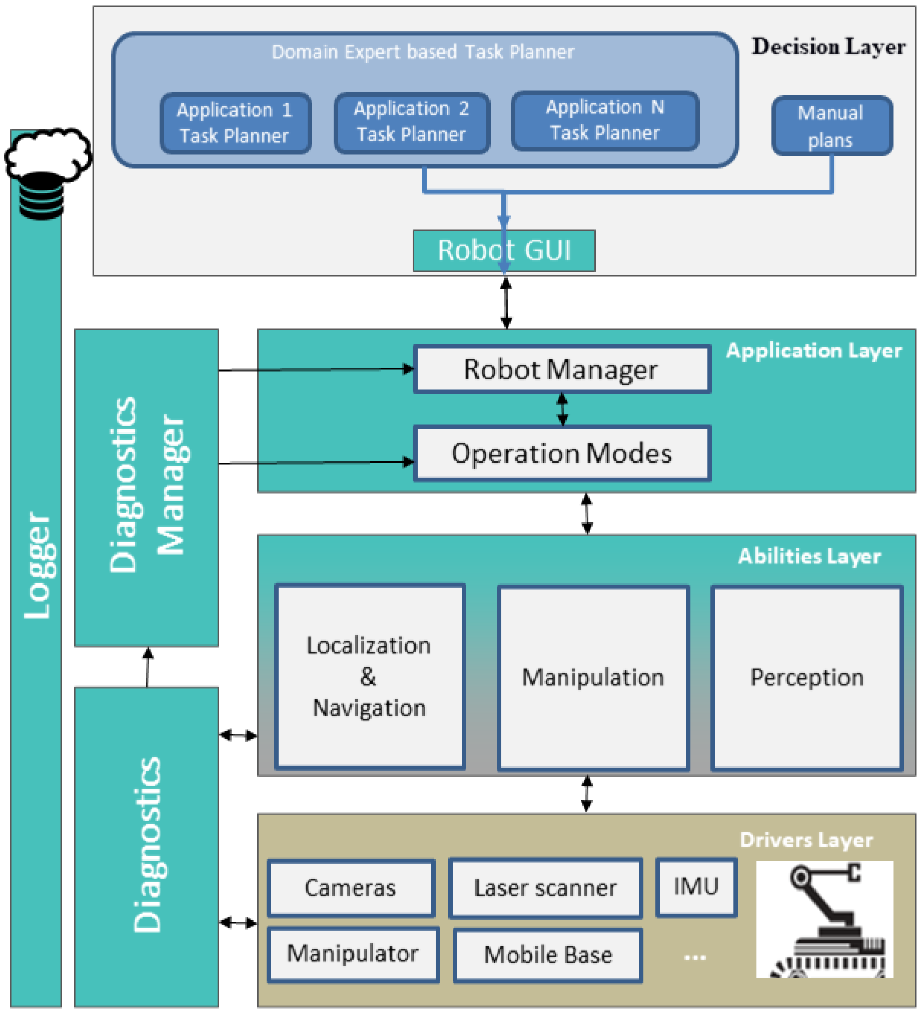


Figure 4.21. Four-layer Robotframework control architecture. Modules represented with turquoise color are common to any application. Grey modules are standard ROS modules while the rest are application specific.

tion specific targets that the mobile manipulator must address. Targets are composed of a navigation goal that the mobile manipulator must reach and the tasks to be executed if necessary. This thesis proposes a meta-model for the generic definition of these plans, containing all the necessary information so that the manipulator can navigate to the targets and execute the necessary tasks. In addition, it is proposed a guideline, driven by template examples, for the generation of new plans using the human-readable JSON notation.

The plans passed to the robot, according to said meta-model, are the mechanism that allows the interaction between layers 1 (Decision Layer) and 2 (Application Layer). More specifically, the plans are generated in the Decision Layer, where the domain experts knowledge is, and shelled in the Application Layer which is in charge of translating that information into instructions for modules of the two lower layers closest to the robot's intrinsic functionalities and hardware. For its part, the interaction between layers 2 (Application Layer) and 3 (Drivers Layer) is based on the implementation of the robot's abilities in the form of plugins, from which the robot's drivers, located in layer 4, are used.

Finally, there are modules whose functionality is transversal to these 4 layers, since they have mechanisms related to the monitoring and diagnosis of the actions carried out by the rest of the modules. The gathered information is used to diagnose the system and trigger the corresponding recovery actions and saved in data bases, increasing the factory digitization and enabling its better understanding and optimization. The information is first collected in the edge, processed and then moved to the cloud.

Therefore, the architecture consists of a generic core, common to all applications, and custom modules dependant on the application and the

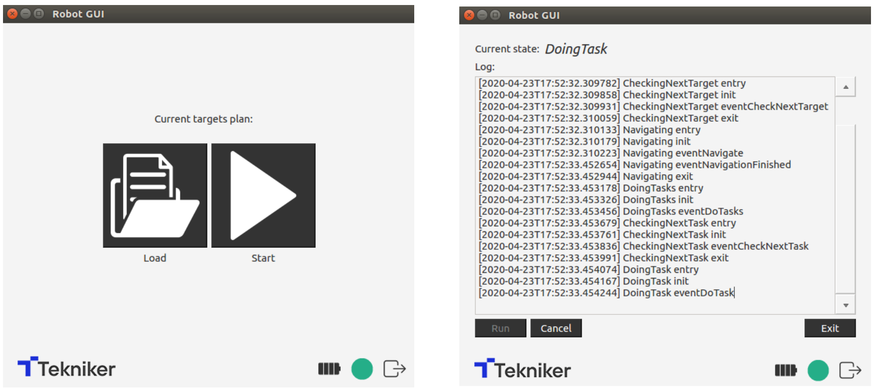
MMR being used. In concrete, common modules are represented in turquoise color and include the robot user interface, the robot_manager in charge of the overall system, the targets_plan_operation_mode in charge of managing the plans generated in the application layer, the monitoring, diagnosis and logging modules and some common parts of the Abilities Layer. The gradient turquoise-gray colour represents the robot functionalities that are build on top of ROS modules and which compose the basic but constantly expanding Robotframework abilities. These abilities are based on state of the art ROS functionalities or newly developed and tested ones during different projects. Most of them are now available within the architecture, but it is up to the user to use them or implement new modules using the available ones as templates. Modules represented by gray colour are based on pure ROS packages, such as robot drivers, which depend on the specific robotic components used. Finally, the high-level decision support modules, represented in blue, will be subject to the application and dependent on the domain experts decisions. This modules however, must follow the plan meta-model rules in order to be understood and executed in the lower layers of Robotframework.

4.2.1 Decision Layer

The Decision Layer is the upper layer of the Robotframework architecture that contains the modules involved in application related high-level decisions, which are expressed in terms of plans for the MMR (addressing the system requirement SR4). In this context, the robot provides services contracted by higher level management modules, such as an operator or a MES. These orders are processed, based: on rules defined by domain expert knowledge, on the current environment state (map or obstacles), and on information obtained from previous plan executions. For example, in the RoboCup use case the Referee Box is in charge of generating the orders, whereas in the Bosch ATV's scenario a production system planner supplies transportation orders. As observed in Figure 4.21, the Decision Layer consists of one to several **Application Domain Task Planner** modules that, based on higher level orders, generate modular and structured plans for the specific MMR that are then executed and managed in lower Robotframework layers. This layer also considers situations in which plans are generated manually, represented by the so-called **Manual plans** module. Note that task planners vary from one application to another.

The Decision Layer is also provided with a GUI (**Robot GUI** module in Figure 4.21), common for any application. It provides an easy-to-use user interface that enables partial control of the robot, starting/stopping specific operation modes, as well as the monitoring of the MMR's general status (fulfilling the system requirement SR7). An operation mode is the implementation of a specific robot application. The proposed architecture is designed to allow implementing new operation modes and running them in the system without having to touch the core of the framework. This is achieved using ROS pluginlib, a package that provides tools for writing and dynamically loading plugins using the ROS build infrastructure. Not always required in operation modes, a user interface provides interaction between the operation mode and a human operator, or displays relevant information about an operation mode on a host PC. Current design allows (optionally) creating a user interface for the operation mode.

When the user starts an operation mode, the GUI corresponding to the selected operation mode is shown by the Robot GUI. In this case, the GUI can be used to load a previously generated plan (by a task planner or manually), and to activate it, by pushing the button *Start* as seen



Current state: *DoingTask*

Log:

```
[2020-04-23T17:52:32.309782] CheckingNextTarget entry
[2020-04-23T17:52:32.309858] CheckingNextTarget init
[2020-04-23T17:52:32.309931] CheckingNextTarget eventCheckNextTarget
[2020-04-23T17:52:32.310059] CheckingNextTarget exit
[2020-04-23T17:52:32.310133] Navigating entry
[2020-04-23T17:52:32.310179] Navigating init
[2020-04-23T17:52:32.310223] Navigating eventNavigate
[2020-04-23T17:52:33.452654] Navigating eventNavigationFinished
[2020-04-23T17:52:33.452944] Navigating exit
[2020-04-23T17:52:33.453178] DoingTasks entry
[2020-04-23T17:52:33.453326] DoingTasks init
[2020-04-23T17:52:33.453456] DoingTasks eventDoTasks
[2020-04-23T17:52:33.453679] CheckingNextTask entry
[2020-04-23T17:52:33.453761] CheckingNextTask init
[2020-04-23T17:52:33.453836] CheckingNextTask eventCheckNextTask
[2020-04-23T17:52:33.453991] CheckingNextTask exit
[2020-04-23T17:52:33.454074] DoingTask entry
[2020-04-23T17:52:33.454167] DoingTask init
[2020-04-23T17:52:33.454244] DoingTask eventDoTask
```

Figure 4.22. Robot GUI interface to load and start a JSON plan (left) and the application workflow messages once the plan has been initialized (right).

in Figure 4.22-left. The plan is then interpreted and executed by the robot while the GUI displays the current state of the system including robot-status, task-information, alerts, or the batteries level as shown in Figure 4.22-right. It is also possible to stop, re run the plan or to cancel it in any moment, by pushing the cancel button for security or re-planning reasons. There is also available a button to exit the application.

The Robotframework architecture has been designed to be generic enough to abstract application dependent modules from the robot specific modules. This is achieved by proposing a generic language to create plans, common to all applications, which is defined through the meta-model depicted in Figure 4.23. The meta-model identifies the elements needed to specify plans, their attributes and the relations between them.

As it is observed in Figure 4.23, a plan (*Plan* element) is composed by targets (*Target* element) that contain a navigation goal (*Navigation* element), to move the robot to a desired position and, if necessary, a set of tasks to be performed there (*Tasks* element).

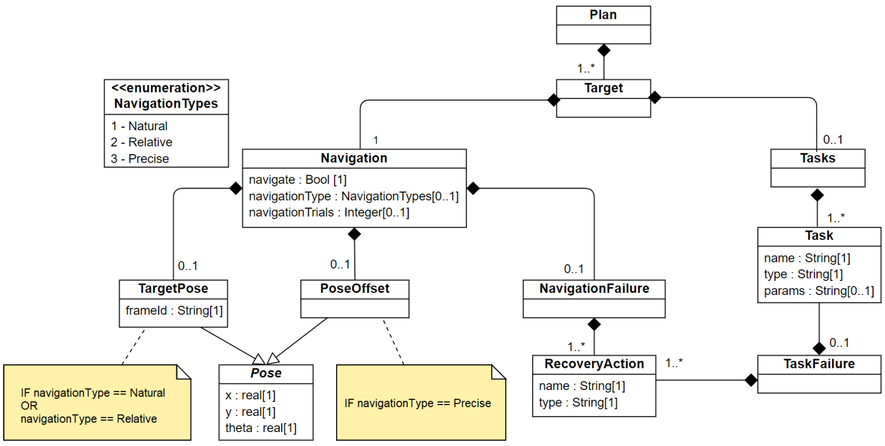


Figure 4.23. Meta-model for the plans.

The navigation step is usually required but it could be skipped by setting the *navigate* attribute to false. If there is a navigation step, it is required to set the (*navigationType* attribute) as well as the number of trials for navigation in case of failure (*navigationTrials* attribute). Three navigation types have been distinguished (*NavigationTypes* enumeration):

1. *Natural navigation*: given a position of a goal location in the workspace (*TargetPose* element; *x*, *y* and *theta* attributes), the robot can plan a path free of obstacles and generate a continuous motion to reach that goal, while avoiding any unforeseen obstacle that might appear. To achieve this, the robot has a map of the environment and the ability to determine its own position and orientation within the map's reference frame (*frameId* attribute of the *TargetPose* element).

2. *Relative navigation*: Given a position relative to the robot's current position (*TargetPose* element; *x*, *y*, *theta*, and *frameId* attributes), it generates a continuous motion to reach the goal. The displace-

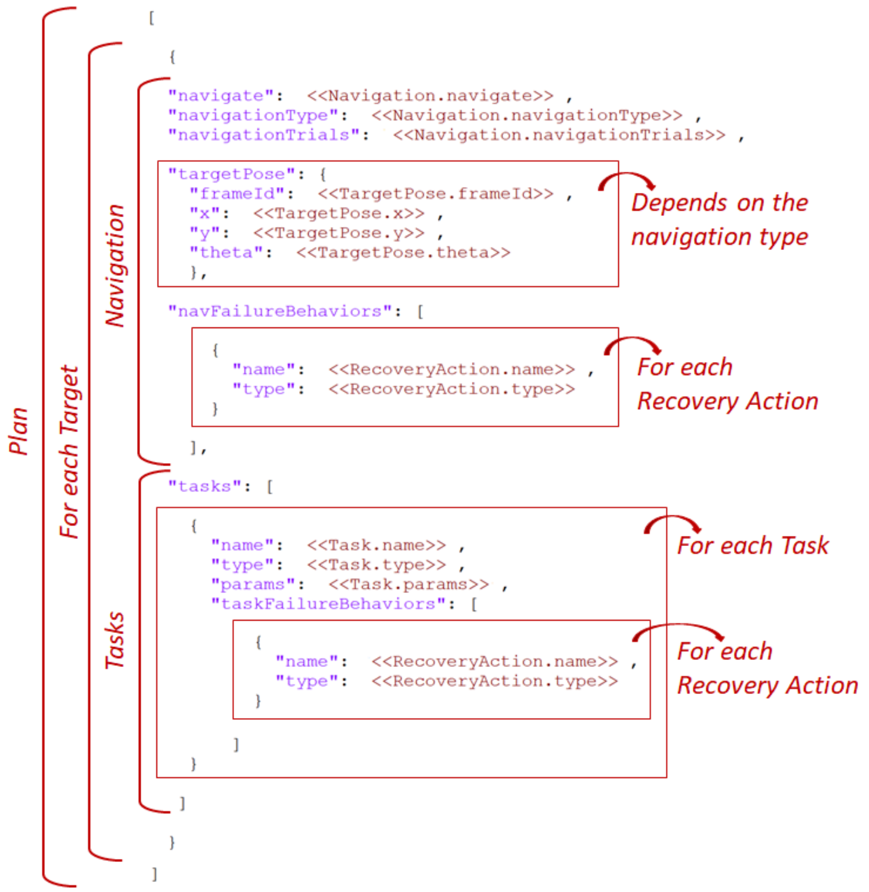
ment is measured using odometry information, i.e. wheel encoders and an inertial measurement unit.

3. *Precise navigation*: Given a position relative to an artificial mark (*PoseOffset* element), it generates a continuous motion to position the robot with respect to that artificial mark. As feedback, it uses the information provided by a camera, capable of detecting the marks using artificial vision techniques.

The proposal allows establishing which actions to take when a navigation failure occurs, which means that the robot cannot reach the navigation target, through the *NavigationFailure* element (addressing SR5). Every recovery action (*RecoveryAction* element) is determined by a name (*name* attribute) and the action type to be carried out (*type* attribute).

Once the robot reaches the navigation target, it might be necessary to execute one or more tasks (*Task* element), each of which is characterized by its name (*name* attribute), type (*type* attribute) and, if needed, the set of data required to perform it (*params* attribute). The following sub-sections will describe in detail the relations between these tasks and the MMR's abilities. Similarly to navigation failures, it is also possible to determine which recovery actions to take in case of a task failure (*TaskFailure* element).

The plans are implemented using a JavaScript Object Notation (JSON) format, which is a very common open standard and language-independent, that uses human-readable text to store and transmit data objects. For this, the template depicted in Figure 4.24 is provided, at which the attributes of the meta-model elements are stated following the format: Element.attribute (in red color). JSON distinguishes two types of elements, which can be nested interchangeably: arrays and objects. Arrays are delimited by square brackets, being array elements represented by comma-separated values. Objects are delimited by braces, and they represent lists of name-value pairs, which are separated by commas. The mane of the pair is enclosed in double quotation marks, and it is separated from its value by a colon. Note that the template of Figure 4.24 presents the case of a natural or relative navigation type. In the case of a precise navigation type, the TargetPose object should be replaced by PoseOffset and the frameId attribute should be removed.


```
[
  {
    "navigate":  <<Navigation.navigate>> ,
    "navigationType":  <<Navigation.navigationType>> ,
    "navigationTrials":  <<Navigation.navigationTrials>> ,

    "targetPose": {
      "frameId":  <<TargetPose.frameId>> ,
      "x":  <<TargetPose.x>> ,
      "y":  <<TargetPose.y>> ,
      "theta":  <<TargetPose.theta>>
    },

    "navFailureBehaviors": [

      {
        "name":  <<RecoveryAction.name>> ,
        "type":  <<RecoveryAction.type>>
      }
    ],
    "tasks": [

      {
        "name":  <<Task.name>> ,
        "type":  <<Task.type>> ,
        "params":  <<Task.params>> ,
        "taskFailureBehaviors": [

          {
            "name":  <<RecoveryAction.name>> ,
            "type":  <<RecoveryAction.type>>
          }

        ]
      }
    ]
  }
]
```

Figure 4.24. Meta-model based template for plans generation.

When implementing a plan, the JSON file represents the plan itself, which is defined by an array (enclosed by square brackets). Each target of the plan is an array element defined as an object (enclosed by braces). The information related to the navigation goal is indicated by name-value pairs, where the name is the attribute name and the value must follow the attribute-type established in the meta-model. The set of tasks to be performed at the navigation target as well as the set of recovery actions against navigation or task failures are defined by an array, whose elements correspond to each of the tasks or actions (objects enclosed by braces). Again, the information related to each task or recovery action is indicated by name-value pairs.

Implemented plans, both manually implemented and those generated by task planners, are passed to the following Application Layer. It is important to remark that task planners are the responsible for translating high-level orders into plans that follow the proposed meta-model. Therefore, it can be concluded that the plan meta-model is the mechanism used by Robotframework to abstract the architecture from the application particularities, being possible to reuse the same framework in different domains and/or applications, just developing the concrete task planners.

4.2.2 Application layer

The application layer includes the **Robot Manager** module, which is responsible for controlling the overall robotic system, maintaining its status continuously. In concrete, it is in charge of the following functions:

- Act as the *entry point for the modules in the Decision Layer*. Modules in the Decision Layer can interact with the system using the

API provided by the Robot Manager. This functionality is available via ROS topics and services.

- *Manage the initialisation and stop of the operation modes*. When Robot Manager is run, it loads a set of available operation modes specified in a configuration file. The API contains methods for initialising and stopping the loaded operation modes. It is important to note that only one operation mode can be running at the same time. Concurrent execution of operation modes is not foreseen in the proposed architecture.

- *Handle diagnostics events*. Using the output generated by the Diagnostics Manager module, the Robot Manager can act accordingly (i.e. decide whether to modify the operation execution flow or cancel the operation).

This layer also includes the **Operation Mode** module, which interprets the high-level plans and implements a specific robotic application. Therefore, it is aware of the meta-model and implementation format proposed in the previous subsection, in order to process it to extract the information needed to create instructions for the robot. This work presents the so-called *targets plan operation mode*, designed to be as general as possible, easily configurable for a variety of mobile manipulation robotic processes and, thus, avoiding an ad-hoc implementation which are only useful for specific workspace configurations. This operation mode is part of the core of the Robotframework architecture. Figure 4.25 shows the state machine implemented for this operation mode. It starts in a *Waiting* state until a new-plan event indicates the beginning of the operation. The system switches then to *Checking next Target* state, analysing the next target in the plan sequence (Target element).

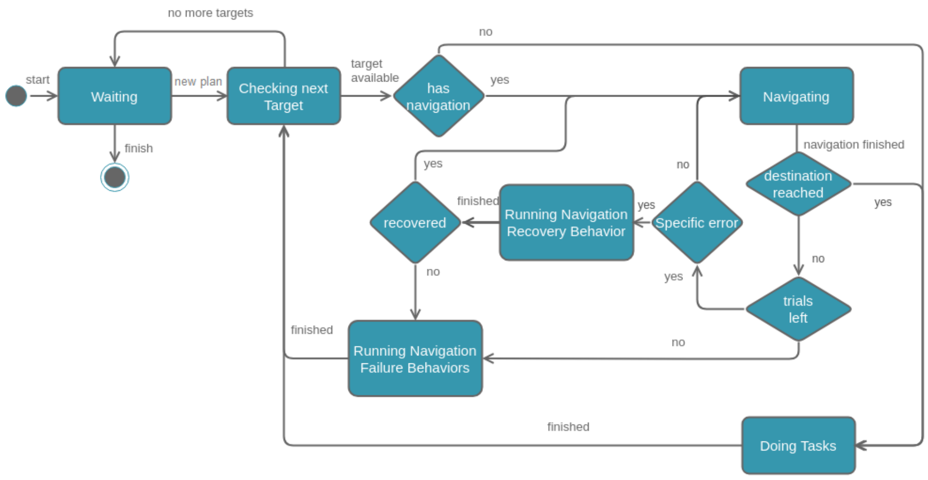


Figure 4.25. State machine representing the operation mode called *targets plan* and its navigation, tasks, and error-handling behavior states.

If the target has a navigation step, its corresponding Navigation element is processed in the *Navigating* state, which is responsible for coordinating the autonomous movements of the MMR. This state has been developed in a generic way to support different global, relative, and precise navigation modules, as it will be explained later, which cover all the navigation types identified in the meta-model. As a consequence, the architecture permits an easy integration of different navigation modules and provides the management of their results.

If the navigation is not able to reach the target pose, due to obstacles on the way or localization problems, this will be notified in the result, two possible situations are considered. On the one hand, the navigation can be repeated until the number of trials is finished (information available on the navigationTrials attribute) or the target pose is reached. On the other hand, if required, the recovery behavior stated for the Target is triggered (NavigationFailure element). Every RecoveryAction is developed as a plugin and accomplished in the *Running Navigation Recovery Behavior* state. If there are no more trials left, meaning the robot failed to reach the destination, the failure is notified in the *Running Navigation Failure Behavior* state, and the tasks to be performed at this point are skipped, addressing the following Target.

If the navigation finishes correctly, the configured set of tasks (Tasks element of the plan model) are executed in the *Doing Tasks* state, whose execution is detailed in Figure 4.26. A Task is the implementation of a robots' specific set of actions. The proposed architecture has been designed to implement these robotic tasks by using the ROS *pluginlib* mechanism. Indeed, tasks are not directly implemented in the Targets plan operation mode. Instead, the operation mode is designed to allow loading available, parameterized tasks as plugins, using ROS pluginlib mechanism. Tasks are implemented in separate shared libraries, and pluginlib can dynamically load those libraries at runtime and execute the tasks. There is therefore no need to touch or recompile the core of the framework. This is useful for reusing the application and provides a great extendibility to the system. Section 4.3 presents a task example for an aileron inspection in an industrial context and two task implementations in the context of precision agriculture for pest detection and treatment are presented to demonstrate the generalization of the architecture. These task plugins can be later used as templates and be adapted for future applications.

It is important to note that the implemented state machine also considers the possibility of failure in the task execution process. In such case, it provides a way to execute a set of Recovery Actions within the *Running Task Failure Behaviors* state. This can be used, for example, to log the failure in the results, or to notify the operator that something

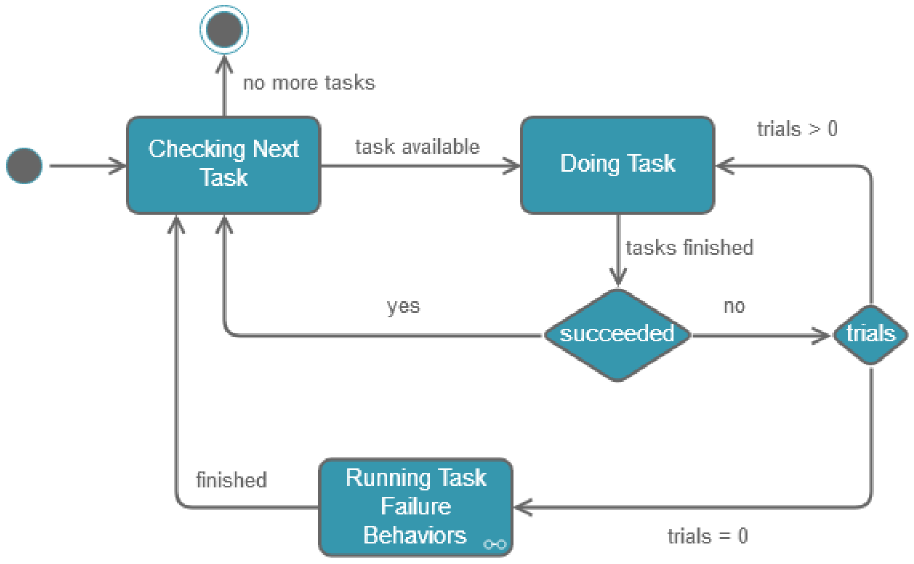


Figure 4.26. State machine detailing the execution of the *Doing Tasks* state.

went wrong with a beep or light system. Again, those recovery actions are developed as plugins.

4.2.3 Abilities layer

This layer is composed by the ROS nodes involved in the basic control functionalities of a robot. These nodes manage the sensor and actuator components, and provide robot capabilities such as autonomous navigation, manipulation, and inspection. At this level, ROS provides a wide range of state-of-the-art robotic algorithms, which are integrated as part of Robotframework: *GMaping* [232] for generating maps using data from the on board 2D laser scanners, IMUs and odometry sensors. The robot position is estimated using the AMCL probabilistic approach.

This approach uses a particle filter in which each particle corresponds to an individual map of the environment. It computes an accurate proposal distribution considering not only the movement of the robot but also the most recent observation. The maps can be manually modified to include, for instance, forbidden areas for the robot; the *Navigation Stack* used in the Navigation State for planning global and local paths, uses combined 2D laser scanners and IMU sensors to generate the velocity commands for the mobile base while avoiding the obstacles on the working environment; the *Unified Robot Description File (URDF)* for generating a combined robot description; *MoveIt!* is used for generating and executing collision free manipulation trajectories in the Doing Tasks State as it will be later presented in Section 4.3. MoveIt! related tools can be also used to integrate 3D point cloud based obstacles or useful simulation tools among other utilities.

The navigation step consists always on several main steps: mapping, map modification (manually), robot localization and navigation, which are very well resolved with standard ROS module, addressing SR1 and SR3 requirements. However, the tasks that need to be performed once the navigation goal is reached (composed by different manipulation and perception robotic skills) are very dependant on the application itself, and require from new modules development. It has been therefore decided to decouple the navigation from the following task execution and to encapsulate this second task into pluginlibs to maintain a generic state machine (fulfilling SR2 requirement). The tasks are first developed including software modules such as vision algorithms for inspection or object recognition, manipulation strategies that make use of the already mentioned MoveIt!, and so on. These tasks are then encapsulated following a plugin template provided by Robotframework which defines: how to start/stop the task; the information that needs to return during its execution; and, the information to return once finished. This information should be valid for any task and permits the generalization of the state machine for a wide range of applications. Similarly, the plugins are parameterized in order to make them reusable in a wider contexts. The task is then dynamically loaded and executed at run-time by the state machine.

Therefore, the entity of the Decision Layer in charge of generating plans must be aware of the abilities of the robot and the plugins implemented on it. More precisely, the type of every Task to execute when the navigation target is reached must correspond to the name of the pluginlib that implements it. Furthermore, the values of the parameters for the plugins are also determined trough the params attribute of the Task element in the plans.

4.2.4 Drivers layer

The drivers layer depends on the robotic platform used within a specific application. It includes the modules that allow interacting with the robot sensors and actuators. A benefit of using ROS is the availability of a wide variety of robotic components drivers (mobile robots, manipulators, cameras, lasers) and the possibility to replace them with similar ones, while maintaining the core modules. Another benefit is that ROS drivers are supposed to publish their own diagnostics messages, having already available the input data for the architectures' diagnostics module.

For example, the drivers for all the robotic components presented in this work are available in ROS. From the already presented YouBot platform, used during the RoboCup competition, or the AutoBod ATV prototype, developed with Bosch GmbH and used during the docking manoeuvre tests. Also the platforms that will be later presented: The segway mobile platform and the iiwa manipulator used to perform both industrial aileron inspections and pest inspection tasks; the kobuki platforms used to develop multi-robot task allocation strategies. Also the different drivers for 2D or 3D cameras, 2D laser scanner, IMUS, etc. are available in ROS.

This layer also contains the robot description as a URDF file. Any MMR, for example, is composed by a mobile base containing differential of omnidirectional wheels. Coupled to it, laser scanners or cameras are used to perceive the environment, map it and find possible obstacles on the way. The manipulator is mounted on top of the mobile platform and it is usually equipped with cameras to detect the objects to be manipulated, and with actuators that endow grasping capabilities. The relationships among these components as well as many other valuable information for their different software modules are defined within the robots' URDF description.

4.2.5 Monitoring Mechanisms

The three modules shown on the left side of Figure 4.21 are available with the architecture to monitor the functional state of the system. The **Diagnostics** module has been designed for collecting and preprocessing specific data from drivers and abilities layers, which are then passed to the **Diagnostics Manager** module for automatic decision making and incidents notification, fulfilling system requirement SR5. These two modules must be adapted to the application on demand.

The **Logging** module permits saving different log messages to all the layers in the architecture. The logs are used to record historical track of the process, ensuring SR6, and can either be stored locally in the robot or, in case of having an internet connection, in the cloud. The data storage and management system is built upon open source platforms and it is prepared to store and manage data coming from several robots as seen in Figure 4.27. In concrete, a Docker container has been created which already includes all these open source platforms installed and configured ready to use, making it easy and consistent to deploy and portable on any operating system.

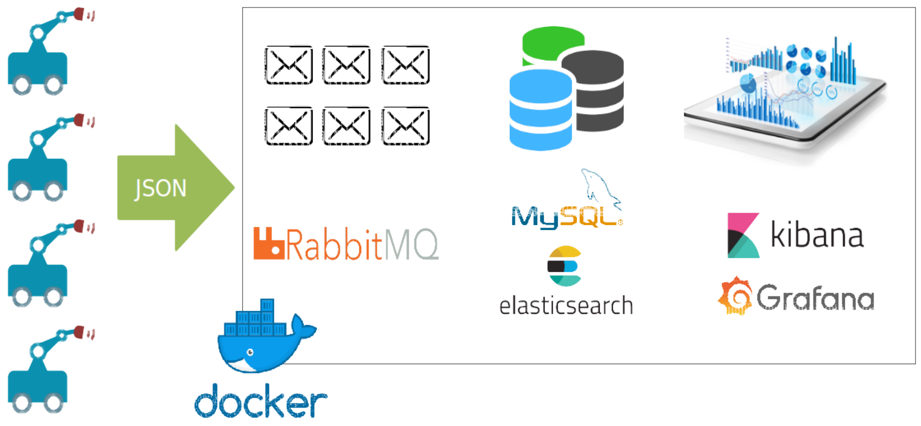


Figure 4.27. Monitoring tools for data storage and visualization.

The logger module of each robot generates a JSON file for each message being logged. This JSON file contains a robot unique identifier, timestamp of the message, and the message data, which can be classified in four levels: DEBUG, INFO, WARNING and ERROR. The generated JSON files are sent to a queue, in which messages are inserted and removed according to first-in first-out (FIFO) rule. The selected queue technology follows the publish/subscribe messaging pattern. In a publish/subscribe messaging pattern, any message published to a topic is immediately received by all the subscribers to the topic. Publish/subscribe messaging pattern provides greater network scalability and a more dynamic network topology.

For message storage, databases of both relational and non-relational types can be subscribed to the queue, depending on the interests and needs. On the one hand, a relational database is a type of database that stores and provides access to data points that are related to one another, making it easy to establish the relationships among data points. Here, MySQL is being used: an open source relational database management system compatible with all major operating systems, that is written in C and C++. MySQL is a secure relational database management system that consists of a solid data security layer that protects sensitive data. On the other hand, non-relational databases allow flexible schemes and they do not require relations between objects. Here, Elasticsearch is being used: an open source distributed search engine, usually used for log analytics and full-text search. Elasticsearch is a fast search engine that quickly finds any text search in huge data-sets. Any information that does not require strict data integrity can be stored in a non-relational database, being its main advantage its fast performance retrieving information, especially when dealing with high amount of data.

Finally, all information can be visualized using a web dashboard creator tool. This type of user interface tools helps users to easily create and edit dashboards. Here, the open source dashboard tools Grafana or Kibana can be used, which work with Graphite, ElasticSearch, InfluxDB, MySQL, and more data sources. It helps users to easily create and edit dashboards. An example of a web dashboard using Grafana is shown in Figure 4.28

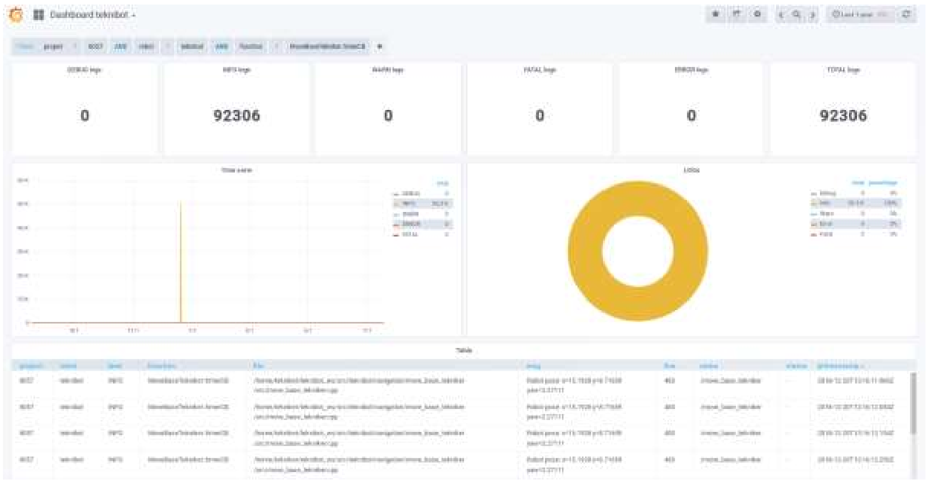


Figure 4.28. Example of a web dashboard created with Grafana.

4.3 Robotframework validations

This section is devoted to the validation of the proposed architecture, in terms of its usability and generalisation capacity. For this, the same manipulator robot has been adapted to be used in two different use cases: 1) an industrial use case where a single MMR performs a precise aileron inspection task using an ultrasonic sensor; 2) a non industrial use case at which the MMR is equipped with pest inspection and treatment tools. Apart from being representative enough to prove the generalisation capacity of the framework, these use cases are also useful to introduce future application developments. In concrete, this section presents the key configuration changes required to reuse the real-time data collection tools, diagnosis and error handling modules or user interface in order to reduce the time required to develop new mobile manipulation applications. It will also demonstrate how to create new plugings for novel tasks by using the available ones as templates, and how to extend the navigation capabilities as well as the recovery behaviours, if required.

4.3.1 Setup and Configure Robortframework

The current section provides a guide with the most relevant steps to setup a new robot using Robotframework, deploy available infrastructure for diagnostics or logging, and modify the most relevant configuration files. As commented above, the color palette used in Figure 4.21 informs about the customization possibilities of the the main architecture modules. The turquoise components are provided by the architecture, but they may require slight configuration changes. Some of these modules can also be used as templates to develop new customized ones. The gray components are ROS standard packages or modules developed in previous projects. These packages can be directly reused or modified on demand, depending on the application requirements. The modules on the Abilities Layer are most likely developed using pluginlibs. Lastly, the blue colored Task Planner modules are closely related to the application and are developed in close collaboration with domain experts.

4.3.1.1 Robot setup

On the one hand, each robotic platform needs its drivers for the platform and the sensors that it carries. There are several ROS-enabled robots. If a robot is not supported to be run in ROS, it will be necessary to implement its corresponding ROS driver. There are also several robotic sensors that are supported by official ROS packages, and many more supported by the ROS community. On the other hand, the robot needs to offer the following ROS services which are used by the Robot Manager module:

- **/start_robot** (messages of type *std_srvs::Trigger*). This service is called to start the robot. This is where the robot should perform tasks like enable the motors and check system components status.

- **/stop_robot** (messages of type *std_srvs::Trigger*). This service is called to stop the robot when the system exits from an operation mode. This is where the robot should perform tasks like stop and disable the motors.

- **/notify_status** (messages of type *std_srvs::SetBool*). This service is called when the system wants to notify that it is ready to enter an operation mode, or the operation mode starts. This is where the robot should perform tasks like show a light code, beep a sound or even talk.

4.3.1.2 Robotframework setup

Once the robot is setup, it is time for integrating the application specific modules and preparing the ROS launchfiles to start the system. The *robot_bringup* package contains configuration files to launch Robotframework. These configuration files have to be edited to add robot specific nodes and suit it to application needs:

- **Platform configuration**: The `robot_bringup/launch/platform/robot.launch` file is edited to add specific platform ROS driver nodes. If the added node accepts parameters, it is necessary to state them through a YAML file in the `robot/bringup/launch/platform/config/` directory, and load them from the launch file, using the *rosparam* tag.
- **Sensors configuration**: Specific sensors ROS driver nodes are added by editing the `robot_bringup/launch/sensors/robot_sensors`.launch file. Again, parameters should be defined into YAML files in the `robot_bringup/launch/sensors/config` directory, and load them from the launch file using the *rosparam* tag.
- **Logger configuration**: The logger configuration parameters are set in the file `robot_bringup/launch/logger/config/logger_parameters.yaml`. These are the parameters available for the Logger:
  – **buffer_size** (int, default:1000): The size of the internal circular buffer used to store the log messages before being sent.
  – **robot** (string, default: "my_robot"): The name of the robot. This information is included in each logged message.
  – **project** (int, default: 0): The project number. This info is included in each logged message.

- **send_period** (double, default: 1.0): The rate at which messages are sent to the log server (in seconds).
- **max_log_file_size** (int, default: 1000000): Maximum log file size before rotating.
- **log_to_file** (bool, default: true): Whether or not to save logs to file.
- **log_location** (string, default: ""): Path in which log files will be created.
- **rabbitmq_heartbeat** (int, default: 10): Heartbeat timeout value in seconds to ensure that the application layer promptly finds out about disrupted connections.
- **hostname** (string, default: "52.47.176.128"): The host name of the log server. It is possible to set the name or IP address of an existing log server, or to deploy the log server locally (and set this value to localhost).

- **Operation mode configuration**: The Robot Manager module is the responsible for loading the operation modes at system startup. Robotframework provides the here presented `targets_plan_operation_mode` which is reusable for a wide range of mobile manipulation applications. However, if other operation modes are needed, they have to be added to the `robot_bringup/launch/manager/config/modes.yaml` file.

- **GUI configuration**: Parameters for the Robot GUI module are located in the `robot_bringup/launch/gui/config/gui_parameters.yaml` file. This file can be used to set the GUI width or height, to visualize a custom logo on it, or to change the update rate or thresholds of specific information being shown on it (e.g. batery_level, robot_status).

4.3.1.3 Create a new task plugin

As commented above, the proposed architecture is designed to allow implementing new robot tasks that make use of the robot abilities without having to touch the core of the framework. This is achieved by using ROS pluginlib, a package that provides tools for writing and dynamically loading plugins using the ROS build infrastructure. New tasks, both application specific tasks and recovery behaviour tasks, must be implemented using C++ programming language following the steps bellow:

- Create a new package for the task that depends on ROS pluginlib and the *op_mode_targets_plan* package delivered with Robotframework.
- Create a new class that inherits from *op_mode_targets_plan::TasK*. This class will have to implement three main functions:
  - **initialize()**: Called initially when loading the plugin. Creates the TaskAction that defines interaction with the State machine in Figure 4.25.
  - **initParams()**: Parses the parameters defined in the plan when loading the plugin. This modifies the behaviour of the Task.
  - **executeCB()**: Coordinates the calls and results with the robot abilities in order to execute the task.
- Create a new *TaskName_plugin.xml* file. This file is used by the ROS ecosystem to acknowledge the availability of the plugin and permit loading this peace of code dynamically when needed.

In the following sections three different task plugins are presented. These plugins are available with Robotframework and can be used as templates to create new customize ones:

- **Croinspect-inspection-task**: Focused on airplane inspection through an ultrasonic sensor.
- **Greenpatrol-inspection-task**: Focused on finding plagues on tomato plants.
- **Greenpatrol-treatment-task**: Focused on treating the plagues by precisely spraying pesticide.

4.3.1.4 Deploying the Diagnostics architecture

The hardware based diagnostics are done using the ROS standard diagnostics mechanism. The core of the diagnostic system is the reporting mechanism. Reporting is carried out by message publication on the topic */diagnostics* using the *diagnostic_msgs/DiagnosticArray* data type. Every ROS hardware driver node is responsible for publishing the following data:

- **Component name**: This is the name of the component in the system.
- **Operational level**: Three levels of functionality are defined:
  - **OK**: Everything is running as expected.
  - **Warn**: There is an unexpected behaviour that should be resolved and may affect operation.
  - **Error**: There is a problem that should be fixed; the component cannot be relied upon to operate correctly.
- **Hardware id**: If applicable, this identifies the specific hardware running.
- **Hardware specific data**: Hardware specific data captured in string key value pairs.

4.3.1.5 Deploying the Log architecture

An internal log architecture for Robotframework is already available in the cloud hosted in Amazon Web Services (AWS). However, the Log architecture can be easily deployed in any PC as it is available as a set of open source Docker containers. A DockerFile containing the configuration for several docker images with elasticsearch, grafana, rabittmq and a a few extra configuration steps are the only requirements to deploy a personal database and to start recording desired data from the robot.

Step-by-step tutorials are available at Tekniker's personal Gitlab repositories.

4.3.2 CRO-INSPECT: an industrial aileron inspection use case

Highly integrated composite parts are superior to metals in terms of weight-specific stiffness and strength. The aerospace sector tries to exploit these advantages in order to reduce the structural weight and, as a consequence, to reduce the fuel consumption, leading to operational costs reduction. However, the inherent complex structure of these composites makes more difficult its inspection activities. Currently, most inspections are performed manually, leading to two main disadvantages: an excessive amount of time and the possibility of potential safety failures due to human errors. It is therefore necessary the development of more flexible, more reliable and faster Non Destructive Solutions (NDS). To overcome these issues, Ultrasonic Testing (UT) techniques have demonstrated to be a suitable NDS solution for the inspection of large and hidden composite areas. Also, the automation of inspection processes has been envisioned as a strategic solution. The integration of robotics in manufacturing lines has proved improvements in process reliability, reduction of manufacturing times and, therefore, costs. Nevertheless, many industrial operations cannot be fully automated at a reasonable cost. In such a case, human-robot cooperation may be the most cost-efficient and productive solution.

This is precisely the scope of the CRO-INSPECT European project [238]. The project seeks the development of a sound aircraft inspection solution with enhanced robotic capabilities, using advanced UT techniques. The proposed flexible robotic assistant solution helps operators to carry out faster and more reliable inspections of complex composite parts, such as low access areas or large areas. One main goal within the project is the development of a mobile manipulator assistant robot, equipped with force and ultrasonic sensors that offers flexible aileron inspection services within different areas in the factory.

To achieve these goals, the robotic solution must integrate navigation, manipulation, and perception abilities while following the high-level instructions from a production management system or by individual operators that require their help. Apart from generic tasks, such as the global and precise navigation tasks to reach the area where the aileron is located, an aileron inspection task must also be integrated together with the monitoring and visualization of the obtained data or the generation of alerts to inform the operator about important events. This robotic solution has been developed based on the proposed Robotframework, as it is depicted in Figure 4.29.

Next subsections illustrate how the Robotframework has been adapted to cover the needs of the CRO-INSPECT project as well as how the aileron inspection tasks are carried out within the scope of the project.

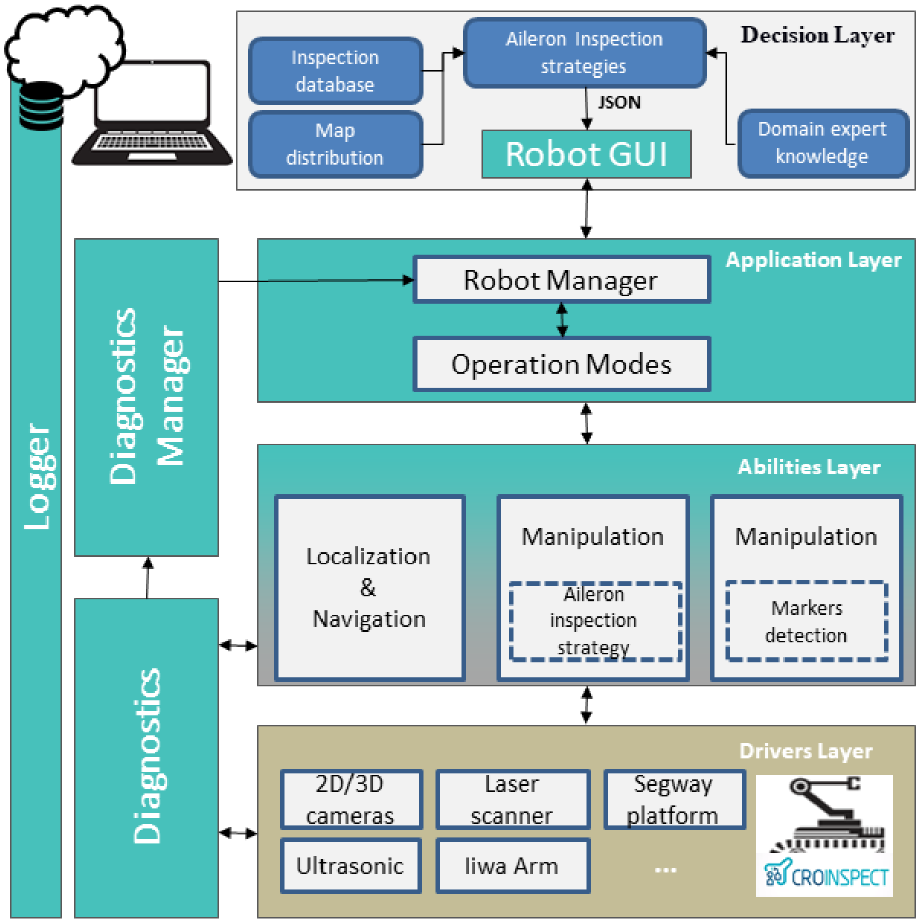


Figure 4.29. Robotframework architecture customized to the CRO-INSPECT use case.

4.3.2.1 Drivers Layer

The Drivers Layer includes the modules that allow interacting with the robot platform sensors and actuators. An overview of the specific robotic system used for validation purposes is shown in Figure 4.30.

The mobile platform consists on the Segway RMP 440 Omni Flex [239] with mecanum wheels. The platform is equipped with an on-board PC, a Velodyne 3D laser scanner [240] for obstacle detection and two OS32C safety laser scanners [241] for obstacle detection, mapping and navigation. A KUKA LBR iiwa manipulator [242] has been mounted on the middle-left-side of the platform to allow inspecting the Aileron with an ultrasonic sensor. The inspection system consists of an IDS RGB 2D camera [243] for precisely referencing the robot with respect to the aileron and the UT sensing Rollscan shown in Figure 4.31.

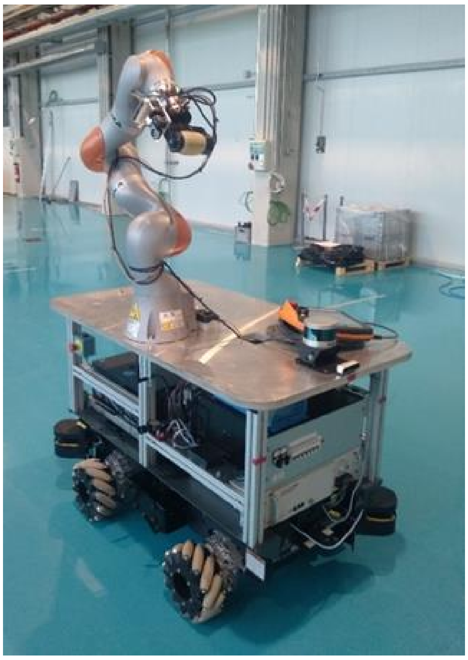


Figure 4.30. Mobile platform.

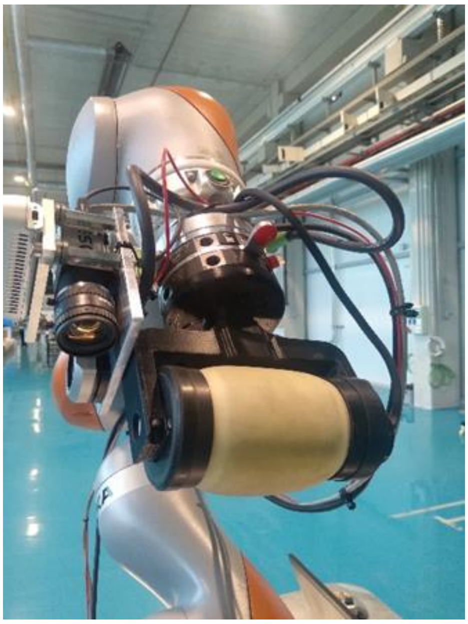


Figure 4.31. NDT Rollscan and 2D camera.

4.3.2.2 Decision Layer

The robotic inspection of complex composite parts involves that a MMR (with all corresponding hardware components such as cameras, laser scanners, manipulator, etc.) freely navigates through an industrial workspace to position in front of an aileron using the global navigation. The global navigation needs to be corrected to precisely position in front of the aileron with an accuracy under 1 cm using markers. Then, the robot is ready to perform the aileron inspection task, making use of force sensors and an ultrasonic inspection tool.

This Robotframework instance must consider the creation of aileron inspection plans, its execution, monitoring and error handling while precisely orchestrating the different modules taking part on the process.

The aircraft flap inspection strategy is defined through inspection plans that are manually generated by an operator. This fact is represented in Figure 4.29 by the *Aileron Inspection Strategies* module. Figure 4.32 represents a visual example of a plan that an operator could define, where the workspace at which the mobile manipulator navigates is represented as a sequence of six target positions. Additionally, at the targets marked as red points (T2, T4 and T6), the mobile manipulator has also to perform inspection operations, by using the robotic arm with the ultrasonic equipment. The blue rectangles represent artificial markers that are used to position the mobile robot precisely with respect to the aircraft flap. The generated plan is then loaded on the robot using Robotframework's GUI which permits the initialization of the plan and a continuous monitoring of its execution status.

The plans are generated based on domain knowledge and the distribution of the map of the different ailerons and robots, following the meta-model

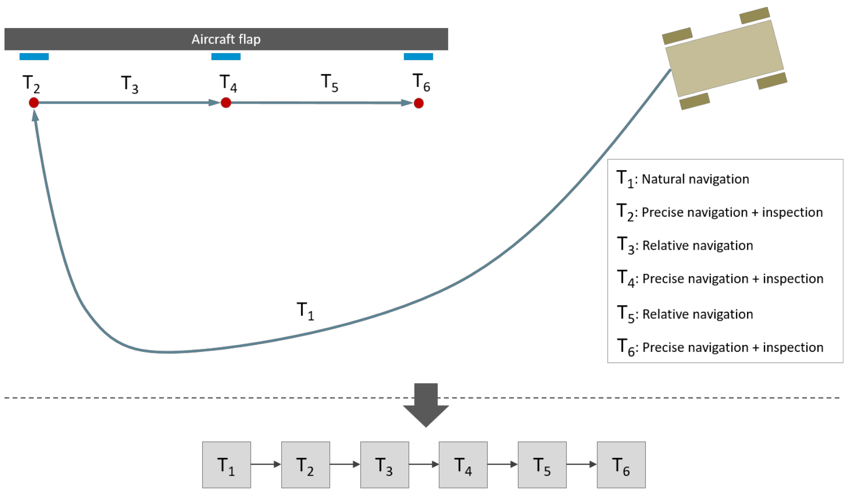

Aircraft flap
T_2
T_3
T_4
T_5
T_6
T_1
T_1: Natural navigation
T_2: Precise navigation + inspection
T_3: Relative navigation
T_4: Precise navigation + inspection
T_5: Relative navigation
T_6: Precise navigation + inspection
T_1 → T_2 → T_3 → T_4 → T_5 → T_6

Figure 4.32. The CRO-INSPECT aircraft inspection operation represented as a sequence of targets.

and by using the JSON notation format presented in Section 4.2. List 4.1 presents the piece of the plan implementation corresponding to the three first steps [T1-T3] of the aircraft inspection operation presented in Figure 4.32. The plan begins with a *Natural navigation* (navigationType 1) to get close to the aircraft flap to start with the inspection operation (T1). Second, to perform the inspection task the robot must improve its position to locate itself in front of the aileron, using the markers-based *Precise navigation* (navigationType 3). Once the robot is correctly positioned, it can start with the inspection strategy, developed as the *CroinspectInspectionTaskPlugin*, using the manipulator force sensors and an ultrasonic inspection tool. These two steps are presented together as T2. Third, due to the extended length of the aileron and the reachability constrains of the robot arm to inspect the complete aileron at once, it is necessary to perform an additional *relative navigation* (navigationType 2) to reach a further aileron section (T3). The rest of the target positions (not shown in the example) are achieved by repeating actions similar to T2 and T3, until the complete aileron is inspected. These robot abilities are later explained.

Listing 4.1: Example of a simple CRO-INSPECT Inspection Plan.

```
[
  {
    "navigate": true,
    "navigationType": 1,
    "navigationTrials": 3,
    "targetPose": {
      "frameId": "map",
      "x": 12.5,
      "y": 25.46,
      "theta": 1.57
    }
  },
  {
    "navigate": true,
    "navigationType": 3,
    "navigationTrials": 2,
    "poseOffset": {
      "x": 0.015,
      "y": -0.95,
      "theta": 3.102
      },
    "tasks": [
      {
        "name": "inspection",
        "type": "croinspect_inspection_task::
            CroinspectInspectionTaskPlugin",
      }
    ]
  },
  {
    "navigate": true,
    "navigationType": 2,
    "navigationTrials": 2,
    "targetPose": {
      "frameId": "map",
      "x": 1.0,
      "y": -0.2,
      "theta": 0.0
    }
  }
]
```

4.3.2.3 Application Layer

The operation mode used in this application corresponds to the generic one described in Figure 4.25: the so-called *targets_plan_operation_mode*. The operation mode executes the plan generated in the higher decision level described before, parsing its indications and ensuring the different configuration specifications.

For the example defined in List 4.1, the robot will therefore perform a global navigation first to approach the inspection area. This is defined by the navigationType 1, which contains the x, y and theta coordinates in the map. If it fails, it will retry the navigation three times as specified in the plan. Note that in this example, there is no recovery action defined for navigation failures. Once it reaches the aileron zone it performs a more precise navigation approach to the aileron (this time having two trials for it). In this case, the x, y and theta values correspond to the maximum relative offset error from the robot to the marker position, that the precise navigation module must ensure. After being precisely located in front of the aileron, the robot performs the main application task as defined in the *CroinspectInspectionTaskPlugin*. The last two activities are then repeated until the complete aileron has been inspected. In this case, the task functionality together with its recovery behaviours is entirely outsourced to the specific plugin, described in the following subsection. Therefore, no task failures are explicitly specified in the plan.

Although this use case is a very simple example, as it will be presented in the next validation use case, the plan may contain additional parameters that make the framework more flexible for complex manipulation tasks that require more detailed configuration changes.

4.3.2.4 Abilities Layer

To successfully perform the above presented application, the robot must present the following main abilities: create a map of the working environment; localize itself on it; navigate to the working area in which the aileron is located; perform a precise docking manoeuvre to let the platform, and specially the arm, referenced and with a proper reachability to the aileron; and, finally, perform the NDT inspection. Next paragraphs describe how these abilities are achieved withing the customized *Robotframework* architecture.

Localization & Navigation

To generate a virtual representation of the environment, the robot uses a process called SLAM (Simultaneous Localization and Mapping) to build the map while trying to localize itself on it. The experiments have demonstrated that poor mapping leads directly to an unstable pose estimation, so, it must be done carefully limiting the velocities and accelerations of the platform, specially the rotational ones. In addition, during this procedure the robot is teleoperated and the expertise shows that creating a good map can be the most time-consuming task, but there are some general recommendations: Avoid rough movements, as they difficult the scan-matching; Drive some more meters to get plenty overlap between the start and the end of the loop. This improves the loop closure, which is the process of overcoming the drift accumulated in the robot trajectory over time; Visualize what the robot captures with the laser, to avoid loss of relevant information.

Once a reliable map is built and based on the information obtained from the sensors, the robot can estimate its pose with respect to a known reference frame. For that estimation the AMCL probabilistic approach is used. This approach uses a particle filter in which each particle corresponds to an individual map of the environment. It computes an accurate proposal distribution considering not only the movement of the robot but also the most recent observation. Nevertheless, although the robot tracks and updates its position and orientation autonomously during navigation, the localization must be known on start-up, giving it externally or setting a specific initial pose on the map.

Now, the *Natural navigation* available with Robotframework can be used to autonomously navigate through the working area. Autonomous navigation refers to the capability of going from one point to another without any external support, which includes the abilities of path planning, trajectory tracking and obstacle avoidance. The navigation system divides the ground of the map in cells, getting the corresponding grid-map. This allows a more efficient path planning and obstacle avoidance. Accordingly, it operates in the discrete space with the consequent loss of information.

The map in Figure 4.33-left is represented as a grid cell where each cell (pixel) depicts if that area is occupied (high value) or not (low value). Apart from that, obstacles also have an inflation radius. This means that the cells around the object have values between occupied and free, their cost is reduced gradually. The meaning of the colors is the following: The white rectangle is the robot and the green line the global path that it follows; Black color represents the obstacles recorded in the static map itself; The blue and red areas show the inflation radius of the obstacles; The yellow points correspond what the lasers are acquiring.

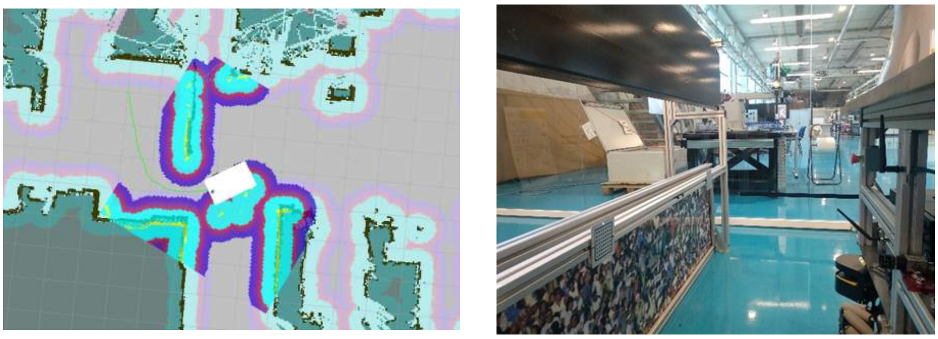


Figure 4.33. CRO-INSPECT testing-area map used by the global navigation module to reach the working zone where the aileron is located (left). Closer view of the referencing of the robot to the aileron with the help of markers and a 2D camera on board (right).

The planner uses the inflation radius to avoid driving close to the obstacles if there is a safer alternative. When an obstacle appears in the middle of the current navigation path, the system uses first the local planner to try to avoid it. If the algorithm takes too much time or calculates paths with distances above a threshold, it is the global planner which takes the control and recalculate a new global navigation path. If there is no other way to reach the specified goal, the system aborts the process, notifies it and waits until a next goal is received.

For the precise approach to the aileron, some markers have been located along the base of the aileron and a 2D Camera has been integrated to the mobile base as seen in Figure 4.33-right. For the calibration plates location, a vision system has been developed which generates as an output the pose (translation and rotation) of the centre of the sheet regarding to the camera. These markers are then tracked by the *Robotframework Precise navigation* as references and used to improve the robot position in front of the aileron with an accuracy under 1 cm. Both navigation types have been exhaustively tested, comparing different algorithms' limitations and capabilities in [244]. The main results will be summarized in the experiments section.

Manipulation: Aileron Inspection Strategy

Once the robot is precisely positioned in front of the aileron, the inspection strategy takes place, which has been developed as a plugin and represents standalone integration cases within the architecture presented here: the so-called *croinspect_inspection_task::CroinspectInspectionTask Plugin*.

The force control-based aileron inspection task inspects a rectangular pattern along the x axis of the aileron, based on the use of a mobile robot and NDT system. For that purpose, the inspection strategy is divided in two main tasks as seen in Figure 4.34: First a marker detection attached on the aileron for precisely referencing the robotic arm with the aileron. Second, the rastering shape trajectory execution for the aileron inspection.

On the one hand, the fact that the manipulator is located on top of a mobile platform carries the uncertainty of knowing how the manipulator is positioned with respect to the aileron. The autonomous navigation accumulates a positioning error of some centimetres and although the

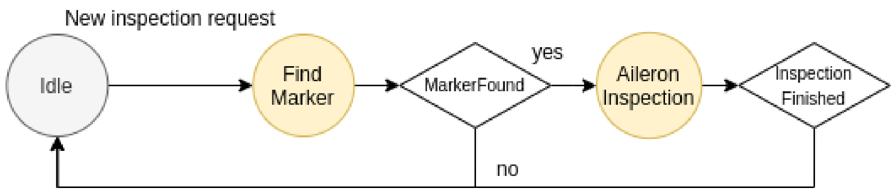


Figure 4.34. Aileron Inspection strategy workflow.

deviation is compensated with the precise positioning process based on vision techniques, the robot doesn't locate always exactly at the same point. Therefore, executing a referencing strategy is mandatory in order to absorb the remaining error.

The *Find Marker* referencing consists in moving the robot to a position where the calibration plate is within the field of view of the on-board camera. The same marker detection module used for referencing the mobile platform is now used to obtain the $F_{\text{caltab-in-camera}}$ frame in Figure 4.35, with the peculiarity that the arm is much more precise in positioning, reaching an accuracy of millimetres instead of centimetres.

The trajectory for the ultrasonic test inspection of the aileron is a raster trajectory that is defined offline as a set of points. Each point corresponds to a cartesian pose composed by: X, Y, Z translation axis and rotation angles, with a ZYX rotation order. These trajectory is here referenced to the markers as $F_{\text{traj-in-caltab}}$ frame and represented as the green trajectory. As these positions are referenced with respect to the coordinate origin of the aileron in the CAD model, they must be transformed to the IIWA robot base, which is represented in Figure 4.35. The $F_{\text{camera-in-robot\_base}}$ frame is fixed and available on the robot_description.

On the other hand, the *Aileron Inspection* state is in charge of executing the rastering pattern trajectory. The marker referencing is used over each

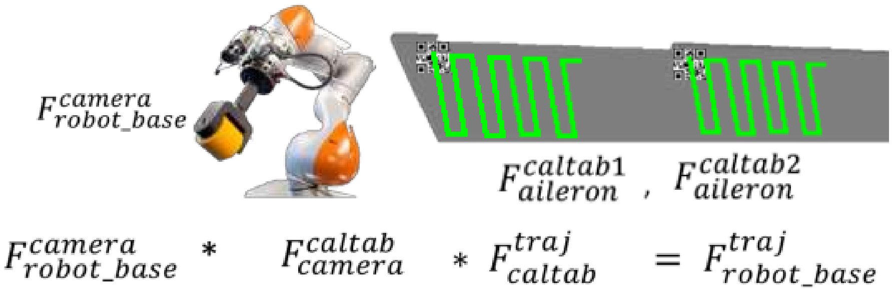

$F^{camera}_{robot\_base}$
$F^{caltab1}_{aileron}$, $F^{caltab2}_{aileron}$
$F^{camera}_{robot\_base} * F^{caltab}_{camera} * F^{traj}_{caltab} = F^{traj}_{robot\_base}$

Figure 4.35. Trajectory in robot base frame.

point of the trajectory, the last transformation is applied, and the robot starts moving to that point with the impedance control mode activated. While the arm is moving along the aileron surface following the inspection trajectory, the NDT sensor is recording each measured data. In case that an attenuation is detected in the ultrasounds signal, lower than the configured threshold, the system generates an alarm signal. This signal is connected directly to one of the inputs of the robot, and when it is activated, the robot stops and starts a rescanning process in the current point. This procedure allows checking if the signal loss has been due to a deviation of the tool or because of a real defect in the aileron.

4.3.2.5 CRO-INSPECT Experiments

This section presents a summary of the validation tests performed in the context of the European project CRO-INSPECT within the simulated and real working environments. More detailed tests can be found in the official deliverables of the project or in the extensive navigation tests article in [244]. The aim of the here presented tests is the assessment of the following features: first, the correct integration of Robotframework with the different robotic modules; second, successful execution of global navigation, accurate navigation and inspection abilities using JSON format plans; third, the logging capabilities of the system to generate data; finally, the verification of the system requirements proposed in Table 4.3. Every application execution has been performed using JSON format plans similar to the ones shown in List 4.1.

In order to calculate the best position of the KUKA LBR iiwa robot arm, a Gazebo simulation tool was used. This tool allows to find the best height and position of the arm with regard to the chassis of the Segway mobile platform choosing different positions and checking the reachability and capability of achieving an inspection trajectory. Figure 4.36 shows the Segway Flex Omni robot with the arm in a selected position for the test, the aileron as well as the structure to attach it. The simulation has also been used to test preliminary global and precise navigation, and also to see the manoeuvrability of the arm to perform the aileron inspection trajectory. The simulation allows the deployment of the architecture and all of its modules within a safe environment, but unfortunately the physics are not yet the most reliable ones so the results for timings, forces, accelerations, etc. obtained here cannot be taken into account.

To evaluate the accuracy of the mobile platform in the real scenario a laser tracker has been used. A laser tracker is an external unit that measures distances with micro metric accuracy. The pose estimation of the robot's localization algorithm is tracked while its real position is saved with the laser tracker, and afterwards both data are associated based on their timestamp.

For the evaluation tests, the robot starts in a big workshop and navigates to a laboratory next to it, passing through a narrow entrance and travelling approximately 20 m to its destination. To proceed with the precise navigation, the robot must finish its trajectory inside an area where the camera can detect the marker. Statistical values obtained from the anal-

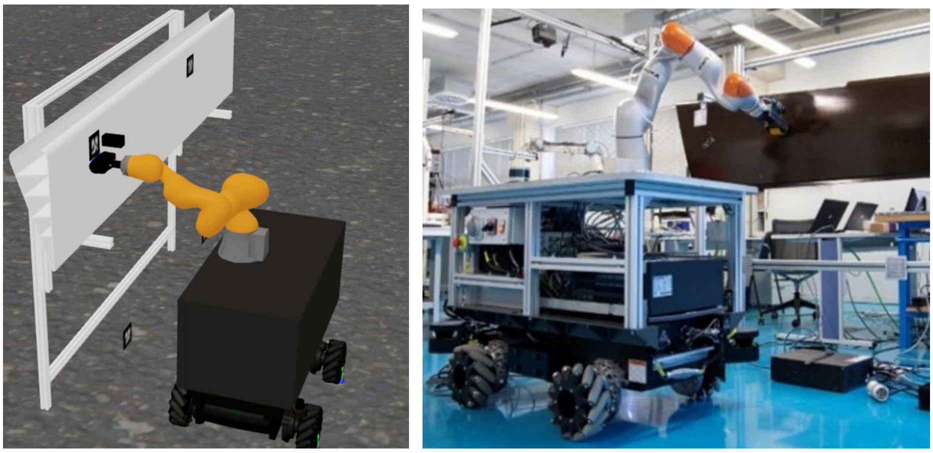


Figure 4.36. CRO-INSPECT mobile manipulator during simulation (left) and real (right) scenario experiments.

ysis of global navigation tests show an average deviation of under 0.2 m in translation and 2° in rotation. Then, the marker is detected, and the precise navigation positioning is executed. In 25 experiments the robot, starting from different points, corrects its position using the marker as reference. Thanks to the vision system the robot can reach the goal accurately, with a tolerance of 20 mm in x, 3 mm in y, and 0.28° (0.002 rad) in rotation (set by software due to the limitations of the control of the platform itself). In all tests, a mean error of 217 mm is achieved with a standard deviation of 104 mm and a minimum improvement of 68 mm.

Finally, Table 4.4 presents the timing required for each step to inspect a 2.1 m of the aileron. In summary, the complete process takes 16 minutes and 32 seconds. However, this value can change depending on some factors related with the navigation process. During the autonomous navigation, the localization of the robot has a direct effect in the global navigation. If it is well located the movements are smother and faster. The obstacles that the robot may encounter on the way also have an important impact on the performance. During the precise positioning, depending on the deviation from the goal and the number of corrections needed, the required time can vary. The inspection process has resulted to be quite static as less external disturbances are encountered during the process resulting on a stable avarage time of 45 s per each 0.1 m.

Table 4.4: Performance of NDT aileron inspection process with a mobile manipulator.

Task	Duration
Global Navigation (8 m)	32 s
Precise Navigation	3 s
Aileron Inspection (1 m)	451 s
Relative Navigation (1.5 m)	6.5 s
Precise Navigation	2.5 s
Aileron Inspection (1.1 m)	497 s

4.3.2.6 CRO-INSPECT conclusions

Robotframework has been partially built in the context of CRO-INSPECT, considering its generalisation as a main goal in order to be reusable for future projects that require mobile manipulation robotic capabilities. It has been used here as a first time being able to test a bunch of characteristics of Robotframework and also to develop new ones proving its extendibility.

The high-level parametrized plans have been generated considering the working space map distribution and domain expert knowledge for the aileron inspection strategy. The plans follow the meta-model and its corresponding JSON notation introduced in the previous section (SR4), and the *Robotframework* GUI is used for starting the plan and visualizing system status (SR7). The plans contain natural, precise and relative navigation tasks that are not only useful in the context of the current project, but will be reusable for future ones. The plans also contains an application specific task integrated as pluginlib. In this case, the aileron inspection makes use of a camera, force sensor and a NDT ultrasonic sensor. The main *Robotframework* state machine presented in the *Application Layer* of the previous section needed no change in order to execute these plans. It is expected that this state machine will not require any change also for future projects, due to the generalisation and extedibility characteristics taken into account during its development. Historical events have been logged to obtain metrics such as the position of the robot or the results obtained from the NDT inspection sensor. This information has been manually used to monitor the correct performance of the system and detect incidents (SR6), but in the future this should be an automatic process integrated in the cloud.

The Abilities Layer is usually the most customizable layer. The abilities that the robot mus exhibit differ from one application to another. In this project, the robot has validated the following abilities: create an static map in which it will later localize itself based on board laser scanner, IMU and odometry data (SR1); the Robotframework global navigation ability has been integrated and a new marker-based Precise Navigation has been developed and tested (SR2). The precise navigation has been integrated in Robotframework and is expected to be useful for future projects where higher positioning accuracy is necessary. The main perception abilities (SR3) consist on: laser scanner data, for localization and obstacle detection purposes; the marker detection module, which has been used first for the precise navigation and then reused for the arm referencing with respect to the aileron. Finally, an aileron inspection ability based on force and ultrasonic sensors has been developed and integrated as pluginlib, which is expected to be less reusable in the future as it is very application specific. This will be done and proved in the following use case.

The simulation tests have been used to validate the following key points: the positioning of the arm on top of the mobile platform ensuring an optimal reachability to the aileron; The integration of Robotframework with navigation and manipulation modules using the plans; a first instance of the precise navigation module in a safe and controlled environment.

The real scenario tests have been used to validate the complete execution of an aileron inspection plan. The performed tests have shown that the global navigation is able to reach an average accuracy of 20 cm. This accuracy can be enhance up to the 0.5 cm using the markers-based precise navigation. Also timings have been presented of how much time takes to perform each one of the tasks to carry out a complete aileron inspection task. It has been measured a total time of 16 minutes and 32 seconds to inspect an aileron of 2.10 m long, and starting the mobile platform at a distance of 11 m away from the aileron.

4.3.3 Greenpatrol: an agricultural pest inspection and treatment use case

The European agriculture land surface is decreasing due to deforestation and urbanization while population continues increasing. In order to achieve a more sustainable business model, protect the crops from adverse weather conditions and control the temperature and water of the plant, the greenhouse production is continuously growing during the last decade. However, the presence of warm, humidity conditions and abundant food under protected structures provide favourable habitats for pest development, being this the main threat to production and productivity of greenhouse crops worldwide. Digital farming can help through sensors, robotics and data analysis to automatically maintain and monitor greenhouses, making cropping system smart and, thus, enhancing the agricultural productivity. In this context, the combination of different robotic skills is necessary to perform early pest detection. The challenge here is to detect and eradicate insects in their early eggs state, which can measure as less as 0.3 mm. For this purpose, advance perception and dexterity skills need to be merged to automatically obtain close and good quality pictures of the pests from different sides of the leaves. The information obtained must be processed to generate efficient high-level instructions to command the robot according to an Integrated Pest Management (IPM) system.

In this context, Robotframework has been customized to integrate the navigation, manipulation, and perception abilities, while following the high-level instructions from an IPM decision support system for early pest detection and treatment in greenhouses. The use of Robotframework to obtain the architecture in Figure 4.37 has remarkably reduced the development time required to perform ROS-based field robotic experiments due to efficient reuse of common modules across projects and robot platforms. To demonstrate the easy integration and the benefits of combining different robotic skills within the architecture, flexible manipulation strategies to enhance pest detection and targeted spraying have been developed. Finally, to evaluate the obtained architecture, several tests in simulated and field commercial greenhouses have been performed in the context of the European GreenPatrol project [245].

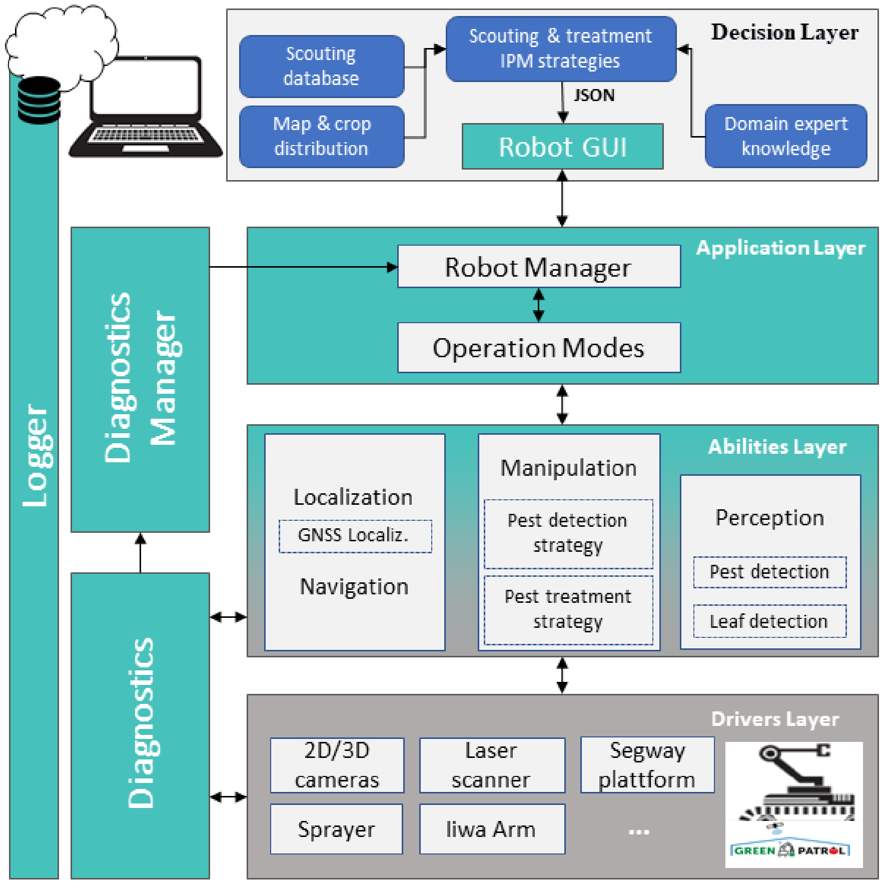


Figure 4.37. Robotframework architecture customized for the Greenpatrol use case.

Next subsections describe how this customization and experiments have been carried out.

4.3.3.1 Drivers Layer

An overview of the specific robotic system used for validation purposes is shown in Figure 4.38.

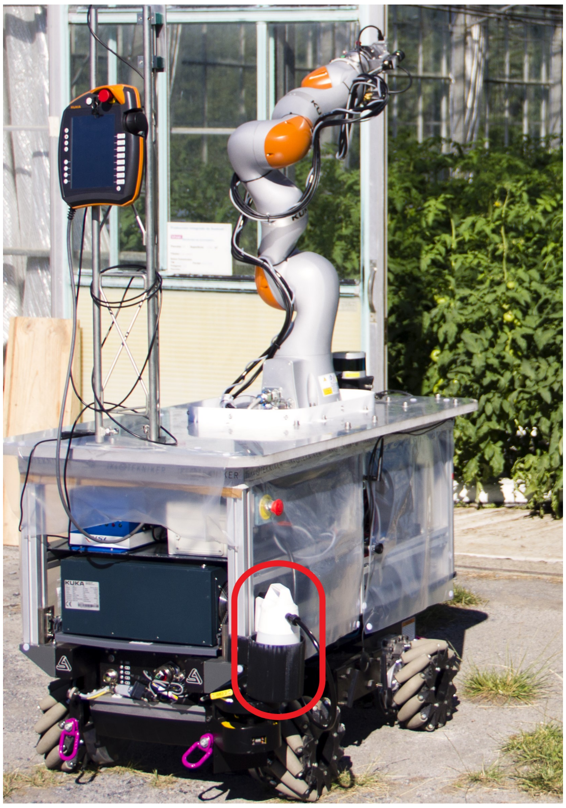


Figure 4.38. GreenPatrol robotic platform, entering the greenhouse.

The mobile platform consists on the Segway RMP 440 Omni Flex [239] with mecanum wheels to improve mobility in greenhouse narrow corridors. The platform is equipped with an on-board PC, a Velodyne 3D laser scanner [240] , for obstacle detection and two OS32C safety laser scanners [241], for obstacle detection, mapping and navigation. The absolute localization unit consists of a multi-constellation, GNSS receiver, IMU and odometry. A KUKA LBR iiwa manipulator [242] has been mounted on the middle of the platform to allow inspecting the leaves on the right and left sides. The vision system consists of a 3D RealSense camera [246], for leaves positions, and an IDS RGB autofocus camera [243], to acquire good quality pictures of the pests, as seen in Figure 4.39.

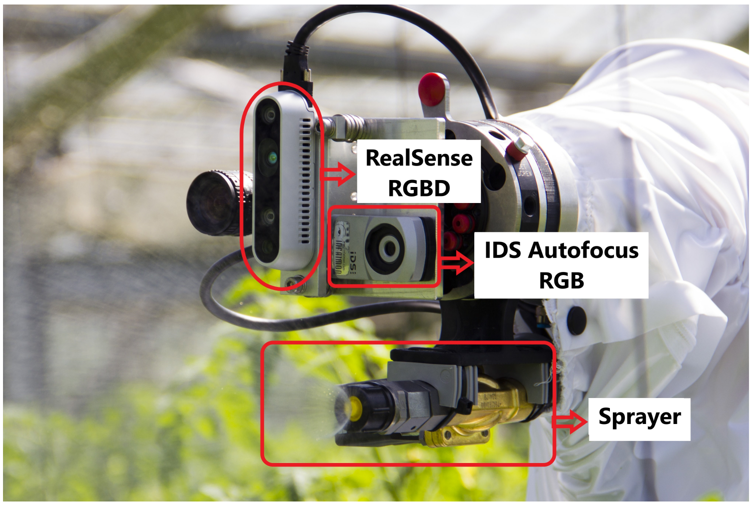


Figure 4.39. GreenPatrol pest inspection and treatment tools, mounted at the robot arms end-effector. 3D and RGB-autofocus cameras for perceptions tasks, and sprayer for pest treatment tasks.

4.3.3.2 Decision Layer

For the current application, an IPM strategy generates pest scouting and treatment plans based on domain expert knowledge, crops distribution in the greenhouse and information obtained from previous plan executions as seen in Figure 4.37 (*Scouting & treatment IPM strategies*, *Domain expert knowledge*, *Map & crop distribution*, and *Scouting database* modules, respectively). The graphical representation of an example plan for the current application can be seen in Figure 4.40, where the robot must navigate to four different greenhouse zones and perform there a pest inspection task.

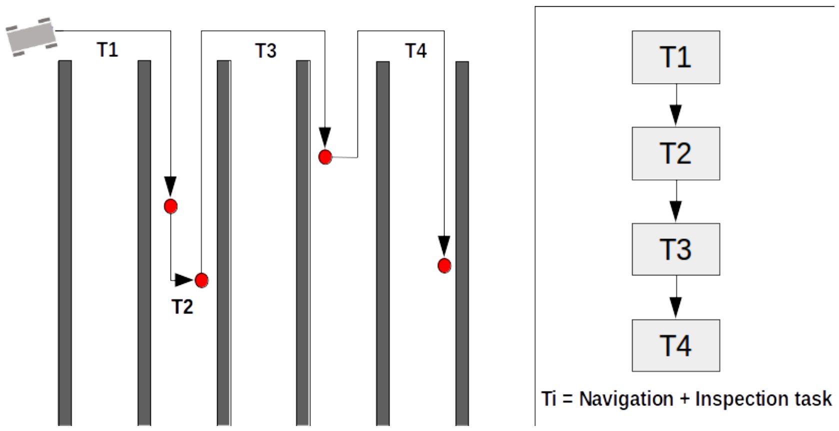


Figure 4.40. Representation of an IPM pest inspection plan composed by four targets, consisting each one on a navigation task and a pest inspection task.

The plans are implemented following the meta-model described in the previous section, by using the proposed JSON format, allowing navigation and application specific tasks based on the pluginlib concept. The parameterization of the plan tasks provides the IPM strategy with a high reconfigurability to define, for example, the zones to be inspected or the amount of pesticide to use, depending on the infection level of the plant.

On the one hand, the pest inspection process can be defined as the accurate image acquisition process on different plant zones (namely high, middle and low). The obtained pictures are transferred to the cloud once this process is finished, in order to be processed. The pest distribution results obtained from this process are used to generate new inspection and treatment plans, as later shown in the inspection manipulation strategy section.

An example of a simple inspection plan is shown in Listing 4.2. The robot navigates first to an specific area defined by the x, y and theta coordinates of the greenhouse, using a global navigation (navigation type 1). Once there, it performs a pest inspection task on different plant heights and leaf zones (*plant_height* and *row_side* parameters, respectively). Furthermore, the *id_inspection* and *date* parameters are used to link the inspection results to an specific plan and simplify, this way, the traceability on the automatically generated logs.

Listing 4.2: Simple Greenpatrol inspection plan.

```
[
  {
  "navigate": true,
  "navigationType": 1,
  "navigationTrials": 3,
  "targetPose": {
    "frameId": "map",
    "x": 12.0,
    "y": 3.0,
    "theta": 1.5708
    },
  "tasks": [
    {
      "name": "inspection",
      "type": "greenpatrol_inspection_task::
         GreenpatrolInspectionTaskPlugin"
      "params": "high_up;high_bottom;middle_up;
         middle_bottom;low_up;low_bottom;
         plant_height:1.5;row_side:right;
         id_inspection:45;date:2020-09-25"

    }
   ]
  }
]
```

On the other hand, the pest treatment process can be defined as the precise spraying of pesticide on different plant zones (namely high, middle and low), being the pesticide spraying dose at each plant determined by the IPM strategy (*dosage* parameter). An example of a simple pest treatment plan is shown in Listing 4.3. In this case, the robot navigates first to an specific area defined by the x, y and theta coordinates on the greenhouse, using a global navigation (navigation type 1). Once there, it performs a pest spraying task on different plant heights and leaf zones (*plant_height* and *row_side* parameters, respectively). The *id_treatment* and *date* parameters are again used to link the pest treatment results to the specific plans.

Listing 4.3: Simple Greenpatrol pest treatment plan.

```
[
  {
  "navigate": true,
  "navigationType": 1,
  "navigationTrials": 3,
  "targetPose": {
    "frameId": "map",
    "x": 12.0,
    "y": 3.0,
    "theta": 1.5708
    },
  "tasks": [
    {
      "name": "spraying",
      "type": "greenpatrol_spraying_task::
         GreenpatrolSprayingTaskPlugin",
      "params": "high_up;high_bottom;middle_up;
         middle_bottom;low_up;low_bottom;
         plant_height:1.5;row_side:right;dosage
         :0.5;id_treatment:10;date:2020-09-25",
      "taskFailureBehaviors": [
        {
          "name": "
             greenpatrol_navigation_failure_
             behavior",
          "type": "op_mode_targets_plan::
             GreenpatrolNavFailureBehavior
             Plugin"
        }
      ]
    }
  ]
  }
]
```

4.3.3.3 Application layer

This layer is kept the same as in the previous CRO-INSPECT use case. Due to the criticality of the satellite-based localization in greenhouses to reach a position accurately, it has been necessary to include an additional feature in the navigation stack to provide information about the localization quality.

In case of a remarkable localization quality loss, the greenpatrol_navigation _recovery_behaviour is triggered. This behaviour consists in sending the robot to a well-known greenhouse position where the localization signal is known to be strong, and retrying from there the previous navigation goal.

A second recovery behaviour triggered could be also implemented when, despite having a proper localization signal, the robot does not reach the destination with enough accuracy. This can happen because a slightly better localization is needed, or because there are obstacles on the way that the navigation module cannot overcome. In both cases, the recovery behaviour consists of waiting for a predefined time still, while playing an advertisement sound. The sound notifies the operators in the vicinity about the current robot state and, if necessary, about the need of removing the obstacle on the way.

4.3.3.4 Abilities layer

There are several challenges for developing a robotic system able to perform autonomous and continuous monitoring in greenhouses for the detection, identification and control of pests. As shown in Figure 4.41, the plants grow remarkably during the growing season affecting: (1) the localization and navigation systems, because of a constant change of the environment and the narrowing of the corridors; (2) the manipulation strategy, as the arm needs to approach the leaves to obtain good quality pictures while avoiding damaging the crops; and (3) the vision modules, dealing with changes in illumination and focus distance.

Localization & Navigation

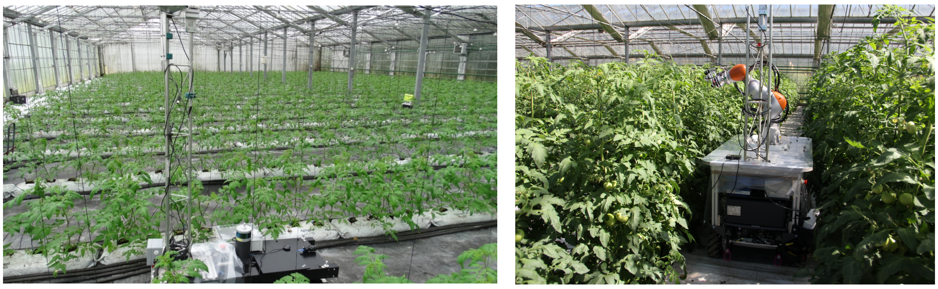


Figure 4.41. Tomato crop greenhouse evolution at the beginning (left side) and at the end of the season (right side).

At this level, ROS provides a wide range of state-of-the-art robotic algorithms:

- *GMaping* [232], for generating maps using the on board 2D laser scanners. The maps can be manually modified to include, for instance, forbidden areas for the robot.
- *Navigation Stack*, used in the *Navigation State* for planning global and local paths. It uses combined 2D laser scanners and satellite based localization to generate the velocity commands for the mobile base, while avoiding the obstacles on the unstructured greenhouse environment.
- *URDF*, for generating a combined robot description as presented in Figure 4.42.
- *MoveIt!*, used for generating and executing collision free manipulation trajectories in the *Doing Tasks* state as it will be later presented. MoveIt! tools can be also used to integrate 3D point cloud based obstacles or useful simulation tools, among other utilities.

On top of them, several additional nodes have been developed. To ensure the system requirement SR1 and freely navigate within the greenhouse,

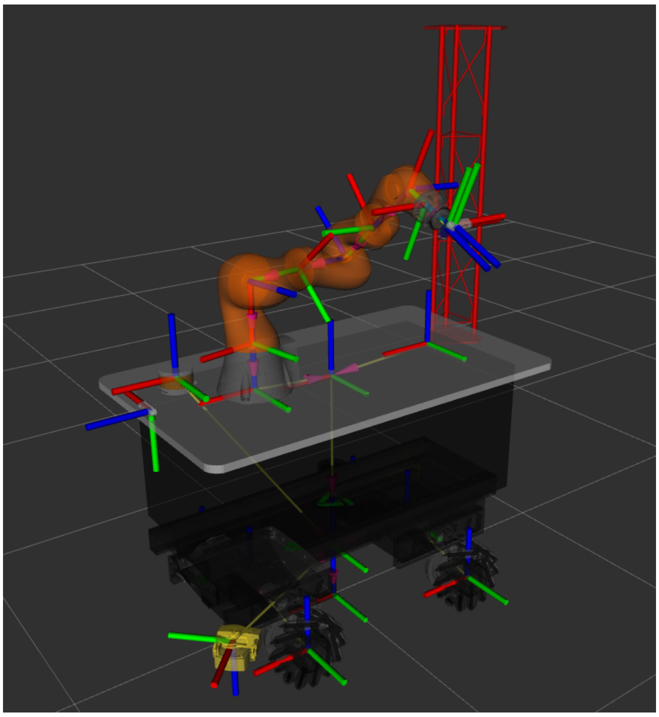


Figure 4.42. GreenPatrol robot description in ROS-visualization RViz. It presents the main platform components (mobile platform, arm, sensor...) and the transformation links between them.

the localization module benefits from the multiple signal frequencies and the higher accuracy provided by the European Global Navigation Satellite System (EGNSS) of the Galileo constellation, as explained in [77]. The navigation, then, consists of the previously presented global navigation type with configuration changes to adapt it to the greenhouse terrain. The system requirement SR3 is achieved using a deep learning model for detecting the most harmful pests in greenhouse tomato crops: Bemisia Tabaci, Tuta Absoluta and Whitefly [247]. In addition, a leaf detection deep learning model has been implemented to safely and accurately detect and approach individual leaves by using a 3D camera. Then, closer and high resolution pictures of the pests are taken as presented hereunder, answering to system requirement SR2.

Once the mobile platform navigates in front of the target plant, the robotic arm mounted on the middle-top of the mobile platform performs the corresponding pest inspection or treatment task, on right and left sides of the platform. This section presents the strategies taken and the execution workflow for each manipulation task. The tasks have been developed as plugins and represent standalone integration cases within the architecture presented here.

Manipulation & Perception: Pest Detection Strategy

In the developed pest inspection task, the plant zones to be inspected and the number of pictures that need to be taken at each zone are determined by the Pest Monitoring Index (PMI), as shown in Table 4.5 and in Figure 4.43. Lower and darker zones tend to provide more suitable habitats for the pests, resulting on a higher number of pictures required. As an example, in the high-up zone (PMI6), the robot must inspect leaves above 1m and requires two pictures to be taken, while in the middle-bottom zone (PMI2), the robot must inspect leaves in between 0.5 m and 1 m and requires 4 pictures to be taken.

Table 4.5: Pest Monitoring Index for defining the number of pictures to take at each plant zone.

PMI	1	2	3	4	5	6
N°of Photos	5	4	3	3	3	2

The manipulation strategy for pest detection consists of the workflow defined in Figure 4.44. First, the arm is moved to the next inspection zone. Second, the leaf detector model and the RGBD image are used to find leaves' poses. If no leaf is found, the arm is moved to the following

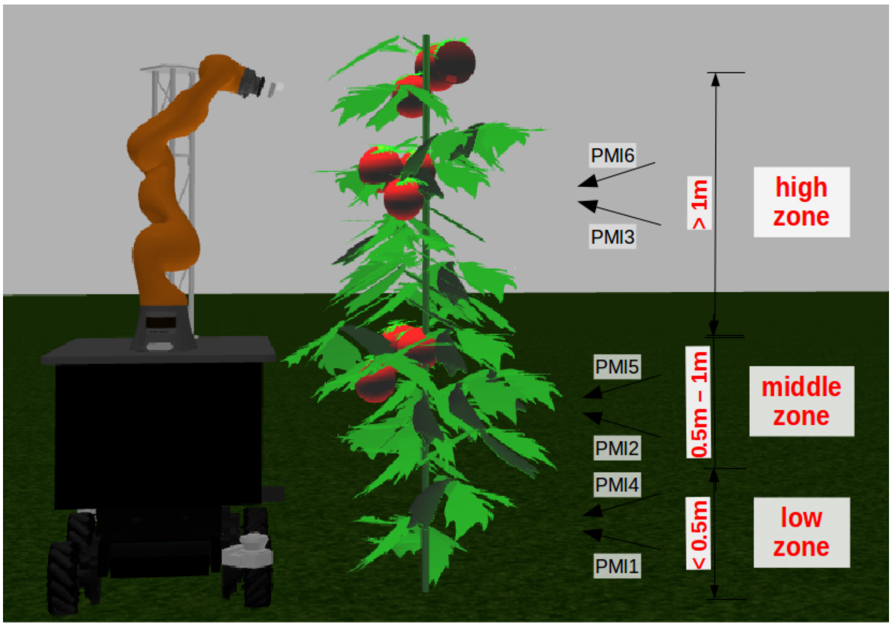


Figure 4.43. PMI values related to a tomato plant inspection with the GreenPatrol robot facing the plant at its high-up zone in Gazebo simulation.

inspection zone. Third, the arm approaches the leaves found in the previous step and takes closer pictures of them, using the RGB autofocus camera. An algorithm determines the quality of the picture. If the quality is not good enough, the arm makes a predefined small movement, and takes a new picture from there. This process is repeated until all required plant zones have been inspected.

The pictures taken in this process are saved locally and then sent to the cloud in order to be analysed by the Deep Learning (DL) model to identify infection areas in the greenhouse. The *Scouting & treatment IPM strategies* module uses these results along with additional information such as the current harvest season conditions, the working area size, the size of the plant or legal aspects on pesticides on the working country. As a result, new inspection and treatment plans can be generated.

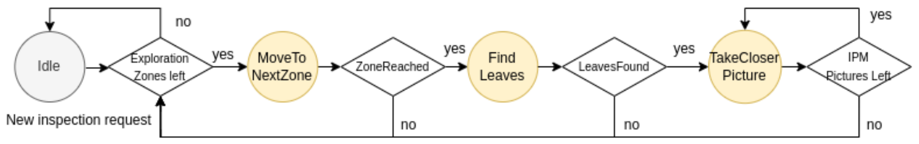


Figure 4.44. Workflow of the manipulation strategy for pest detection.

Manipulation & Perception: Pest Treatment Strategy

The manipulation strategy followed in the developed pest treatment task is represented by the workflow of Figure 4.45. First, the arm is moved to the next spraying zone. Second, the sprayer is activated and in order to cover the whole plant zone, the manipulator performs small and controlled movements until the complete dose has been sprayed. This process is repeated until all required plant zones have been sprayed.

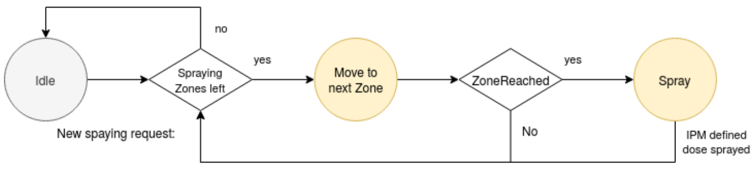
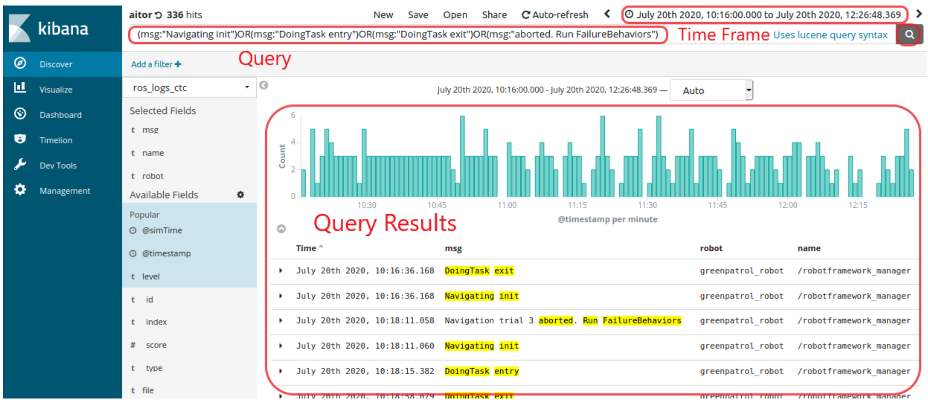


Figure 4.45. Workflow of the manipulation strategy for the pest treatment.

4.3.3.5 Monitoring

The log data is saved in non-relational databases and can be accessed through elastic-search queries as seen in the Kibana user interface presented in Figure 4.46 (up) marked in red. First, the time frame at which the test was performed must be defined. To filter the logs and search for specific information, queries can be done to the database. The dashboard can also be configured to make automatic queries of desired sensor or execution results and present them in different chart forms. Finally, Grafana could also have been used seamlessly instead of Kibana.



July 20th 2020, 10:16:00.000 to July 20th 2020, 12:26:48.369

(msg:"Navigating init")OR(msg:"DoingTask entry")OR(msg:"DoingTask exit")OR(msg:"aborted. Run FailureBehaviors")

July 20th 2020, 10:16:00.000 - July 20th 2020, 12:26:48.369 —

```
Time                            msg                                          robot              name
July 20th 2020, 10:16:36.168    DoingTask exit                               greenpatrol_robot  /robotframework_manager
July 20th 2020, 10:16:36.168    Navigating init                              greenpatrol_robot  /robotframework_manager
July 20th 2020, 10:18:11.058    Navigation trial 3 aborted. Run FailureBehaviors  greenpatrol_robot  /robotframework_manager
July 20th 2020, 10:18:11.060    Navigating init                              greenpatrol_robot  /robotframework_manager
July 20th 2020, 10:18:15.382    DoingTask entry                              greenpatrol_robot  /robotframework_manager
```

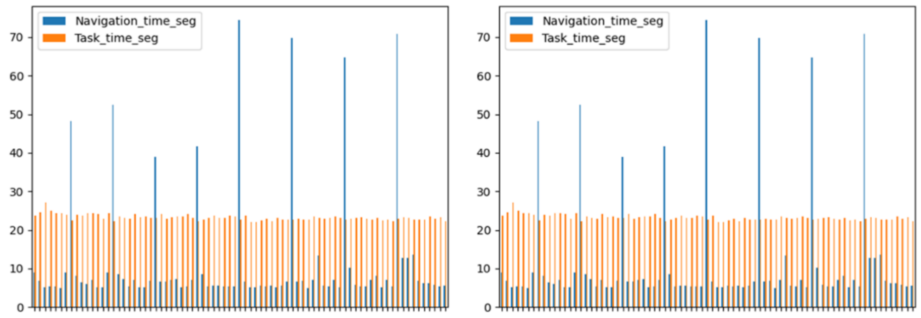


Figure 4.46. Log queries visualized in the elastic search Kibana user interface for the here presented semirandom pest inspection test (up side). The graphs present timing metrics obtained from navigation and inspection tasks during the scouting (down -left side) and spraying (down-right side) plans execution.

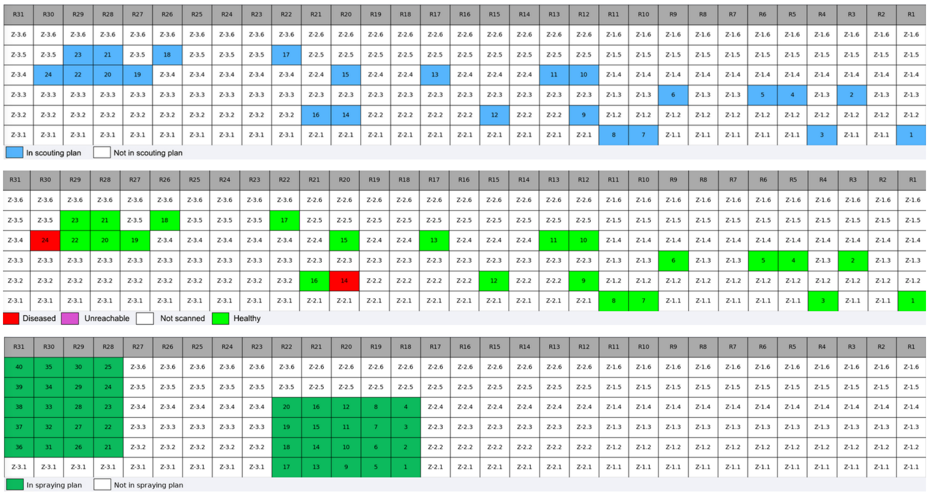
4.3.3.6 Greenpatrol Experiments

This section presents the validation tests performed within the simulated and real greenhouses, of 52x30 m and 31 corridors between plants . The aim of these tests has been the assessment of the following features: 1) the correct integration of Robotframework with the different robotic modules; 2) successful execution of pest inspection and treatment plans; 3) the logging capabilities of the system to generate and use the collected data; and 4) the system requirements proposed in Table 4.3.

Greenhouse simulation tests

Gazebo simulator [202] has been used to simulate the crops, the robot sensory information (laser, images...), the physics involved (collisions, inertia...) and the localization data (global coordinates, errors...). The leaves, despite realistic, do not perfectly represent the real world and do not contain insects on them. Thus, a simulated vision module for leaf detection provides their position. Also, the images used for validating the pest detection and identification modules are semi-randomly acquired from our custom dataset of labelled images (a set not used for training the model), with infected and healthy images. The same software as in the real scenario has been used.

The simulation test presented consists of the following steps. Firstly, an IPM algorithm generates a new semi random scouting plan, which is presented in Figure 4.47 (up). It bases on the greenhouse dimensions, which in this case corresponds to 31 rows (R1-R31) in the horizontal axis and 6 vertical zones that, in turn, consist of a bunch of plants. More precisely, this plan consists of 120 targets to be performed in the zones marked in blue. These targets contain navigation steps together with inspection tasks. While the plan is executed, the log messages are generated and sent to the cloud through the *Logger* module. The results



R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6
Z-3.5	Z-3.5	23	21	Z-3.5	18	Z-3.5	Z-3.5	Z-3.5	17	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5
Z-3.4	24	22	20	19	Z-3.4	Z-3.4	Z-3.4	Z-3.4	Z-3.4	Z-2.4	15	Z-2.4	Z-2.4	13	Z-2.4	Z-2.4	Z-2.4	11	10	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4
Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-1.3	Z-1.3	6	Z-1.3	Z-1.3	5	4	Z-1.3	2	Z-1.3	Z-1.3
Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	16	14	Z-2.2	Z-2.2	Z-2.2	Z-2.2	12	Z-2.2	Z-2.2	9	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2
Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	8	7	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	3	Z-1.1	Z-1.1	1

🟦 In scouting plan ⬜ Not in scouting plan

R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6
Z-3.5	Z-3.5	🟩 23	🟩 21	Z-3.5	🟩 18	Z-3.5	Z-3.5	Z-3.5	🟩 17	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5
Z-3.4	🟥 24	🟩 22	🟩 20	🟩 19	Z-3.4	Z-3.4	Z-3.4	Z-3.4	Z-3.4	Z-2.4	🟩 15	Z-2.4	Z-2.4	🟩 13	Z-2.4	Z-2.4	Z-2.4	🟩 11	🟩 10	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4
Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-1.3	Z-1.3	🟩 6	Z-1.3	Z-1.3	🟩 5	🟩 4	Z-1.3	🟩 2	Z-1.3	Z-1.3
Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	🟩 16	🟥 14	Z-2.2	Z-2.2	Z-2.2	Z-2.2	🟩 12	Z-2.2	Z-2.2	🟩 9	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2
Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	🟩 8	🟩 7	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	🟩 3	Z-1.1	Z-1.1	🟩 1

🟥 Diseased 🟪 Unreachable ⬜ Not scanned 🟩 Healthy

R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
40	35	30	25	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-3.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-2.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6	Z-1.6
39	34	29	24	Z-3.5	Z-3.5	Z-3.5	Z-3.5	Z-3.5	Z-3.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-2.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5	Z-1.5
38	33	28	23	Z-3.4	Z-3.4	Z-3.4	Z-3.4	Z-3.4	20	16	12	8	4	Z-2.4	Z-2.4	Z-2.4	Z-2.4	Z-2.4	Z-2.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4	Z-1.4
37	32	27	22	Z-3.3	Z-3.3	Z-3.3	Z-3.3	Z-3.3	19	15	11	7	3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-2.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3	Z-1.3
36	31	26	21	Z-3.2	Z-3.2	Z-3.2	Z-3.2	Z-3.2	18	14	10	6	2	Z-2.2	Z-2.2	Z-2.2	Z-2.2	Z-2.2	Z-2.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2	Z-1.2
Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	Z-3.1	17	13	9	5	1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-2.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1	Z-1.1

🟩 In spraying plan ⬜ Not in spraying plan

Figure 4.47. Representation of the greenhouse inspected/sprayed zones during the different simulation test-steps: Initial Semirandom Scouting Plan (up); Scouting Results representing the infected and healthy zones (middle); Spraying Plan generated for infected zones (down).

obtained from the analysed data are presented in Figure 4.47 (middle), where the green color represents the healthy zones and the red color the infected ones. From these results, the IPM algorithm can generate a new spraying plan with 80 targets, as seen in Figure 4.47 (down). Similarly, the plan is executed, the results are logged, and the results are analysed again.

The graphs in Figure 4.46 present timing metrics obtained from the execution of these inspection (down-left) and spraying (down-right) plans. The observed navigation time peaks occur when moving from one row to another. Smaller navigation times reflect the movements to plants nearby. We could, for example, capture the exact moment at which the robot failed to reach a destination after aborting the maximum number of trials (*Navigation Trials 3 aborted. Run FailureBehaviour*, as shown in Figure 4.46 up). When a navigation fails, the following task is not performed, resulting on 119 successful inspection and a single task skipped during the scouting plan execution. Also, minimum, maximum, or average times for the different navigation, inspection, or spraying tasks can be easily obtained to analyse the system performance. The total simulation task time is 1h and 40min for the semi-random scouting plan execution, 2h and 24min for image analysis, and 47min for spraying plan execution, resulting on 4h and 51min test.

On the other hand, 1547 pictures were acquired and processed during the test, resulting on a 98.25% of the greenhouse being pest-free (green zones) and a 1.75% being infected (red zones). As the leaves positioning detection and image acquisition modules are simulated, the arm movements are controlled, resulting on all inspection tasks success. Similarly, the spraying manipulation strategy is based on prefixed arm movements, and these movements are therefore always successful. The semi-randomly acquired pictures could be used to partially validate the pest detection and identification module, allowing the IPM module to generate new spraying and treatment plans based on these results.

Finally, it is interesting to remark that logs are available as long as the databases are maintained, allowing to analyse information ex-post. This enhances the traceability capabilities by, for example, allowing to include new required queries not overseen before.

Greenhouse field tests

The objective of the field tests is to ensure the integration of the robot manipulation and perception modules within the architecture in real operational conditions. The robot shown in Figure 4.38 replaces the simulated one and the leaves detection and pest inspection DL models are included. The rest of the software remains the same as in the simulation tests. To check the integration of robot manipulation, inspection, and spraying functionalities two different tests-sets have been performed.

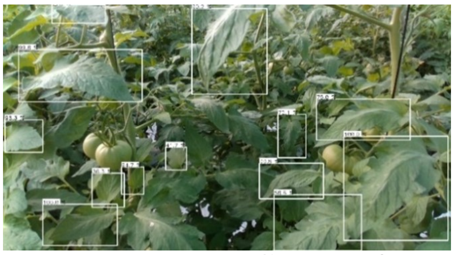
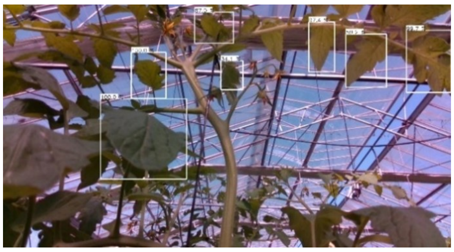
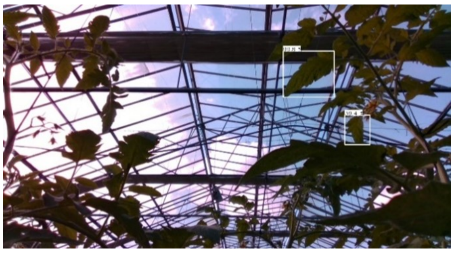
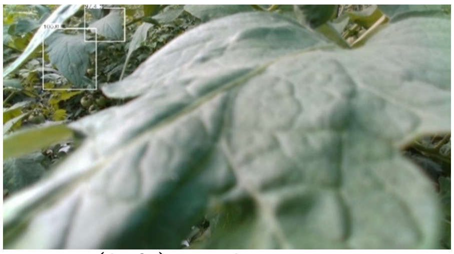
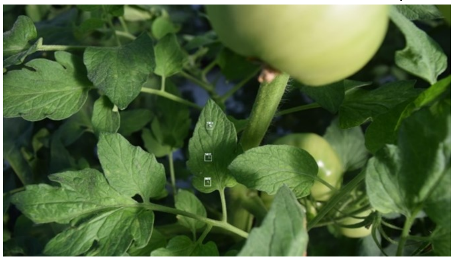
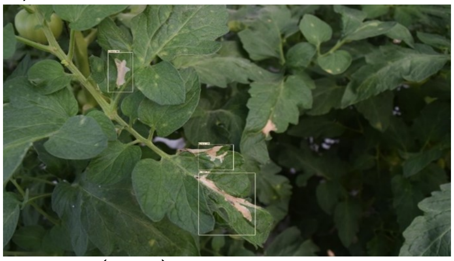
The first tests-set consists of executing simple scouting plans for inspecting the high zone of plants, on the right and on the left sides of the robot. As explained above, the robot navigates to a plant, finds tomato leaves' poses and approaches them to take good-quality and closer pictures. According to the Pest Monitoring Index in Table 4.6, the robot should take 5 pictures in total, 2 from above the leaves (PMI6) and 3 from bellow them (PMI3). The robot is teleoperated to the next plant and the process is repeated 28 times, resulting in the acquisition of 140 pictures.

Table 4.6: Acquired number of images and causes of missed pictures.

Desired number of pictures	140
Missed pictures during Find Leaves step	38
Missed pictures during the Take Closer Picture step	27
Total acquired healthy pictures	64
Total acquired infected pictures	11

The log data is again used to obtain different metrics, similarly as in the previous test. The total valid number of pictures acquired depends on the number of leaves found during the *FindLeavesStep* in Figure 4.44. Most of the times plenty of leaves are found as seen in the Table 4.7 row 1. However, bad illumination or closeness to leaves may cause that not enough leaves are found as seen in Table 4.7 row 2. In that case, the subsequent closer approach to leaf cannot take place. Among the closer acquired pictures, several have not enough quality and are not valid for the pest detection and identification model. This quality is measured through different parameters such as blur, noise, and distortions of different intensities. Finally, the remaining good quality pictures are offline post-processed in order to detect and identify Whitefly insects and Tuta Absoluta damaged areas on the leaves (see Table 4.7 row 3).

Table 4.7: Image acquisition results acquired during the greenhouse field inspection tests.

	
Successfully found leaves.	
	
Illumination changes (left) and closeness (right) challenges.	
	
Successful White Fly (left) and Tuta Absoluta damage (right) detection.	

4.3.3.7 Greenpatrol conclusions

The Greenpatrol use case has extended the Robotframework functionalities, including an automatic transfer of the generated log to the cloud using non relational databases. This information has been used to make queries and generate new plans, based on the results from previous plan executions. All generated historical logs are now constantly available on the cloud to perform system optimisations and promote the continuous improvement.

The greenhouse simulation tests have validated the following key points:

- The generation of pest inspection and treatment plans by the IPM.
- The integration of Robotframework with navigation and manipulation modules.
- The execution of the plans generated by IPM (SR1, SR2 and SR4).
- The proper notification and continuation of the plan when a navigation fails (SR5).
- The manipulation strategies workflow for pest inspection and treatment tasks.

The greenhouse field tests have been used to validate the execution of simple pest inspection and treatment plans on robot's right and left sides, using the real perception modules and spraying equipment (fulfilling, also, SR2, SR3 and SR4).

In both cases the following additional validations have been carried out:

- Use of the GUI for starting the plan, visualizing system status (SR7), and pausing, stopping or restarting the plan on demand.

- Storage of logs in the cloud and subsequent use of the logs to obtain metrics, monitor the correct performance of the system and detect incidents (SR6).

Regarding application specific issues, valuable observations have been obtained from the greenhouse field tests.First, the number of movements and pictures to be acquired depends on the number of leaves found. In the real scenario, this depends on the leaf detector module which has faced the following challenges: closeness to leaves reduces the camera field of view and therefore the number of detected leaves; changes in the illumination also reduces the number of detected leaves. Therefore, the leaf detector DL model should be retrained with closer images, perspectives, and illuminations in greenhouse environment.

Second, it was observed that most pictures analysed during the tests where healthy images despite some more plants where actually infected. It should be possible to increase the probabilities of acquiring infected leaves pictures by including a Single Shot Detector (SSD) real time pest detector model. This model could be combined with the current leaf detector model to approach the most probable leaves with pests first. In addition, some pictures had not enough quality and were not valid for the pest detection model. In the future the quality threshold must be increased.

In summary, greenhouse simulation and field tests have been performed to validate the architecture. It should be mentioned that, although a single simulation test has been presented here, more than 60 hours of simulation have been carried out during the validation of the GreenPatrol project. The field tests have been used to evaluate manipulation, leaves detection and pest identification functionalities in addition to observing limitations of the current robotic platform. The project's youtube channel [239] contains audiovisual material presenting the robot during the simulated and greenhouse validation tests, the user interface and the achievements reached during this almost 3 years project. Robotframework has certainly reduced the time to build the here presented pest detection and treatment application.

4.4 Chapter Summary

The *Robotframework* architecture presents an innovative, efficient and easy to use solution that facilitates the development of robotic applications belonging to different domains. Specifically, *Robotframework* allows the seamless integration of high-level decision systems, which generate the application-dependent plans, with low-level robot abilities, which are related to the robot drivers. The architecture also includes additional logging, monitoring and error handling modules and an intuitive GUI to control the robot and monitor its general status.

The architecture has been designed to be generic and not limited to a concrete application. For this, *Robotframework* is based on ROS, and follows a modular design that together with the meta-model and JSON format, proposed for plans generation, as well as the pluginlibs design concept, used for tasks development, make *Robotframework* easy to be reused in other contexts, without needing to change the main core of the architecture.

The proposal for plans generation assures a unified, generic and technology-agnostic way of defining robotic applications. Additionally, the state machine presented in the Application Layer is common for all applications, but the parametrizable plan generation and execution methods makes it highly adaptable and extensible for new applications. From the abilities point of view, on the one hand, most common global, relative and precise navigation functionalities are already provided by the framework core and can be used on demand. In concrete, the architecture core supports three different types of navigation modes, which can be used as a template, to be adapted or enhanced with newer functionalities. On the other hand, new tasks can be implemented by using the ROS pluginlib concept, without needing to touch the core of the framework.

In fact, three different mobile manipulation tasks have been presented in the form of plugins, which can be again used as template for new applications. In addition, the available framework infrastructure with reusable and tested GUI, logging or application management modules will significantly reduce the time to build up a new robotic prototype with similar requirements.

Initially, *Robotframework* has been built upon the needs and requirements collected from different industrial MMR projects and then validated, first with an industrial aileron inspection use case, and after with a pest inspection and treatment use case, showing its general purpose and re-usability potential.

The previous tests have shown how different tasks for pest detection and treatment can share navigation functionalities thanks to the architectures' modular design based on plugins. During the tests only the first navigation type available in the presented framework has been used being 10 cm accuracy sufficient for a greenhouse navigation solution. Some applications such as a precise drilling or inspection operations, require global navigation to be supported by a more accurate precise navigation. For these cases, *Robotframework* supports the possibility to use relative and/or precise navigation to enhance the manoeuvrability and robot platform's positioning accuracy.

In conclusion, the use of *Robotframework* can significantly reduce the time to build up new MMR applications for other agriculture or industry related tasks, or even any other domain. Figure 4.48 presents a summary of the already integrated applications with *Robotframework* and how the previously introduced applications could be integrated with it.

However, as we have seen in Chapter 2, most RFs have initially been developed thinking on single robot operations, being ROS no exception to it. Most frameworks lack of mechanisms that would enhance the multi-robot socialization capabilities and it is, therefore, difficult to integrate the robots within a flexible manufacturing system. Also, decentralized control and intercommunication with other non-robotics CPPS are challenges that need to be addressed.

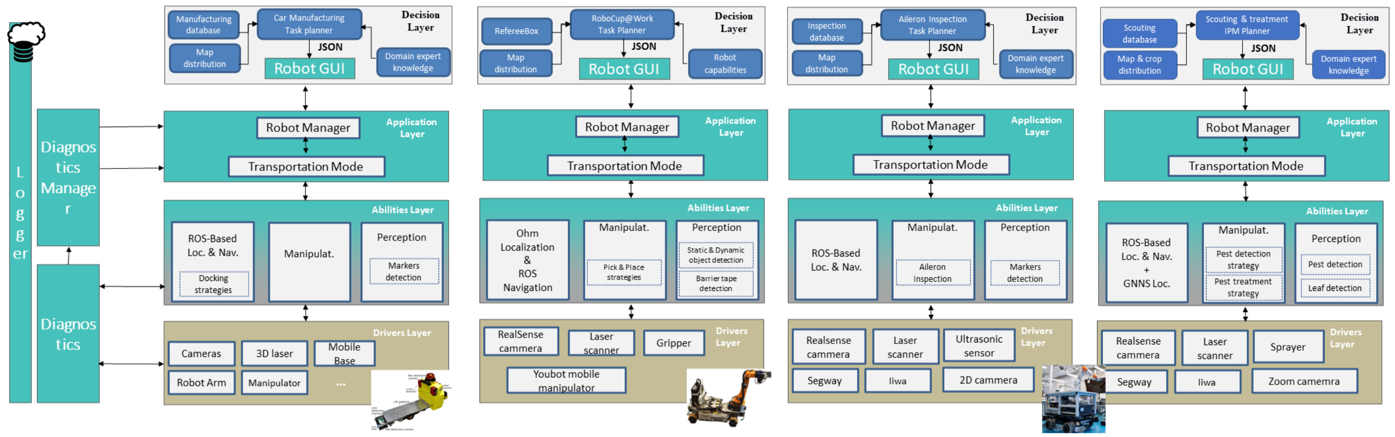


Figure 4.48. Robotframework architecture customized for different applications (from left to right: YouBot, AutoBod, CRO-INSPECT and GreenPatrol). The first two use cases (YouBot and AutoBod) are presented to show code reuse in different MMR applications. The last two use cases (CRO-INSPECT and GreenPatrol) have been used to develop and test the architecture. Modules represented with turquoise color are common to any application. Grey modules are standard ROS modules while the rest are application specific.

5 Integration of the ATV in the Agent based Flexible Manufacturing

While traditional manufacturing processes have been focused on fixed production lines to reduce costs, nowadays manufacturing needs to be more flexible in order to produce customized products with shorter life cycles. Modern manufacturing has become more modular and distributed, transferring the control to self-organized modules that grant flexibility and improve the adaptability to market demanding changes. This limits the top management layer to high-level strategical goals definition and supervision of the factory, rather than the control of the complete top-down system. Each entity in the factory (machine, robot, operator, product, etc.) can be represented as an individual agent with social capabilities, that is able to make decisions based on its local perception and the interaction with other agents. These heterogeneous CPSs communicate with each other collecting and analysing information, increasing their autonomy, decision making, flexibility and efficiency to resolve problems in complex-uncontrolled industrial environments.

In this context, the MMR is one of the key enablers of the factory of the future. As seen in Figure 5.1, MMRs offer flexible transportation services to intelligent machines that manage their own material handling from/to warehouses, enabling the quick and cost adequate reconfigurability of the production and, therefore, increasing the flexibility and scalability of the overall manufacturing system. Figure 5.1 also shows how robots are interconnected with their environment allowing to book free charging stations during the time that are not transporting goods. One of the main benefits is that production orders do not need to follow a determined assembly sequence since the MMRs can perform unplanned or on demand deliveries, e.g., to bring an unfinished product from a broken machine to another one that has some free operation time on its schedule. To command these robots, several RFs such as ROS have appeared to facilitate the development of robotics by offering information ontologies, robotics algorithms, introspection and simulation tools among others. They usually offer modular solutions to common hardware and functional problems which must be combined and extended for different applications. However, the state-of-the-art reflects a lack of social capabilities among their intrinsic characteristics. Thus, it is necessary to develop new generic interaction mechanisms for integrating autonomous and intelligent MMRs within the flexible manufacturing systems.

While Chapter 4 has presented Robotframework, a generic robotic archi-

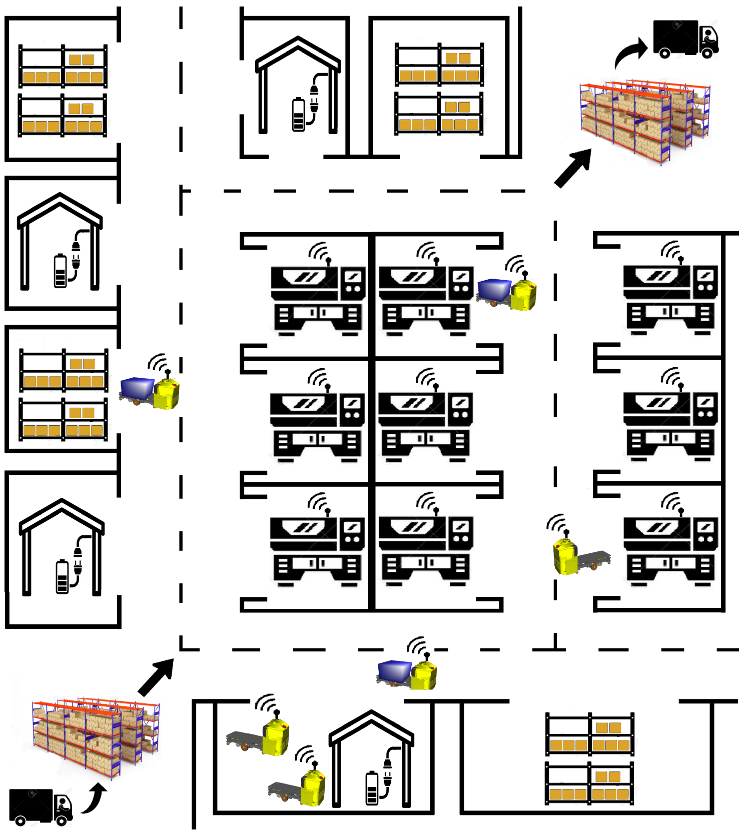


Figure 5.1. The factory layout contains the following components: source and delivery warehouses (bottom left/top right corners); interim storages and charging stations (on the sides); working stations with production machines (center and right side); MMRs spread all over the factory.

tecture that facilitates the integration of single-robot capabilities, this chapter focuses on the problematic of multi-robot decision making, and the decentralized interaction among robots and other non-robotic manufacturing entities. The system needs to benefit from latest standards and technologies to embrace multi-robot problematic such as coordinating several robots, creating intelligent collaboration channels or allowing the robot to interact with other non-robotic industrial systems. The development of these social abilities should benefit the transportation robots as well as any other entity in the production system, enabling the required flexibility, fast reaction to anomalous situations and, finally, leading to the minimum production downtime.

While general requirements related to multi-robot systems have already been referred in the existing literature in Chapter 2, the specific requirements related to the socialization of MMRs with their environment in a flexible manufacturing process are yet to be defined. To that end, a list of more specific requirements (SR) related to the MMR and its socialization abilities have been identified as shown in Table 5.1.

Table 5.1: System Requirements (SR) for the integration of an MMR in a flexible manufacturing process.

System Req.	**Description**
SR1	Offer transportation services.
SR2	Give efficient responses to service requests.
SR3	Notify significant events at a social level.
SR4	Allow reactivity through online service tuning.
SR5	Communicate with any type of CPPS.

With regard to SR1, the MMR services must be published and made available to other entities. Negotiating capabilities are required to reach agreements with other MMRs during task allocation process. SR2 comes from the need of being ready to socialize, while abstracting this duty from low-level, robot-dependent functional tasks. As for SR3, event management mechanisms are needed to notify MMR state changes that may affect other CPSs in the manufacturing environment, e.g., if its battery level is low, the MMR must stop offering services until recharged. Regarding SR4, the MMR functionality must be adaptable to context changes, e.g., it must reduce the maximum navigation speed in presence of human operators or while navigating in restricted areas. Finally, SR5 indicates the MMR must be capable of interacting not only with other MMRs, but also with other heterogeneous, non-robotic CPSs in the factory, e.g., machines demanding transportation services.

This chapter is organized as follow. First, a decentralized architecture for a flexible manufacturing system with focus on the integration of the MMRs is presented. The heterogeneous production entities taking part in the distributed manufacturing control system are identified and the communication mechanisms that enable the interaction among them are defined. Second, the evaluation of the developed social abilities for robots and its integration on the flexible manufacturing is performed through simulation tools and a fleet of low cost Kobuki robots. Here, different communication protocols and frameworks have been integrated to perform the synergy between the MMRs and the heterogeneous manufacturing agents: a pure ROS communication; the de-facto industrial standard OPC UA; and a MAS technology based on JADE framework.

5.1 Decentralized architecture for a Flexible Manufacturing System based on MMRs

This section presents a decentralized control architecture for a flexible manufacturing system with three main layers as seen in Figure 5.2:

- The *strategical layer* contains high level components that offer *Infrastructure Services*, as defined by RAMI 4.0, to support the interaction with and between I4.0 Components.

- The *operational layer* contains resource agents that interact with each other based on their *Application Relevant Services*, as defined by RAMI 4.0, to compose manufacturing applications.
- The *communication layer* is implemented by a middleware that allows the interaction among heterogeneous agents.

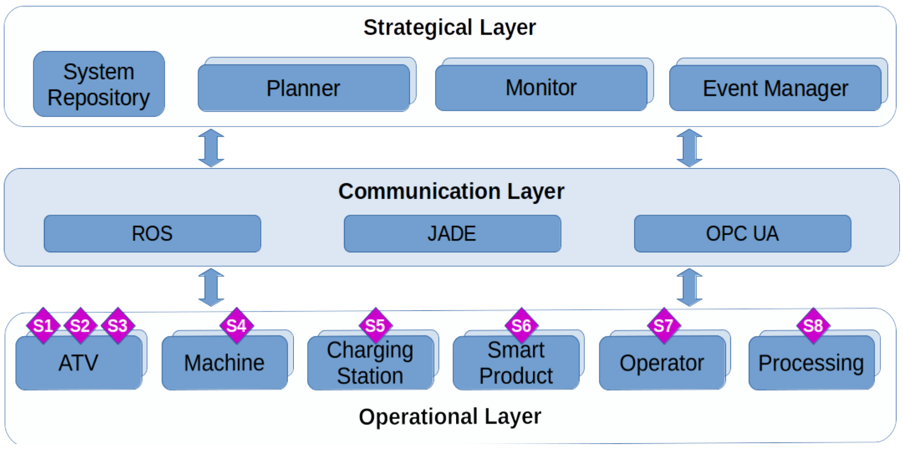


Figure 5.2. Decentralized control architecture for a flexible manufacturing system.

5.1.1 Strategical layer

The strategical layer is responsible for long term operations and monitoring of the overall system. It defines strategical goals based on market requirements or incoming sales orders (e.g. manufacturing of a certain product in predefined quantity). From a technical point of view, this layer provides interoperability between the I4.0 Components in a I4.0 system, manages the operation of the production by tracking registered agents and performed tasks, and monitors events happened during the manufacturing process, triggering the required replenishments.

In order to ensure the required autonomy and modularization, the classic hierarchical control structure is replaced by a hybrid decentralized structure. This narrows this management control layer to a global strategical planning and supervision of the factory, rather than the control of the complete system. This layer is composed of the following agents:

System Repository Agent

It is unique in the system (although it can be replicated). It is responsible for managing the System Repository, which contains essential information about all the I4.0 Components running in the system. Thus, it holds the whole state of the system, keeping track of available resources at run-time, services offered, state of the manufacturing resources, orders being manufactured, etc. The System Repository Agent is responsible for providing the *AAS Infrastructure Services* as defined by RAMI 4.0. it works as a yellow pages directory by updating this information every time a significant change occurs and supplying information to the rest of agents in the system, when required (this functionality corresponds to the *AAS Exposure and Discovery Services* as defined by RAMI 4.0). It also acts as a historical database, saving historical data for learning, forecasting and helps building a more intelligent factory.

Planner Agent

The Planner Agent acts as an interface to the MES, which sets production goals based on incoming sales. These goals are, in turn, transformed into a manufacturing plan which is then transmitted to the respective machines providing all data (e.g. bills of material (BoM), CAD-/CAM-data, etc.) necessary to fulfil these goals. Considering this data, the machines calculate the necessary amount of material. It should be noted that he Planner Agent does not control the organization of the agents in the operational layer, but supervise them and interacts with them only if there is a change in the global plans or if an error event is detected. In these cases, the corresponding replanning is performed.

Monitor Agent

These agents are in charge of continuously collecting general system information, presenting the current state of the factory to operators and generating event alerts when necessary. This information must contain an overall view of the factory with the most valuable information about the agents registered in the system and the activities performed. The operator can trigger specific information queries to obtain the current state of a product, machine or robot. Also, the monitored information could be used by customers to track the progress of their orders. It also saves historical data for learning, forecasting and to help building a more intelligent factory in the future.

Event Manager Agent

It deals with confirming the faults detected by system agents, updating the repository indicating said faults, and managing the error according to what is established in the application. All events must be tracked in historical databases. New events are taken into account to trigger new plan generations, or to notify the agents in the operational layer to take the corresponding actions.

5.1.2 Operational layer

The management of the manufacturing process is usually based on two types of agents: *Product Agents*, which manage manufacturing plans defined as sequences of the manufacturing services they require to be manufactured, whereas *Resource Agents* offer manufacturing services and collaborate which each other to fulfill Product Agent requests. Other types of agents, such as *Order Agents* and *Task Agents* in the case of PROSA and ADACOR architectures, respectively, can be considered if they are necessary to assist the other agents, e.g., to provide coordination or handle scheduling. The simplest way to define agents consists of identifying the basic physical and logical entities present in a manufacturing system and associating agents to them.

Resource Agents usually integrate physical assets and provide *Application Relevant Services*, as defined by RAMI 4.0. These manufacturing resources have been classified in two types:

- *Active agents* such as *Smart Production Machines*, which produce according to specifications from the strategical layers; *MMRs*, which bring the materials from the interim storage areas to the machines and final products to delivery areas; and *Operators*, which offer maintenance services in case of failure or need of reconfiguration to both MMRs and machines.

- *Passive agents* such as *Charging Stations*, which offer their charging services to MMRs; or *Processing Units*, which provide cloud or more likely edge processing capabilities to the active agents.

Based on goals defined in the strategical layer, these individual agents interact with each other to autonomously organize the factory. Since Resource Agents can participate in different manufacturing applications, on demand and while they are available, they are persistent in nature: once created, they remain in the system indefinitely unless there is a problem or they are purposely deleted.

As seen in Table 5.2, agents offer their services in the system which are then contracted by other agents on demand. Thereby, MMRs and machines interact over the communication layer and self-organize the factory according to the global specifications defined in the strategical layer. Moreover, operators have the possibility to supervise and interact with the system in case of failure or hazard via smart devices.

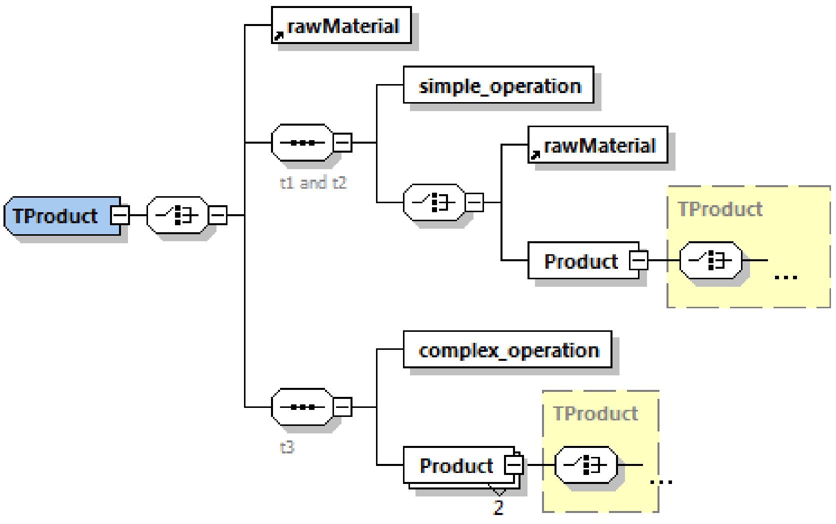
AGENT	**Offered Services**	**Contracted Services**
MMR	(S1) Replenishment (S2) Delivery (S3) CallRobot	Maintenance Charging Cloud/edge
Smart Production Machine	(S4) Manufacture	Maintenance Delivery
iProduct	(S5) Traceability	Delivery Manufacture
Operator	(S6) Maintenance	Call Robot
Charging station	(S7) Charging	Maintenance
Processing	(S8) Cloud/edge	

Table 5.2: Summary of offered and contracted services by different Agents

Product Agent

Intelligent products are software entities responsible for managing, monitoring and tracing all the steps necessary to manufacture a product. Since the intelligent product knows all raw material and manufacturing operations it requires, its agent (iProduct in Table 5.2) can contract MMR services to transport the raw material from the warehouses to an available machine, to perform a required operation (Delivery service). Once loaded on the machine, it informs the machine to start with the operation. When the job is finished, the Product Agent decides if the resulting product needs a new manufacturing operation or if it is ready to be transported to a delivery warehouse.

Figure 5.3. Generic Product Model.



For that purpose, [248] proposes a Product Oriented Manufacturing (POM) solution with interconnected manufacturing resources, which address the manufacture of a product or range of similar products, including the necessary assembly work, simultaneously and in a coordinated manner. The entity *Product* is mapped to an information model that defines its manufacturing steps and its sub-product entities, and makes references to existing machine operations. In general, a manufacture of a product may imply different and parallel lines of raw material over which operations are performed to obtain sub-products that are then combined by compound operations to obtain the final product. Fig 5.3 represents the meta-model of a product type. The product instances, i.e., the manufactured products, can be defined in XML files, as instances of the meta-model, and stored in a product repository.

During the whole process, this agent contracts the services of MMRs and machines, monitors them to react and re-plan in case of failures and saves a track of all steps having a complete view of the process of a determined product. Based on the product model, a generic traceability model could extend the latter to define actual manufacturing information, such as which machine has performed which operation and at what time. It is also possible to obtain information related to the incidences that may have occurred, e.g., if transport was required and the transport time, detection of failures and so on. This information could be queried as a traceability service (S5) and used to inform a client about the exact current status of his product.

This approach allows unitary traceability, but dismisses traceability at other granularity levels. Other traceability entities (and, thus, their corresponding agents) can be defined at different levels, e.g., batch level, customer level (i.e., monitoring all the products or batches that are part of the same customer order) or manufacturing plan level (i.e., monitor the manufacturing plan as a whole).

Transportation Agent (MMR)

The MMR is the main enabler of the flexible manufacturing, offering transportation services (see Table 5.2) which are used by machines to replenish their material reserves (S1), by the Product Agents to transport them from one machine to another and to the delivery supermarkets (S2) and, finally, by operators which may request some kind of material or human-robot capability on their workplaces (S3). This allows a more modular and distributed manufacturing architecture.

During this thesis, the industrial mobile robots in Figure 5.4 have been used, some of which have been presented in Chapter 4 : the YouBot, used during the RoboCup@Work and ERL competitions, presents most of the necessary functionalities of a transportation robot in the industry; the AutoBot, developed as a proprietary ATV for Bosch GmbH, presents the basic navigation and precise docking manoeuvre capabilities for load pickup and delivery; the Segway+Iiwa MMR offers inspection and dexterity capabilities, cooperating with operators on complex and repetitive tasks; and, finally, the Kobuki, a low cost prototype, used in this chapter to demonstrate a multi-robot transportation ecosystem and the cooperation with machines and smart products.

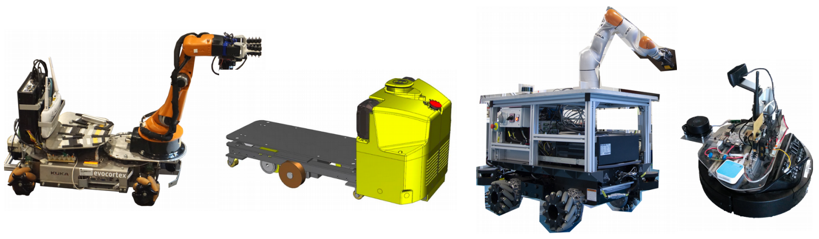


Figure 5.4. Industrial Autonomous Transportation Vehicles. Left to right: Autobod, YouBot, Segway+Iiwa, Kobuki.

Robots are necessarily connected wirelessly to an I4.0 System. They update continually a Robot Heart Beat (RHB) containing a unique identifier, current pose and brief enlightenment about assigned a tasks. When a MMR is plugged into the system, it registers its services in the System Repository and starts informing about its availability and status.

Machine Agent

Machine agents are usually deployed in embedded operating systems that endow, on the one hand, intelligence and the capability to interact with other agents, and, on the other hand, the connection to the asset, which is usually interfaced by a PLC. When the machine requires new material provisions, it sends a request to contract replenishment services of the most suitable MMR. Machines also offer manufacturing services depending on their capabilities (S4) that are then contracted by smart products.

On system start up, machines register their services, location and other important data in the System Repository. This way, new machines can be added or updated respect to the services they offer. Their operation plan can be provided by the Planner Agent (once received, they execute it until they finish, while taking care of their internal material replenishment), or it can be triggered on demand by the Product Agent.

Machines can interact with other agents either wired or wirelessly. They publish Machine Heart Beat (MHB), a brief information about its internal status that contains its identifier, status, identifiers of robots contracted for the transportation, material type required and deadline for the task to be finished.

Operator Agent

The operator performs value added tasks in the manufacturing process, and collaborates with robots in his surroundings using a Human-Robot-Interaction Panel which can run on a tablet-like device. This device provides information about the area in which the operator is working and enables the interaction with nearby MMRs, contracting their services for inspection or hazard tasks. This interaction includes maintenance services (S6) which can be used to trigger modification on robots or machine behaviours, in case of failures.

Charging Station Agent

Because the transportation is carried out by MMRs, it is necessary to have an area prepared to charge their batteries. The robots will charge their batteries during their idle times while waiting to be assigned new tasks. However, in a factory with hundreds of them, it is not viable to have one pre-assigned charging station to each robot. It is more likely to have a pool of charging stations shared by all the robots. For that purpose, the management of these smart charging spaces is performed by software agents that control the slots available and offers them as charging services (S7) to MMRs.

Processing Agent

MMRs are restricted to limited battery and computational power. Increasing their computational power would limit their working autonomy time and the amount of extra load they are able to transport. In general, MMRs need to perform a series of real time calculations for security and efficiency. However, there are other tasks that may require a remarkable computing effort and that do not imply real time processing necessities. It is therefore interesting to include smart processing units that offer computational services (S8) to MMRs on the edge or on the cloud, depending on the privacy of the information exchanged. In addition, software agents such as the Charging Station or Smart Product agents require a place to be executed. Processing Agents represent the processing nodes where those agents could be deployed, providing access to the computing capacity of the system. Processing Agents can negotiate with each other, based on different criteria (e.g., available memory, CPU load, etc.), to determine which one host agents.

5.1.3 Communication layer

In the future, hundreds of interconnected robots, machines, operators and other heterogeneous agents are supposed to collaborate to build a flexible manufacturing system. It is therefore important to create an intelligent communication network that enables the understanding among them without overloading the network. Some information must some-times be directly sent to an specific entity while some other information needs to be shared with several entities. This section presents some of the most common communication middlewares used on robotics.

- ROS has become the standard robotic communication middleware, due to its usability and popularity. Most robotic vendors create drivers for ROS and ROS-Industrial ecosystem.
- OPC UA, the default agnostic PLC communication standard, is used to communicate low-level manufacturing plant floor and upper MES systems to perform configuration changes on machines.
- MAS has proved over the last decade to be an excellent solution to distribute the control on decentralized manufacturing systems. This work focuses on the integration of autonomous multi-robot transportation systems based on ROS into MAS.

ROS as communication middleware

Being the de facto standard for robotics, ROS is built as a set of comprehensive and well-structured libraries and drivers for several robots and sensor devices, and a well defined structure for communication. ROS provides a message passing interface with three information exchange possibilities as explained in Section 3.2: the publisher-subscriber, the service-client and the actions.

- The publisher-subscriber semantic allows the diffusion of information to any other robotic component listening to a published topic. Every agent interested in a topic can subscribe to it and wait for new messages to arrive. There is no response implicit in this type of communication and, therefore, the sender does not know who received the information. This communication option will be used, for example, by the machines and robots to publish their Heart Beat on the cloud. Even though this Heart Beat is limited to a small quantity of bytes, the publication rate will be limited to the smallest required in order not to saturate the network.

- The service-client semantic permits the direct communication between agents. The sender contacts a determined agent and receives back a response from it. The sender knows if the message has been received or not. No other agent will notice this information exchange. A machine can thereby create a point to point communication with a desired MMR to receive a material supply.

- The actions are a combination of the previous two mechanisms. It is used after a negotiation, once it is decided which agent is going to perform the service. First, a service like peer-to-peer communication takes place where the contracting agent sends a goal to the service performing agent. The service agent responses with an status message, confirming the acceptance of the task. Second, the contracted agent sends a continuous feedback message to the contracting agent informing about its current status. Finally, once the service task is completed, a final Result message is sent.

Normally, in the ROS ecosystem, a single and centralized *roscore* master manages the registration of all agents, services, topics and parameter servers as well as the communications between all the ROS nodes. To allow the decentralization, there are several multi-master packages, e.g. multimaster_fkie [2], that allow every agent to be its own master and exchange information between nodes without the need of a central server.

In this work, ROS acts as the main robotic middleware, providing the latest state of the art robotic algorithms and tools. The limitations of ROS as a multi-robot middleware are studied and wrappers are built on top of ROS to integrate the robots into a MAS.

ROS-OPCUA as communication middleware

In today's factories the PLC is one the most common used device to implement a low-level intelligence in production machines, controlling the manufacturing actions performed by a machine and monitoring their internal status such as the filling quantity of a depot. To avoid the limitations of working with different vendor PLCs, OPC UA has become the de-facto standard middleware for controlling them. To connect the PLCs controlling the machines and the robots, this work uses the FreeOpcUa library, which integrates all necessary the mechanisms to enable the communication between robotic components and machines. As explained in Section 3.4, OPC UA is composed of multiple clients and servers exchanging broadcasted on peer-to-peer information.

In this work, the OPC UA-server is used to track the information of virtual machines and trigger the corresponding service requests to robots for material replenishment and delivery. The machines (OPC UA) and robots (ROS) are represented as OPC UA-clients that communicate with each other through a custom communication server based on the *ros_opcua_communication* package [210].

ROS-JADE as communication middleware

JADE is a generic software platform that simplifies the implementation of distributed applications based on MAS technology. As seen in Section 3.3, it can be considered as one of the most popular multi-agent middleware as it offers the basic features expected in a middleware of this type and some added features that make its use, debugging and maintenance very easy. Among these added features, JADE allows creating graphical interfaces with the intention of seeing the operation of a specific agent, the messages that are exchanged between the agents, sending messages in a simple and graphical way or even creating an operator agent that interacts with the rest of agents from buttons. It also provides ready to use mechanisms such as the Agent Management System (AMS), the Directory Facilitator (DF) or the Agent Communication Channel (ACC) which are inline with the infrastructure services defined in the RAMI specifications.

In this work, JADE has been used as a wrapper on every agent to permit a distributed communication of heterogeneous agents in a flexible manufacturing system while endowing from the intrinsic mechanism of MAS systems.

5.2 Integration of the ATV on the flexible manufacturing system

This section presents the simulated and real scenario use cases where MMRs cooperate with each other and with other smart agents in a flexible manufacturing system. To perform the flexible transportation efficiently, these agents require a series of common and well defined mechanisms to allow integration and collaboration among agents. The validation has been carried out progressively in three complementary stages:

- First, a pure ROS and mainly simulated approach has been used to define the foundations of the distributed multi-agent architecture and the multi-robot task allocation adoption.
- Second, the previously defined mechanisms have been used and enhanced with the possibility to communicate robots using ROS and machines using OPC UA. This use case has been validated with a series of MMRs represented by Kobuki robots and machines represented by Raspberry Pi boards using CoDeSys IDE.
- Finally, MMRs have been integrated on a MAS-based architecture. This is presented as a ROS-JADE integration that includes the de-

velopment of a series of mechanisms for efficiently communicating robots and other non robotic systems. In this stage, the concepts of intelligent products and charging stations have been added.

5.2.1 Integration of MMRs and machines based on a pure ROS simulated use case

This use case recreates the smart factory scenario presented in Figure 5.2 into the ROS environment, where interconnected MMRs, machines and operators organize the transportation logistics workflow of a factory. To manage strategical order generation, start the manufacturing process and track the agents actions, a monitoring panel was been developed. This simulated scenario is used to validate the monitoring panel, the multi-agent task allocation process and the general information exchange such as the publishing of heartbeats. The results of the development were released in a github repository [249].

The focus of the simulated scenario is on the MMRs and the Replenishment (S1), Delivery (S2) and CallRobot (S3) services presented in Table 5.2. Agents demanding these services need to define specific information on the request as seen in Table 5.3. Every service request requires a header, containing a time stamp, and an unique request identifier, the identifier of the agent doing the request, the task type and a deadline for the task to be completed. Additionally, depending on the task type, there is a series of meta-data that need to be included in the message:

- *Replenishment* service is used by machines that want to restock their intrinsic raw material supplies, such as nails or screws. This specific order must contain the delivery location (to know where the material should be brought), the material type (used by MMRs to know from which source the material should be picked-up), and, finally, the amount or weight (to ensure that the contracted robot is able to carry such amount of material).

- *Delivery* service is used to bring a by-product from one machine to another and to dispatch warehouses. In this case, the pick-up and delivery locations and the weight of the load are required.

- *Call robot* service is used to take a robot to an specific location. This service could be useful for operators that need semi-autonomous robot collaboration where the specific tasks are defined once the robot is there. In this case, the destination location and the robot type are required. Robots bid for the specific tasks and compete with other robots to acquire it.

Replenishment services could also be contracted by Product Agents which need to bring raw material to machines. Product Agents could also be in charge of controlling the deliveries of products between machines or to warehouses. However, these cases are not covered in this section.

Similarly, operators offer *maintenance* services to robots, machines and charging stations, despite in this first step the failure and maintenance of MMRs have only been considered. The monitoring agent will detect the failure based on the RHB information or based on communication loss with the robot, triggering a failure event to the corresponding operator that has to perform the maintenance service.

Machines follow a predefined production plan that could be manually generated or autommatically transferred from a MES. To execute the plan, machines negotiate with available MMRs, contracting their replenishment services to refill their raw material supplies and delivery services to send finished products to delivery warehouses. To that end, machines

Table 5.3: Different services structure based on pure ROS messages.

service_type	**ROS_msg**	***Metadata***
Replenishment	header, id_sender, service_type, deadline, **metadata*	material_type, delivery_location, quantity
Delivery		pickup_location, delivery_location, quantity
Call Robot		destination_location, robot_type
Maintenance		destination_location, failure_type

are endowed with two types of timers: the first timer triggers the empty raw material supply event and its corresponding material replenishment request, whereas the second timer triggers the manufacturing product completeness and its corresponding delivery service request.

To perform the efficient assignment of a set of tasks to a set of agents, a multi-agent task allocation problem must be solved. The decentralized version of the "Single-Task Robots, Single-Robot Tasks, Instantaneous Assignment (ST-SR-IA)" classification proposed by [185] was used for that purpose. The ST-SR-IA approach focuses on tasks that can be achieved by single robot, robots that cannot perform simultaneously more than one task, and instantaneous assignments of the tasks to the most suitable robot after performing a bidding process.

For the implementation of the task allocation process, a decentralized bidding approach based on the Contract Net Protocol (CNP) [187] has been used. This multi-agent task-sharing protocol is divided into four stages: 1) task announcement by an agent interested on contracting specific services and which takes the role of the coordinator, 2) bid submission by individual agents able to perform the specific task, 3) evaluation and winner selection, 4) and, finally, the contract stage of the winning agent. Thus, the machine fulfils the role of the coordinator, whereas MMRs participate as bidders. However, in order to increase the robustness, besides assigning a primary robot to perform the task, an additional secondary robot will be assigned. It will supervise the primary robot and assume its task in case of failure. Moreover, the auction process is run locally between machines and nearby MMRs. This simplifies the approach, reducing the information exchanged and allowing the scalability.

5.2.1.1 Simulated scenario setup

Some information is shared between several entities at the same time (a machine publishing the request for a transportation service) while some other information is directly exchanged between two individual collaborating agents (once the service is contracted, per-to-per communication between machine and robot is established). ROS provides a message passing interface with two information exchange possibilities for these cases: the *service-client* and the *publisher-subscriber* respectively. When the machine requires a new replenishment or delivery service, it *publishes* a new transportation task on the middleware. The available MMRs *subscribe* to the task over the communication server. They calculate their own specific bidding costs for fulfilling the transportation, and send them through a *service-client* call to the communication server. These costs consist of the path between the current robot position and the target position. More parameters could be included to calculate the costs, e.g. the robot's battery status, or the suitability of a defined type of robot to perform a task. After a certain bidding period, the machine shorts the bids and contracts the services of a primary and a secondary robot based on the received cost. The function of the secondary robot is to backup the primary robot, endowing a bigger robustness to the system. In case the primary robot has a problem, the secondary robot will overtake its task. Once the task is assigned, the MMR navigates to the closest material storage area, performs an autonomous loading manoeuvre [111] (here simulated), and transports the material to the machine. Once the task is finished, the robot returns to a charging station and starts subscribing again for new transportation tasks.

For the simulation, the material is supposed to be automatically distributed from the warehouse to the interim storage areas close to the machines. Fig 5.5 represents in a sequence diagram the course of events between machines and MMRs for the multi-robot task allocation. When a new MMR is introduced in the system, it first requests necessary information to the middleware, such as the newest factory map, positions of current machines and charging stations, and localizes itself in the factory. Once the initialization state is complete, the MMR *subscribes* to the middleware and wait for new transportation tasks. When the machine reaches a specified minimum fill level of raw parts, it *publishes* a new replenishment task on the cloud. Then, available MMRs receive the new task directly from the machine and calculate their own specific costs for fulfilling it. These costs consist of the path (MMR-material source-machine), the robot's battery status, and their suitability for the respective type of task (in case diverse MMRs with different abilities are available). Once the cost has been calculated, the robots check whether they can complete the task within the deadline and, if so, send the cost directly to the machine. After a certain bidding period, the machine shorts the bids and tries to contract the services of the MMR with the lowest cost. It is important for the machine to know if the elected robot

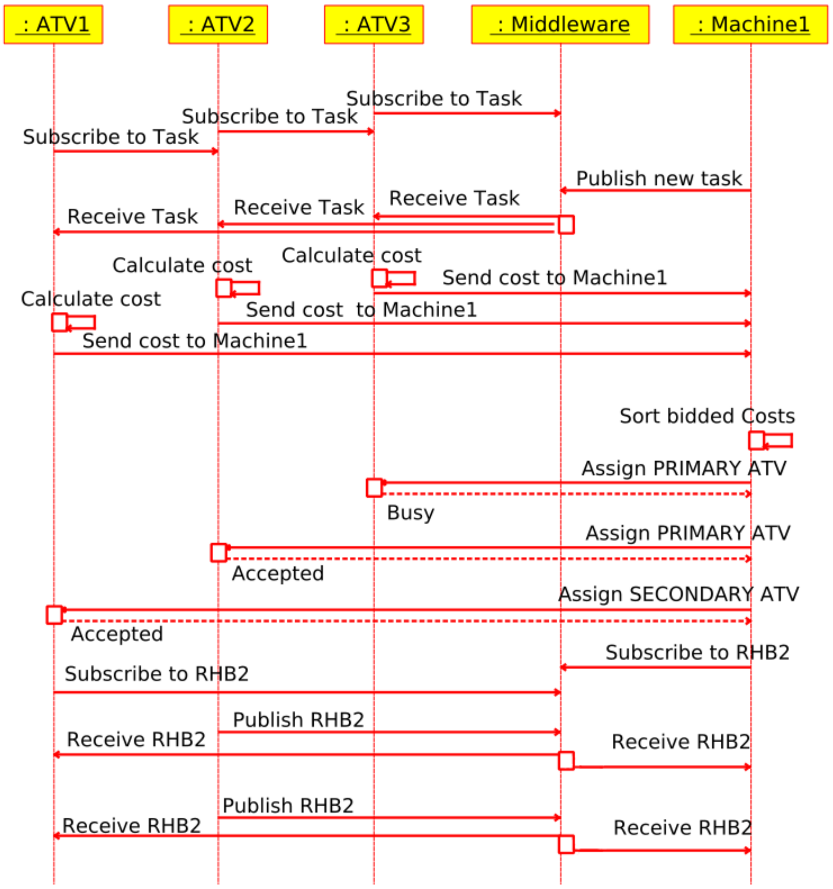


Figure 5.5. Task allocation sequence diagram for an autonomous ATV-machine material replenishment.

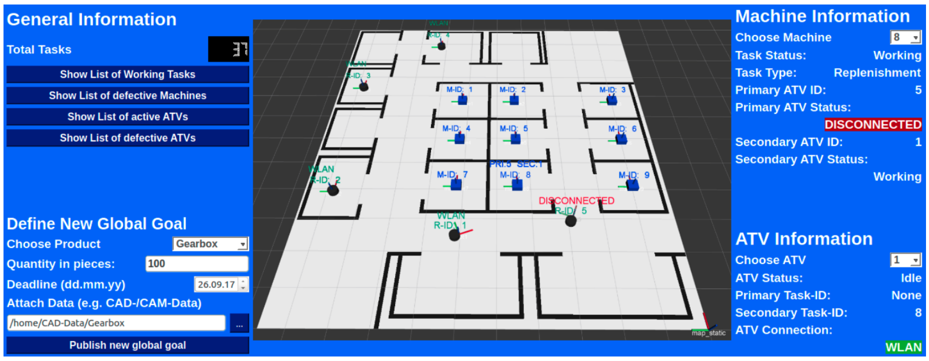
receives the request and if it is still available.

As shown in Figure 5.5, the ATV3 was the robot with the lowest cost, but it was already elected by a different machine by the time the bidding time finished. In case the MMR is still available, it becomes the primary robot for the task. If not, the machine contacts the next robot on the list. As ATV2 had the second lowest cost and was still available, it became the primary robot for the task. To increase the robustness of the system, the machine selected a secondary robot by following the same process (in this case ATV1). Once the task is assigned, the MMR navigates to the closest material storage area, performs an autonomous loading maneuver [111] (here simulated), and transports the material to the machine. After fulfilling the task, the robot starts subscribing again for new transportation tasks. On the bottom of the diagram it can be appreciated how the elected primary robot ATV2 updates its unique RHB2 introduced in the subsection 4.2.5. The heartbeat allows the interested agents, in this case the machine and the secondary ATV3, to receive a status of the primary robot.

5.2.1.2 Simulated scenario evaluation

Two different control panel prototypes have been developed for the evaluation of the system: 1) a management panel responsible for the strategical order generation and monitoring the entire systems, and 2) a human-robot interaction panel that displays information of interest to the operator and allows him to interact with the MMRs in case of hazard or failure.

The management panel starts the manufacturing process and monitors the tasks performed by the different MMRs and machines as shown in Figure 5.6. Qt has been used for the creation of the graphical interface as it is platform-independent, open source, easy to use and has a comprehensive documentation. The Management Panel is mainly used for surveillance purposes. In the center, it presents a real time factory layout with the machines represented as rectangular blue shapes and the MMRs as circular black shapes. The top left corner displays general information about the factory on demand. The top right corner displays information of a chosen machine. The bottom right corner displays information



General Information

Total Tasks 37

Define New Global Goal

Choose Product: Gearbox
Quantity in pieces: 100
Deadline (dd.mm.yy): 26.09.17
Attach Data (e.g. CAD-/CAM-Data): /home/CAD-Data/Gearbox

Machine Information

Choose Machine: 8
Task Status: Working
Task Type: Replenishment
Primary ATV ID: 5
Primary ATV Status: DISCONNECTED
Secondary ATV ID: 1
Secondary ATV Status: Working

ATV Information

Choose ATV: 1
ATV Status: Idle
Primary Task-ID: None
Secondary Task-ID: 8
ATV Connection: WLAN

Figure 5.6. Management Panel used for surveillance purposes by visualizing the plant in ROS (*center*) and displaying several information about the plant (top left), the machines (top right) and the ATVs (bottom right). Additionally, it is used to define and publish global production targets (bottom left).

about the ATVs. It is therefore intuitive to see the amount of tasks that single robots or machines have performed, their connectivity status thanks to the heart beats that both robots and machines are publishing, or their current working status (working, pending or idle). Finally, the user is given the opportunity to define and publish the new products to be manufactured in the bottom left corner. This option should be replaced in the future by an automatic module that reads the incoming orders and publishes the respective answers in the system.

The Human-Robot-Interaction Panel (see Figure 5.7) runs on a tablet-like device, provides the operators with information about the working station they are at, and enables the interaction with nearby MMRs. This interaction includes modification of MMR behavior in case of failures.

In order to evaluate the integration architecture of MMRs and machines based on pure ROS, several cases and the reaction of the system to them

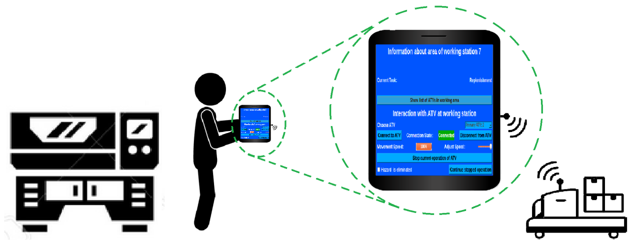

Information about area of working station 7
Current Task:
Replenishment
Show list of ATVs in working area
Interaction with ATV at working station
Choose ATV
Primary ATV: 2
Connect to ATV
Connection State:
Connected
Disconnect from ATV
Movement Speed:
100%
Adjust Speed:
Stop current operation of ATV
Hazard is eliminated
Continue stopped operation

Figure 5.7. Operator at the machine connects to and interacts with an ATV via hand-held device.

are presented below.

Case 1: Replenishment task requested by a machine

A use case was conducted in which A machine publishes a new material replenishment task. The five available robots send their bidding costs to the machine and the machine selects the two MMRs with the lowest costs once the bidding period finishes. The selected primary robot condusts the task until it is finished and returns to it original position. In the future, charging stations could be available and a new bidding process to contract the services of a free charging station would be required.

Case 2: Failure on a determined machine

In this case, an operator intervenes and solves the fault by, e.g., updating the machines software or repairing a hardware component. If possible, the machines orders are redistributed between the other machines in the factory. If not possbile, only its orders will be affected while the rest of the production system proceeds as usual.

Case 3: Peak workload season

In order to react to peak workload seasons it is sufficient to add more agents to the system. The agents that are already on the system will not be affected as every agent makes its own decisions.

Case 4: Network connection problems

A network problem can prevent the MMR from bidding for new tasks. To solve this issue, enduring bidding periods are permitted. This extends the time from the task publication to the contract of the MMR. However, it is enough to consider it in the replenishment time estimation and publish the order on ahead. In addition, a network problem can prevent the MMR from publishing its RHB. Assuming that there is a failure on the primary MMR, if this occurs during a transportation task, the system could react and start a new duplicated transportation task. To avoid this problem, an additional ad-hoc network could be implemented on the MMRs. Thus, if an MMR has a connection problem (e.g., it is too far from an access point, or the connection suffers from huge latencies), it will still have the possibility to communicate its status to nearby MMRs that would transmit the information into the cloud as shown in [250].

Case 5: A busy MMR has issues to fulfil the task

An unavoidable obstacle in the way or an important hardware issue may make it impossible for the MMR to reach its destination. In this case, the problem will be evaluated first. If there is the possibility to solve it within a reasonable time, an operator may be requested to intervene. Otherwise, the secondary MMR could assume the task, or the machine could publish another task request starting a new bidding process. This new bidding immensely increases the robustness of the system, but the task time restrictions might not be achieved.

This problem is represented by Figure 5.8 where the Machine 8 publishes a new task and selects ATV3 as primary robot for being the closest one to the machine and the ATV1 as the secondary robot for being the second closest one. To simulate the failure on the MMR, the primary robot (ATV3) is forced to be switched off. Then, the ATV1 notices that the primary robot is no longer in the system. Thus, ATV1 assumes the task and executes it by itself. Figure 5.8 (right) shows the ATV3 as disconnected and the secondary ATV1 starting to perform the task. The operators notice the disconnection and remove the ATV3 from the factory to repair it.

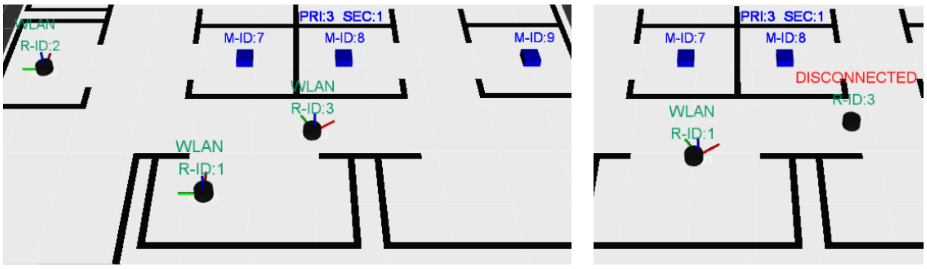


Figure 5.8. Experiments. Machine 8 publishes a new task and selects ATV3 as primary and ATV1 as secondary robots (left); Forced failure in ATV3 lets it as disconnected while ATV1 overtakes the task (right).

Case 6: Autonomous loading manoeuvre fails

The material box might get shifted during loading manoeuvres, which could cause damage on the environment, the MMR itself, or in worst case the operators. Therefore, operators must be able to interact with robots by adjusting their speed or completely stopping their current activity. After solving the problem, the operator can confirm the elimination of the fault and allow the MMR to continue its normal operation.

Case 7: Software updates

Factory upgrades usually require to stop a partial area or even the complete factory for a long time. Moreover, it may be difficult to determine wether the new software contains failures until it is deployed and tested. Modularized agents can help to scale up the factory upgrade. It is not necessary to stop the whole factory, as single agents can be introduced and tested bit by bit before the whole system is upgraded.

5.2.1.3 Summary

In this subsection a simulated scenario for the integration architecture of MMRs and machines based on a pure ROS solution has been presented. The proposed MRTA was kept simple and local. When a machine offers a new task, a distance parameter drives the selection of nearby MMRs to enter in the bidding process. If the machine does not obtain any answer from nearby MMRs during the bidding period, the parameter value is increased so more distant MMRs can enter the bidding. Each MMR estimates its own cost and sends it to the machine, avoiding complex algorithms to run in any agent.

Although this solution allows decentralizing the control of each robot, it continues to present problems in multi-robot applications, due to the general philosophy of the platform. On the one hand, the synchronization of services carried out by master_sync is through pull message pattern, sending unnecessary messages and not updating the information in real time. On the other hand, the master_discovery node is continuously sending multicast messages in search of new Masters, without having any certainty of the existence of the new one. In addition, as ROS is made up of different and very small processing units that exchange information by topic, when communicating nodes from a global network, it is possible that there may be delays and even information loss on some occasions.

This scenario could be improved by extending the experiments in the replicated factory layout in order to observe the influence of networking problems for longer periods, and further research the scalability of the solution in complex situations with a high number of machines, robots and simultaneously performed tasks.

5.2.2 Integration of MMRs and machines based on a ROS-OPC UA real scenario use case

The PLC is the most common used device to implement a low-level intelligence in production machines. The PLCs control the machines and monitor internal status such as the filling quantity of a depot. This section presents a distributed control architecture for a flexible and robust transportation system with MMRs using ROS, that offer transportation services, and PLCs using the OPC UA industrial standard, that contract their services for material replenishment or product deliveries.

ROS was not initially designed as a multi-robot system and, therefore, the standalone ROS configuration must be analysed before building a ROS-OPC UA integration. In this sense, the general objective of this section is to answer the following questions:

- Is it necessary to distribute the ROS master?
- How does the multi-master architecture affect the network and overall system performance?
- How can a stable ROS multi-agent configuration be built?

To answer these questions, three ROS configurations were assessed for the proposed multi-agent system: 1) the standard single master configuration in which a single-master handles the entire registration and XML data requests; 2) a multi-master configuration in which every topics/service from remote masters is synchronized to the local master; and 3) a filtered multi-master configuration in which each agent is specially parametrized, to just synchronize required information and therefore reduce the network bandwidth.

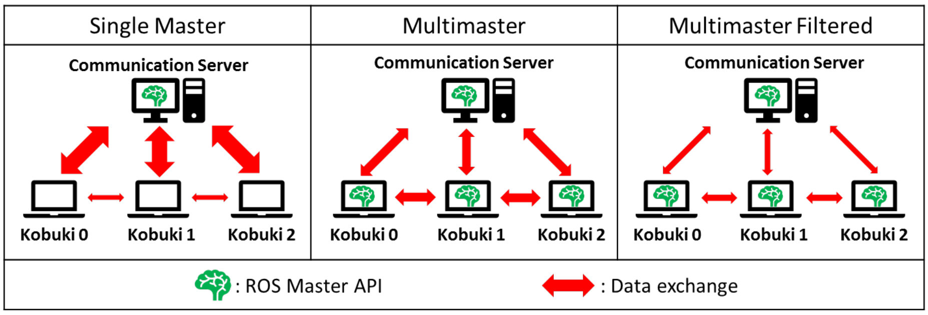


Figure 5.9. ROS single-master, multi-master and filtered multi-master configurations with data exchange representation.

To find the benefits and limitations of each configuration, a test-factory scenario was recreated and several tests were performed to measure three indicators: 1) the network bandwidth, 2) the system performance over the response time of a single ROS service call, and 3) the scalability, by increasing the number of robots in the system and measuring their bidding response time. After analyzing the advantages and disadvantages of each configuration, the most suitable one was selected for a long working period test, where machines and MMRs interact autonomously performing material replenishments and products deliveries.

To apply the multi-master configuration in ROS, the *multimaster_fkie* package [2] was used. It contains a *master discovery* node, that finds remote masters and a *synchronization node* that connects to the discovered masters, requests their actual ROS state and registers the remote topics and services locally. It is important to note that the synchronization node must be correctly parameterized so as not to synchronize unnecessary remote topics that could overload the network.

5.2.2.1 Real scenario setup

The tests were performed in the small factory scenario depicted in Fig. 5.10. MMRs were represented by six Kobukis (see Figure 5.11), equipped with an ODROID-XU4 single-board computer to endow intelligence, a RP-LIDAR laser scanner for localization and mapping, and a TP-LINK router that provides WLAN connection. Machines are represented in the middle by two PLCs running on a Raspberry Pi 3 with CODESYS [251] interface for control and visualization purposes, as seen in Figure 5.12. Each machine was running an OPC UA server for external interaction.

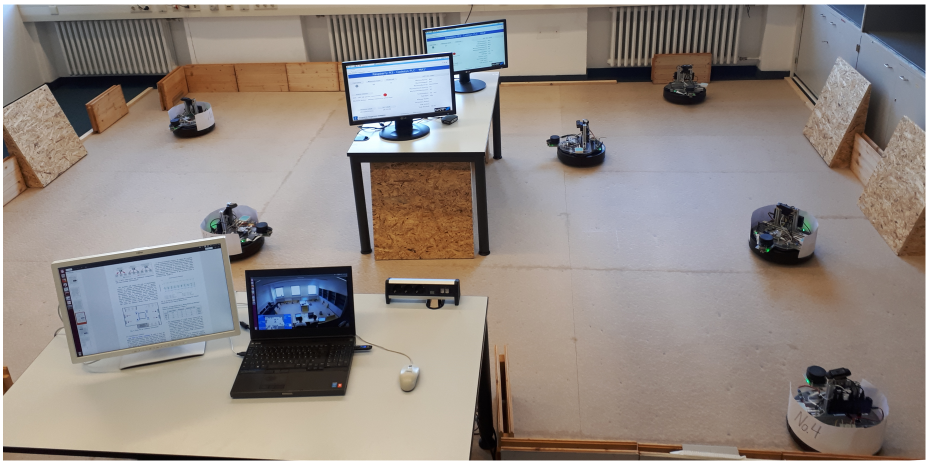


Figure 5.10. Test factory scenario at the Technische Hochschule Nürnberg with up to six ATVs, two machines and a monitoring station.

To enable the communication between OPC UA servers and MMRs using ROS, a *communication server* was developed. This server was designed to act as an interpreter to keep the intelligence distributed. The PLC and the communication server were connected to each other via OPC UA. The communication server subscribes to each PLC to update its internal

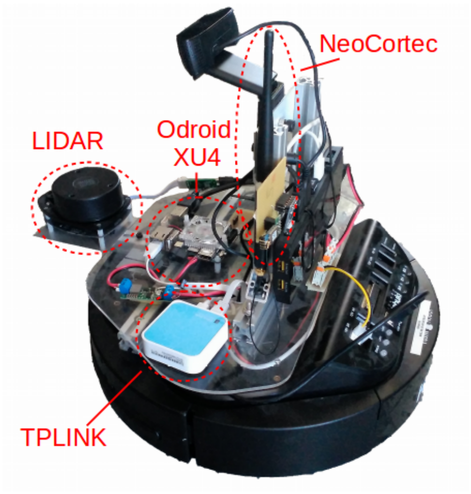
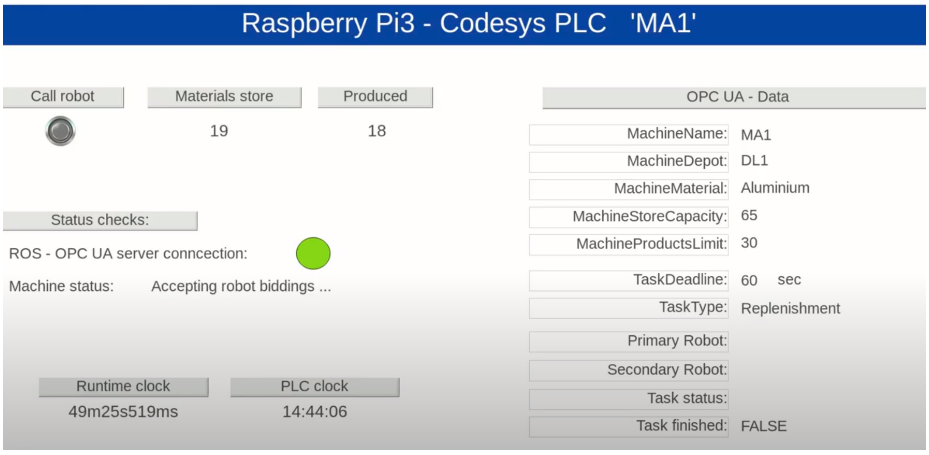


Figure 5.11. Kobuki platform and that represents the ATV for validation tests.



Raspberry Pi3 - Codesys PLC 'MA1'

Call robot	Materials store	Produced
●	19	18

Status checks:

ROS - OPC UA server conncection: 🟢

Machine status: Accepting robot biddings ...

Runtime clock	PLC clock
49m25s519ms	14:44:06

OPC UA - Data

MachineName:	MA1
MachineDepot:	DL1
MachineMaterial:	Aluminium
MachineStoreCapacity:	65
MachineProductsLimit:	30
TaskDeadline:	60 sec
TaskType:	Replenishment
Primary Robot:	
Secondary Robot:	
Task status:	
Task finished:	FALSE

Figure 5.12. Web Server Visualization of the PLC controlled machine1 simulated in codesys.

data and uses this information to trigger the interaction with MMRs.

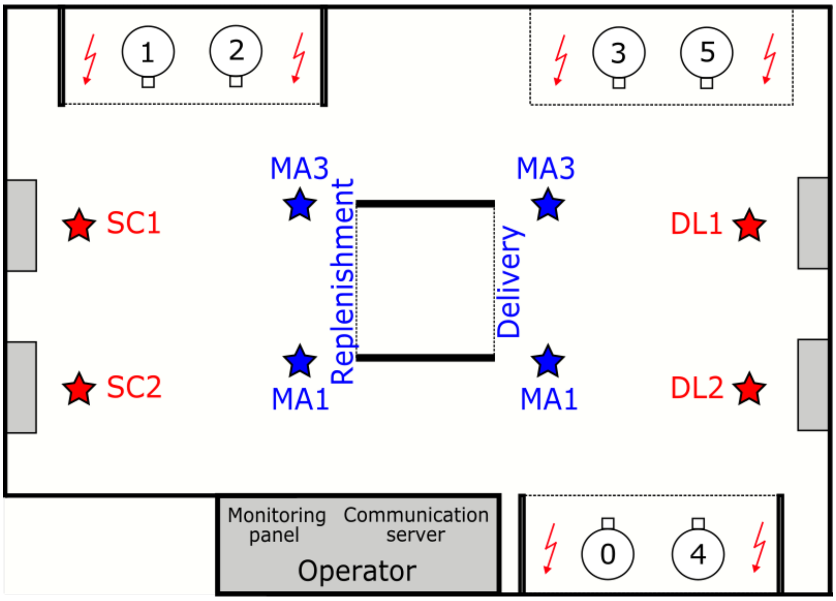


Figure 5.13. Setup of the test arena with MMRs 0 to 5 in the charging stations, machines (MA1 and MA3) in the middle, communication server and monitoring panel in the operator area, and source (SC) and delivery (DL) storage areas on the sides.

The machines required two types of transportation services: a material replenishment and a product delivery. To automate the task publications, two different counters were implemented into the machines: a decremental counter that represents the material left in the storage, and an incremental counter representing the produced products that have to be delivered. To prevent inoperative times in the machines, a replenishment task was published before the first counter reached the value zero. The second counter triggered a delivery task prior to the overloading of the machine's depot.

5.2.2.2 Real scenario evaluation

In order to asses the different ROS-based configurations, the following tests were performed:

- First, the agents were connected over a wired network to measure the overall system network bandwidth. For the remaining tests the agents were connected to another over the wireless network as in a real use case.
- Second, the time response for a ROS service call was measured to evaluate reaction times on each configuration.
- Third, the time response to a single task request with respect to the number of available MMRs was measured in order to analyse the scalability.
- Fourth, the results were summarized and a configuration was selected for an automatic run as described in 5.2.2.4, where the system works non-stop for 15 minutes.

Network bandwidth analysis

Wireshark network protocol analyser [252] and a smart switch with port mirror functionality were used to to capture all the network traffic among the different devices. Fig. 5.14 illustrates a detailed result of the measured robot-robot, robot-server and machine-server communication for the different configurations and a summary of the measured results is provided in Table 5.4.

The conclusions drawn from the analysis of network bandwidth are as follows:

- The implementation of a multi-master system led to a local registration of nodes, services and topics that otherwise would had been

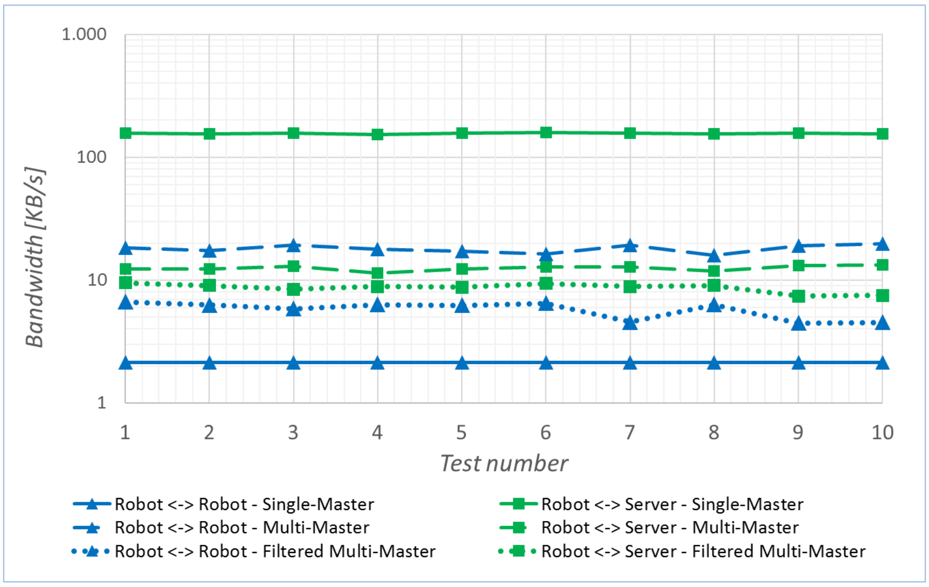


Figure 5.14. Representation of the network bandwidth between agents during 35 sec and 10 test for each configuration.

Table 5.4: Average Robot-Server, Robot-Robot and Machine-Server bandwidth summary.

	SingleMaster	**Multimaster**	**MultimasterFlt.**	
R-S	156	13	9	[KB/s]
R-R	2	18	6	[KB/s]
M-S	33	34	34	[KB/s]
$\sum$	191	64	48	[KB/s]

registered by broadcasting the network. This led to a significant network traffic reduction (156 KB/s $\rightarrow$ 13 KB/s) in the communication server. The network traffic among robots increases (2 KB/s $\rightarrow$ 18 KB/s) because topic and service registration data of remote masters were synchronized and locally stored in each master.

- By proper parametrization of the multimaster synchronization node, the XML data exchange can further be reduced (13 KB/s $\rightarrow$ 9 KB/s and 18 KB/s $\rightarrow$ 6 KB/s) by synchronizing only the necessary topics between agents.

- The data exchange between machines and communication server is not influenced by the ROS configuration, as it uses the OPC UA protocol for the communication.

It is important to keep topics local by using namespaces. If topics are subscribed and published in different agents, they cannot be ignored by using the described multimaster synchronization package and will be therefore synchronized. An example is the global and commonly used "/tf" topic. An intelligent topic handling will increase the network performance for the three configurations.

Service call analysis

To evaluate the system performance under real execution conditions, the agents were connected over a wireless network. This experiment assesses the issue exposed in subsection 5.1.3 which refers to need for agents running on an external device without having their own master API, to register and call all their topics and services over the network. The bidding calculation time of each robot is obtained over the service call *make_plan* offered by the ROS *move-base* node [253] running on each robot for the navigation. A summary of the test results is shown in Table 5.5.

The conclusions drawn from the service call analysis are as follows:

- Service-client connections are not kept fixed and must be repeated over the master API every time. In a single-master configuration, this request is sent over the network for every client on each robot.

Table 5.5: Analysis results of the bidding costs calculation for a replenishment task with three Kobukis in the system.

	Singlemaster	**Multimaster**	**MultimasterFlt.**	
Kobuki0	1,78	0,34	0,35	[s]
Kobuki1	1,82	0,37	0,35	[s]
Kobuki2	1,79	0,36	0,35	[s]

As a result, a bottleneck issue appears in the master registration API that must respond to a huge number of requests. Moreover, the requests are broadcasted over the network instead of locally which results, as expected, in slower response times.

- In the single-master configuration with three robots, the cost calculations took approximately 1.8 seconds while in the multi-master configurations only 0.35 seconds. This is because most registrations are distributed and handled locally. Only the necessary agent-to-agent information should be transmitted over the network. This phenomenon was noticed again on the following scalability tests.

- There was no remarkable difference in the performance between the standard and the filtered multi-master configurations. In both cases the service call is handled locally and it is, therefore, independent of the multi-master function.

Scalability analysis

The scalability tests were focused on the average bidding duration, i.e., the time that elapses between a machine publishing a new task (received by the available robots over the network), the robots calculating their costs and the machine receiving back the bids from the robots (again through the network).

A single replenishment task was triggered to calculate the bidding durations by introducing up to six, maximum available, MMRs. As each machine waits for a maximum defined time to receive robot bids, it is important that the MMRs calculate and send their bids on-time to the machine. Otherwise, the robot will not be taken into account for the task allocation process. The maximum bidding time has been set to 2 seconds. The results are shown in Figure 5.15.

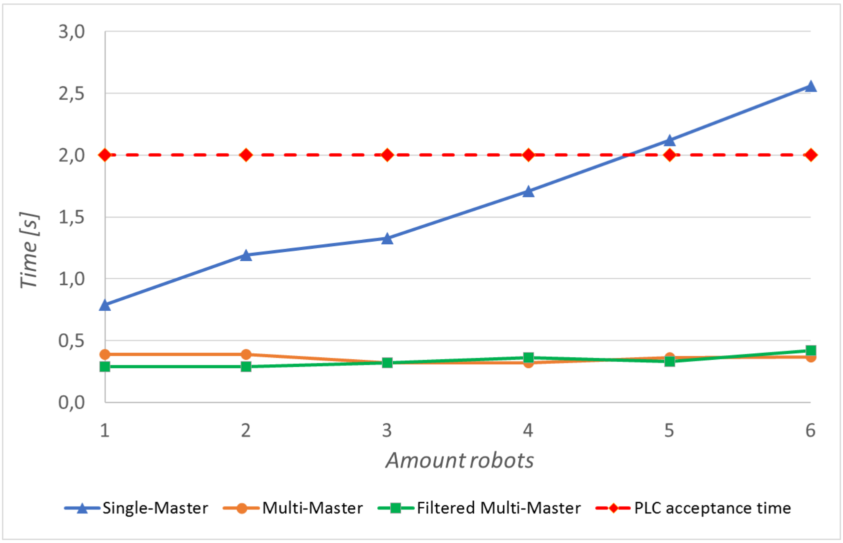


Figure 5.15. Scalability test results with up to six MMRs.

The conclusions drawn from the scalability analysis are as follows:

- An increasing number of MMRs in a single-master configuration leads to a linear raising bidding duration for each robot, while in a multi-master configuration the bidding time is not affected by the number of MMRs.
- The more agents in the system, the more topics and services that have to be managed. This is in concordance with previously per-

formed tests. By distributing the master API most services are maintained locally and the network bottleneck and overloading problems are avoided, increasing the scalability of the system.

- Due to the maximum bidding time acceptance in the machines, only up to four robots could be used in the single-master configuration. With more robots, the bidding times were too slow and, therefore, they were not accepted by the machines. However, the multi-master configurations show a stable bidding time of under 0.5 seconds regardless of the number of robots in the system.
- It takes approximately 1 minunte to launch the system with the single-master configuration and 6 robots, whereas the multi-master is ready within 15 seconds regardless of the amount of robots used. In addition, while MMRs could not be launched at the same time in the single-master tests, it was possible to do so using the multi-master configuration.

5.2.2.3 Summary of the ROS configurations experiments

These tests showed that a decentralized middleware is necessary to build a ROS-based MAS. A properly filtered multi-master configuration, that maintains own topics and services locally, decreases the network bandwidth, avoids the bottleneck problem, and increases system time responses, such as those involved in the bidding process. The tests also showed that the multi-master configuration improves the scalability keeping the response times between agents nearly constant regardless of the amount of robots in the system.

This scenario could be improved by running further scalability tests where dozens of ROS agents communicate among each other to ascertain the current scalability results on a crowded system.

5.2.2.4 Validation tests results

In order to validate the elected filtered multi-master configuration, a final experiment was performed where the system runs non-stop for 15 minutes. For this experiment, machine 1 was configured to require new material every 110 seconds and deliver its manufactured products every 60 seconds. In case of machine 3, these values were set to 160 seconds and 76 seconds, respectively. If machines run out of material or had their delivery depot overloaded, the production was temporarily stopped until the MMRs have finished their transportation tasks. The test started with the assumption that the machines were full of material and producing. The results in Table 5.6 present the activity of machines and MMRs. A video of the performed tests with explanations can be found in [254].

Table 5.6: Test results for 15 min non-stop replenishment (RP) and delivery (DL) transportation with six Kobukis, two machines, a communication server and a monitoring panel.

	MA1 RP-DL	**MA3 RP-DL**	**∑Success**	**∑Failed**
MMR0	0 - 12	0 - 0	12 / 12	0
MMR1	5 - 0	3 - 0	8 / 8	0
MMR2	1 - 0	2 - 0	3 / 3	0
MMR3	0 - 0	0 - 8	8 / 8	0
MMR4	0 - 0	0 - 0	0 / 0	0
MMR5	0 - 0	0 - 1	1 /1	0
∑	6 - 12	5 - 9	32 / 32	0

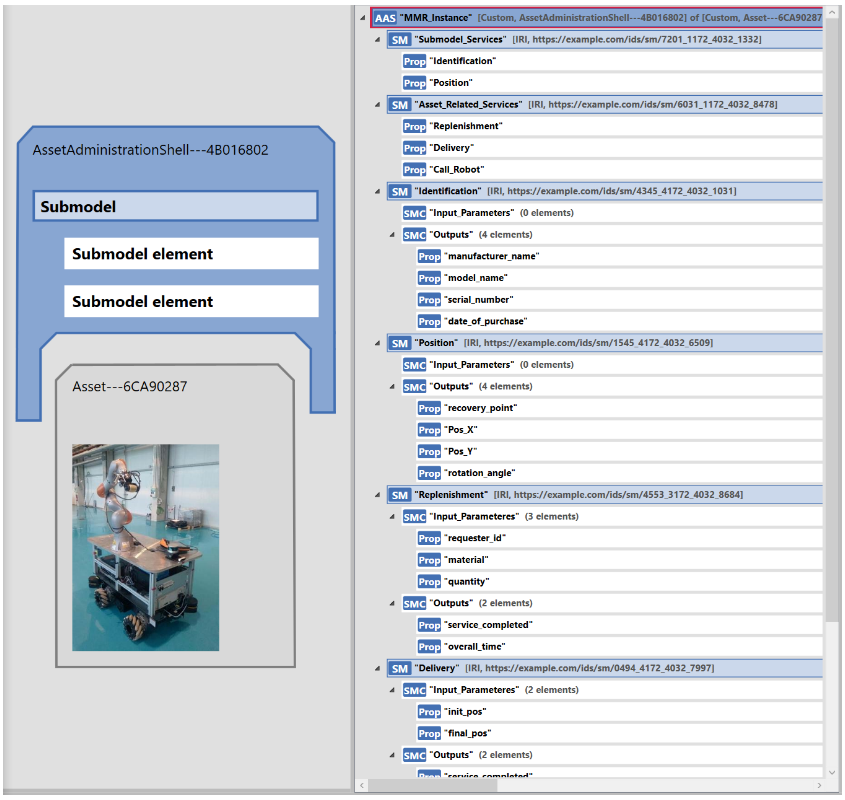
The conclusions drawn from the validation test are as follows:

- According to the closeness from robots starting positions to source and delivery areas, MMRs 0, 1 and 3 offered lower bidding costs and were, therefore, more likely assigned than their partners in the charging stations.
- Towards the end of the tests, a navigation failure occurs while MMR3 tries to drive back to its initial position. Fortunately, the delivery was already finished, so it did not affect the system. The operator intervened to bring the MMR3 to its initial position. The Kobuki was restarted, it localized by itself in the map, and it was already taking part on new biddings a few seconds later.
- The fast restart of MMR3 shows the ability to quickly add new robots into the system if necessary.
- Although the evaluation of the proposed multi-agent architecture and the selected ROS configuration successfully met all the expectations, the results also show that an optimization of the task publications times must be done.

5.2.3 Integration of MMRs in a flexible manufacturing system based on ROS-JADE multi-agent system

This section contributes to the definition of a generic multi-layer architecture for MMR integration in the factory, providing I4.0 Components with social abilities to access *Application Relevant Services* that provide MMR-related functionalities, as depicted in Figure 5.16.

To that end, the concerns related to MMR integration have been abstracted into layers, so that they can be addressed both generically (i.e., regardless of the type of MMR to be integrated) and independently (i.e.,



```
AAS "MMR_Instance" [Custom, AssetAdministrationShell---4B016802] of [Custom, Asset---6CA90287
  SM "Submodel_Services" [IRI, https://example.com/ids/sm/7201_1172_4032_1332]
    Prop "Identification"
    Prop "Position"
  SM "Asset_Related_Services" [IRI, https://example.com/ids/sm/6031_1172_4032_8478]
    Prop "Replenishment"
    Prop "Delivery"
    Prop "Call_Robot"
  SM "Identification" [IRI, https://example.com/ids/sm/4345_4172_4032_1031]
    SMC "Input_Parameters" (0 elements)
    SMC "Outputs" (4 elements)
      Prop "manufacturer_name"
      Prop "model_name"
      Prop "serial_number"
      Prop "date_of_purchase"
  SM "Position" [IRI, https://example.com/ids/sm/1545_4172_4032_6509]
    SMC "Input_Parameters" (0 elements)
    SMC "Outputs" (4 elements)
      Prop "recovery_point"
      Prop "Pos_X"
      Prop "Pos_Y"
      Prop "rotation_angle"
  SM "Replenishment" [IRI, https://example.com/ids/sm/4553_3172_4032_8684]
    SMC "Input_Parameteres" (3 elements)
      Prop "requester_id"
      Prop "material"
      Prop "quantity"
    SMC "Outputs" (2 elements)
      Prop "service_completed"
      Prop "overall_time"
  SM "Delivery" [IRI, https://example.com/ids/sm/0494_4172_4032_7997]
    SMC "Input_Parameteres" (2 elements)
      Prop "init_pos"
      Prop "final_pos"
    SMC "Outputs" (2 elements)
      Prop "service_completed"
```

Figure 5.16. AAS overview (snapshot generated with the "AASX Package Explorer" tool). *Application Relevant Services* offered by the MMR are composed of *Submodel Services* and *Asset Related Services*.

being decoupled from each other). The points that were set to meet this objective were to provide MMRs with distributed decision making and to create a series of elements that would allow the integration of the ROS environment with an agent-based manufacturing application management platform, developed on the JADE platform.

5.2.3.1 Multi-layer integration architecture

Asset integration in Industry 4.0 involves asset-related service implementation and management. If those concerns are detached, they can be addressed in a generic way, so that they are applicable to any type of MMRs, regardless of their communication capabilities and RFs used for their development. To that end, the four-layer integration architecture is illustrated in Figure 5.17.

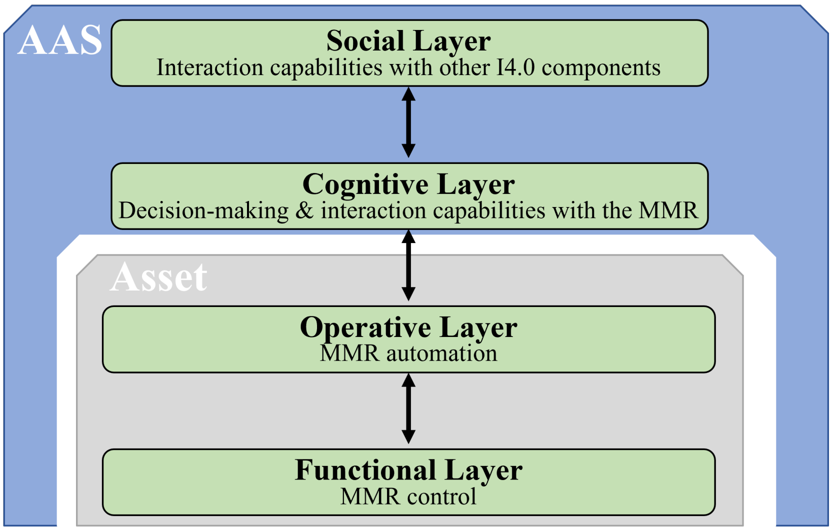

AAS
Social Layer
Interaction capabilities with other I4.0 components
Cognitive Layer
Decision-making & interaction capabilities with the MMR
Asset
Operative Layer
MMR automation
Functional Layer
MMR control

Figure 5.17. Multi-layer approach for MMR integration in a factory as an I4.0 Component.

- The upper layer (Social) implements the communication capabilities needed by the MMR to interact with other I4.0 Components to attend requests for *Application Relevant Services*.
- The second layer downwards (Cognitive) manages service requests as follows: when a request corresponds to a *Submodel Service*, it can be resolved directly at this layer; when a request refers to an

Asset Related Service, it is necessary to pass the request to the MMR (e.g., to ask the MMR to move or manipulate, or to request information about its position or battery level) . In addition, as two MMRs may offer the same *Application Relevant Services* but have different communication capabilities, this layer is also responsible for abstracting communications with the MMR.

- The layer downwards (Operative) manages *Asset Related Service* requests by dividing them in smaller navigation and manipulation functionalities that are executed in an orderly manner. This involves monitoring the MMR state, and acting on it or reporting events.

- The bottom layer (Functional) implements the execution of the navigation and manipulation functionalities composing the MMR-related *Asset Related Services*, as well as manages the communication with the drivers. Thus, it deals with the basic MMR control.

The Social and the Operative layers are located in the AAS, while the Operative and Functional layers are located in the asset, i.e., in the MMR. Considering that the MMR follows the Robotframework architecture, a mapping of the layers of the integration architecture within Robotframework can be performed as shown in Figure 5.18:

- The Social and Cognitive layers of the integration architecture map to the Decision layer of Robotframework. Therefore, the Decision layer of Robotframework is the AAS of the MMR.

- The Operational layer of the integration architecture maps to the Application layer of Robotframework, whereas the Functional layer maps to the Abilities and Drivers layer.

Since the operational and functional layers map to layers already discussed in Chapter 4, only Social and Cognitive layers will be described

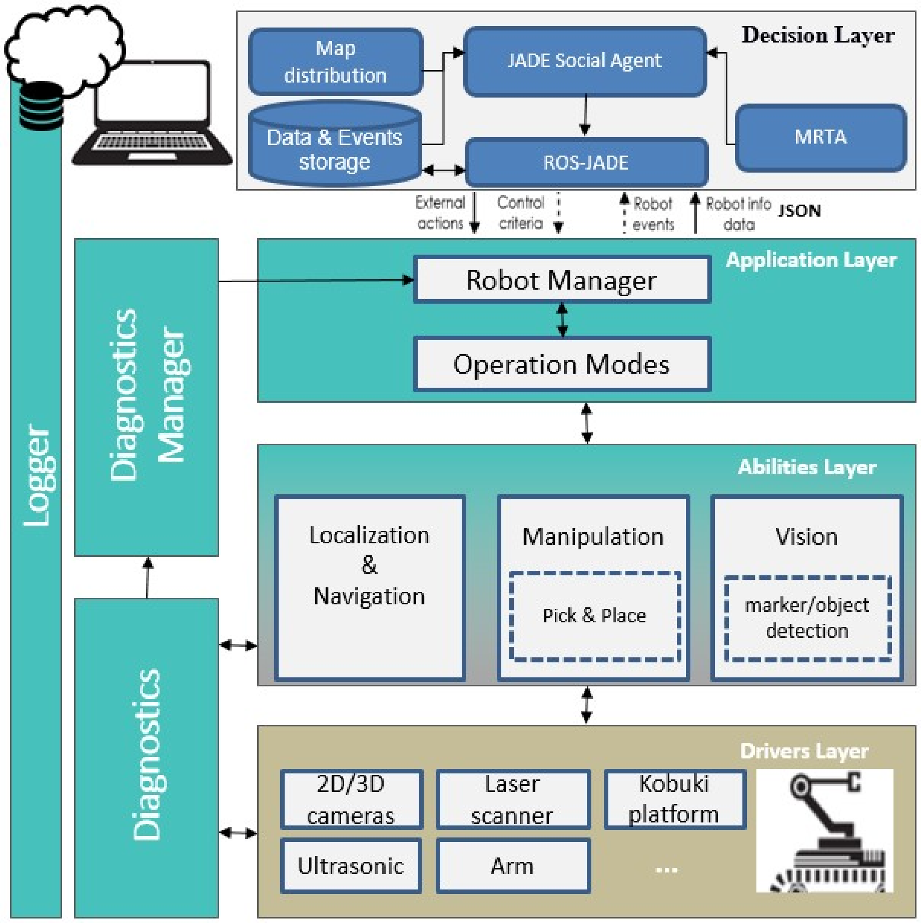


Figure 5.18. Mapping integration architecture into Robotframework.

in this section.

Social layer

The agent (in the prototype, a JADE Agent) offers, at least, transportation services (SR1), and communicates with the rest of the I4.0 Components in the I4.0 system, negotiating and cooperating with them (SR5). However, since a MMR is made up of multiple components, it is possible to offer additional services, such as the use of integrated cameras for monitoring purposes or localization services, giving the possibility of global localization to other entities having limited resources.

The agent in this layer represents the global intelligence of the MMR as an I4.0 Component. Its main functions are offering the MMR capabilities as services, dealing with the requests of other I4.0 Components in the system, and negotiating with other MMRs to decide which of them will actually perform the service. Whenever the agent gains a negotiation or receives an event from another system element, the social layer transmits the order downwards through utilities of the cognitive layer, making it independent from RF and MMR functionalities.

From a structural point of view, the agent incorporates a Finite State Machine which consists of the following states:

- A *boot* state where the agent reads all the information related to the MMR from the AAS *Submodels* and registers itself in the System Repository, if the availability of the physical asset is verified. Then, any initialization task can be performed.

- A *running* state that manages the negotiations for service allocation, the requests for *Application Relevant Services*, and the interactions with the asset. Besides, in this state agent and MMR liveness are also checked.

- An *unavailable* state for situations in which the MMR is not capable of performing more tasks. This situation can occur if the actuators fail or the battery is too low. In this state, it is possible to check agent and MMR liveness just as in the running state. In this way, the agent can find out whether the battery is sufficient again, errors have been fixed or a new error has occurred.

- A *stop* state that performs the finalization tasks.

Cognitive layer

This layer decouples communications between the agent and the MMR into an auxiliary agent, named Gateway Agent (GA). The GA serves as a bridge between the AAS, which uses ACL protocol, inherent to FIPA agents, and the MMR, which uses a ROS communication protocol. The interface of this layer is provided with two methods used to standardize interactions between MMRs and the GAs: *sendDataToMMR* and *rcv-DataFromMMR*.

The GA package integrates all the nodes and Java classes necessary to carry out the communication between a system developed in a Java environment, such as JADE, and the MMR developed in ROS. In essence, it is an environment made up entirely of Java classes, in which there are no ROS nodes per se. However, since it is a package which is developed using *rosjava*, there are Java classes that have the capacity to launch ROS nodes and integrate them into the JADE platform. The most significant classes are summarized as follows and their coordinated operation can be seen in the Figure 5.19:

- **ACLGWAgentROS:** This class integrates the ROS functionalities required to launch a node. This node, with the same name as the class, is in charge of communicating with the ROS environment through the corresponding topics. At the same time, it has the ability to integrate within the JADE platform, configuring the communication with the agents and the name it will take within the platform. Its behavior and agent functionalities are defined within the *GWagentROS* class, which it instantiates.

- **GWagentROS:** This Java class defines the behavior that the GWAgent agent will take regarding its activities within the MAS platform. It is a kind of appendix that provides the ROS node with the capacity to send and receive messages from the rest of the agents that will integrate the platform.

- **StructCommand:** This is the class where the necessary fields for the communication between the transport agents and the GWAgent agent are collected.

It should be noted that, as mentioned above, the ACLGWAgentROS class launches a ROS node that allows it to integrate into the ROS environment as if it were just another ROS node. This node is able to subscribe and publish to topics in the same way as the rest of the nodes in the system, with the exception that, instead of being developed in Python or C++, it is developed in Java. Therefore, it has the possibility of using the necessary libraries, methods and resources to integrate into the JADE platform.

5.2.3.2 Use cases

This section presents several use cases showing the socialization of MMRs among themselves and with other non-robotic CPSs to demonstrate the benefits of combining RF and MAS.

- The first use case shows an MMR -Transportation Agent (TA)- noticing internal low battery levels and its interaction with available charging stations -Charging Station Agents (CSA)-.
- The second use case shows a material replenishment order triggered by a machine -Machine Agent (MA)- and the resulting task allocation between available MMRs.
- The third use case shows a smart sensor which informs the MMR that it is entering a human-working area, triggering a robot reconfiguration.

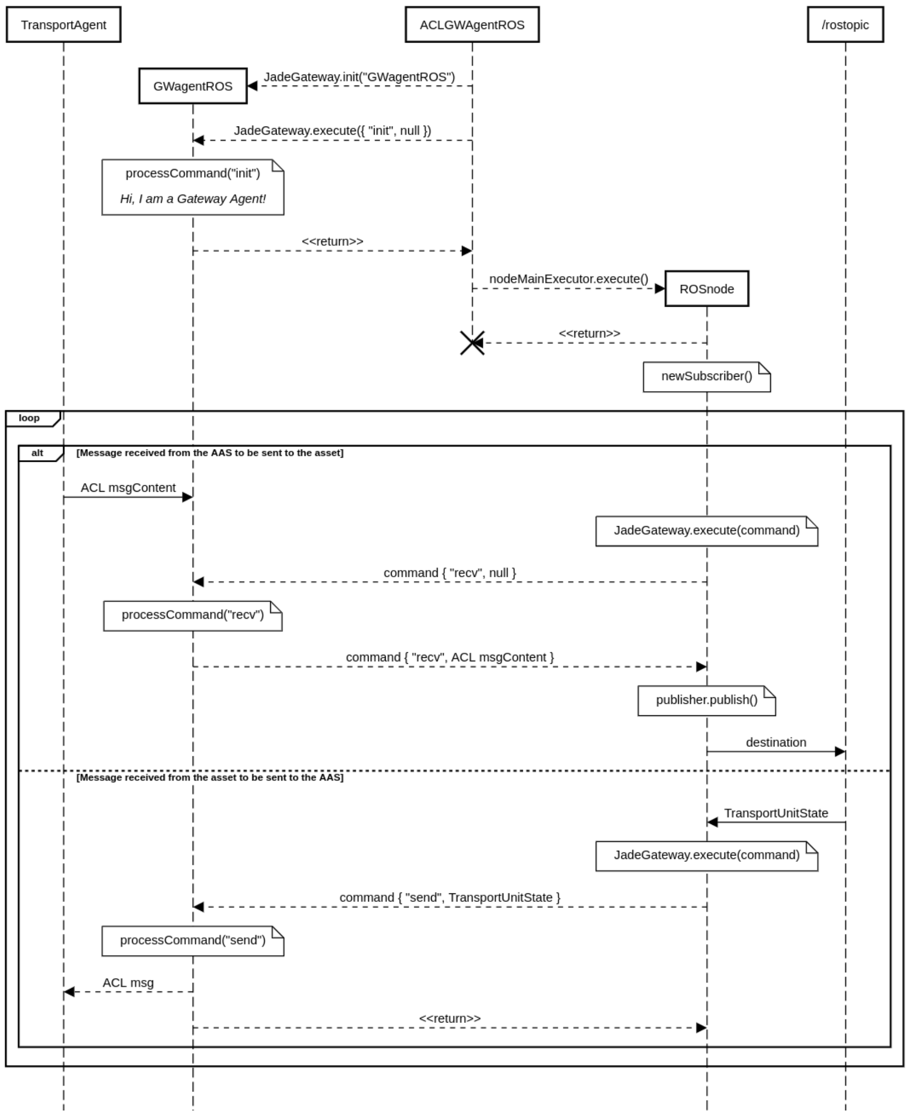

TransportAgent
ACLGWAgentROS
/rostopic
GWagentROS
JadeGateway.init("GWagentROS")
JadeGateway.execute({ "init", null })
processCommand("init")
Hi, I am a Gateway Agent!
<<return>>
nodeMainExecutor.execute()
ROSnode
<<return>>
newSubscriber()
loop
alt
[Message received from the AAS to be sent to the asset]
ACL msgContent
JadeGateway.execute(command)
command { "recv", null }
processCommand("recv")
command { "recv", ACL msgContent }
publisher.publish()
destination
[Message received from the asset to be sent to the AAS]
TransportUnitState
JadeGateway.execute(command)
command { "send", TransportUnitState }
processCommand("send")
ACL msg
<<return>>

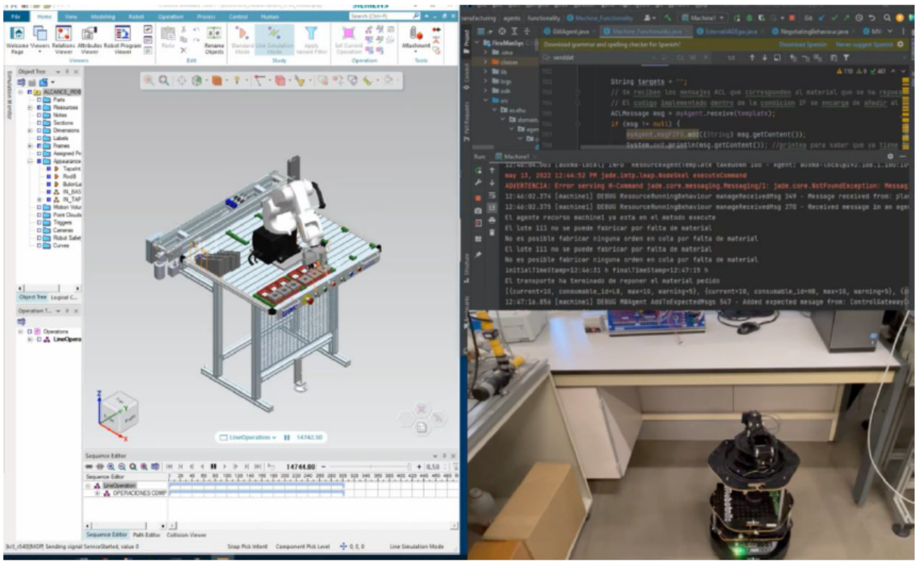
Figure 5.19. The operation of the Gateway Agent for decoupling communications between the AAS and the MMR.

CASE 1: Management of ATV Battery State

When the ATV notices critical battery levels, it requires a free position in a charging station. The process is carried out in five steps: 1) the MMR agent (the TA) receives a low battery event issued from the Functional layer; 2) TA requests available power stations to the DF (*AAS Infrastructure Service*); 3) the TA triggering a negotiation among them in order to contract their charging *Asset Related Service*; 4) CSAs negotiate under the specified criterion, e.g., nearest to the MMR, calculating their cost and sending it to other participants. The winner of the negotiation informs TA about the pose; 5) the MMR navigates to that pose.
The battery state of a MMR robot is also valuable information that may be required on demand by higher level operator or autonomous monitoring systems. This information is internally stored on the *cognitive layer* and requested by the monitoring agents as a *Submodel Service*. Other information that is usually required by monitoring agents is related to its current position, working mode or performing task.

CASE 2:Machine Material Replenishment

The intelligent MMR provides high flexibility to manufacturing processes managing a dynamic plan to serve on-demand requests. This allows solving expected or unexpected events in the plant in an agile way. For instance, when a machine needs material to perform its operations, its agent (the Machine Agent -MA-) makes use of *Asset Related Transport Service* to perform a material replenishment (see Figure 5.20). The replenishment transportation service request can be carried out in five steps: 1) The MA detects lack of material; 2) MA requests available transportation robots to the DF (*AAS Infrastructure Service*); 3) Initiates a negotiation among TAs under a specified criterion; 4) The winner MMR notifies the MA; 5) The MMR starts the transportation task informing the MA about the beginning and the end of the task.
The information related to the transportation (in the case of the MMR)



```
String targets = "";
// Se reciben los mensajes ACL que corresponden al material que se ha repues
// El codigo implementado dentro de la condicion IF se encarga de añadir al
ACLMessage msg = myAgent.receive(template);
if (msg != null) {
    myAgent.msgFIFO.add((String) msg.getContent());
    System.out.println(msg.getContent()); //printea para saber que ya tiene
```

```
may 13, 2022 12:44:52 PM jade.imtp.leap.NodeSkel executeCommand
ADVERTENCIA: Error serving H-Command jade.core.messaging.Messaging/1: jade.core.NotFoundException: Messag
12:46:02.374 [machine1] DEBUG ResourceRunningBehaviour manageReceivedMsg 249 - Message received from: pla
12:46:02.375 [machine1] DEBUG ResourceRunningBehaviour manageReceivedMsg 270 - Received message in an age
El agente recurso machine1 ya esta en el metodo execute
El lote 111 no se puede fabricar por falta de material
No es posible fabricar ninguna orden en cola por falta de material
El lote 111 no se puede fabricar por falta de material
No es posible fabricar ninguna orden en cola por falta de material
initialTimeStamp=12:46:31 h finalTimeStamp=12:47:15 h
El transporte ha terminado de reponer el material pedido
[{current=10, consumable_id=L8, max=10, warning=5}, {current=10, consumable_id=H8, max=10, warning=5}, {
12:47:16.856 [machine1] DEBUG MRAgent AddToExpectedMsgs 547 - Added expected mesage from: ControlGateway
```

Figure 5.20. Interaction between the MMR and the machine.

or manufacturing task (in the case of machines) is valuable information that can be used for traceability purposes and which required on demand by higher level operator or autonomous monitoring systems. This information is internally stored on the *cognitive layer* and requested by the monitoring agents as a *Submodel Service*.

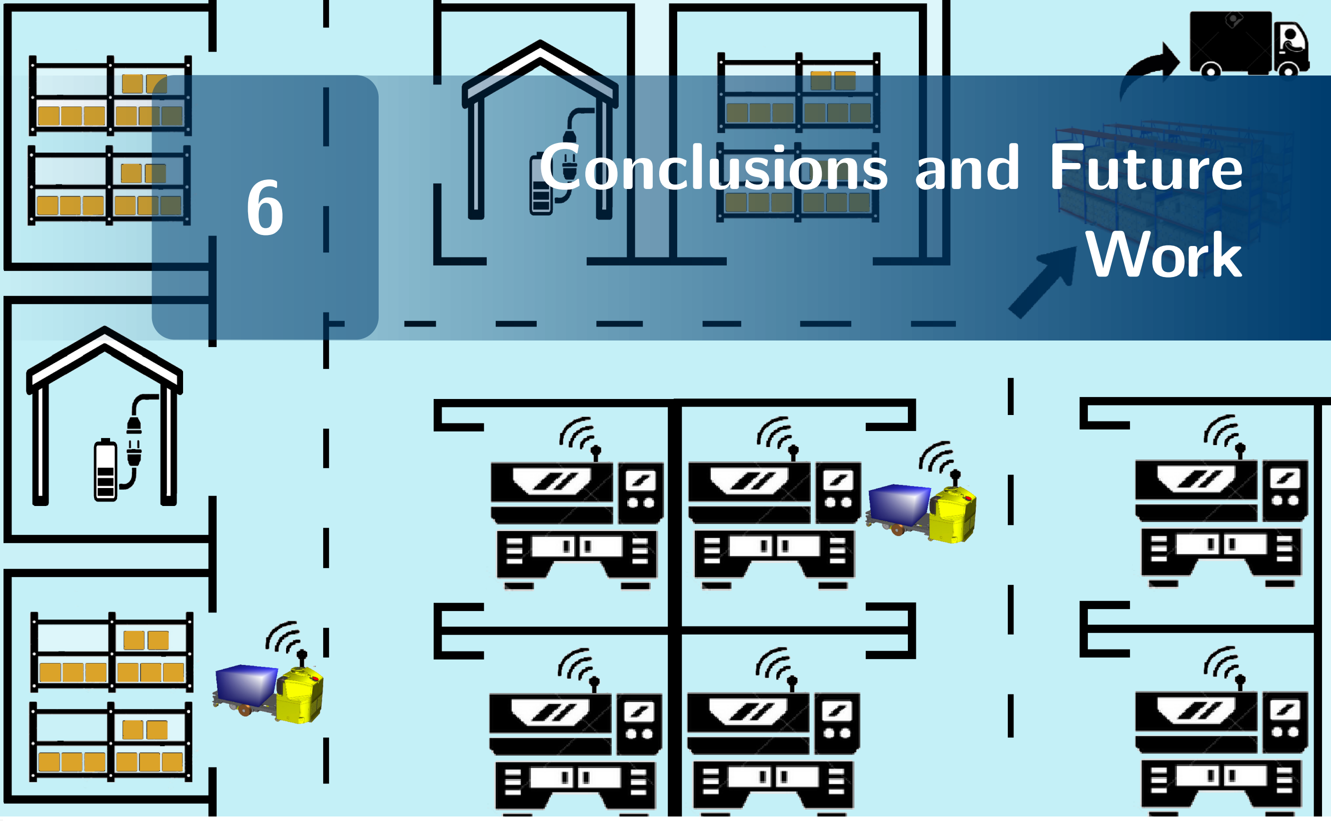
CASE 3: ATV Speed Adaptation on Demand

There are manufacturing entities that are not intelligent enough to be represented as agents, but that could still transmit important information to MMRs, triggering, if necessary, MMR reconfiguration. This is the case of restricted area sensors. They can be easily settled in strategic locations to alert about the entering into a special area. In order to receive this information, the robot must include the hardware component that receives the sensor signal in the functional layer and implement a new monitor in the operational layer to pre-process this information and create the events that wake up the social agent . These events will trigger, then, the MMR reconfiguration.

For example, a robot without load could navigate at maximum speed within an automated warehouse where human operators are not allowed to enter. Thus, the transportation performance is improved, while the risk of hurting humans is avoided. As reconfiguration actions are not directly related to the transportation task, robots must provide *Submodel Services* that permits triggering such reconfiguration of internal robot parameters.

5.2.3.3 Summary

The multi-layered architecture meets the requirements collected in 5.1, while abstracting the social abilities from the control functionalities, decoupling attention to service requests from the high frequency information refreshing at functional level, promoting control code re-use and separation of concerns, as higher-level services can be adapted without modifying the functionality and vice versa. The division on layers is done with efficiency and modularity in mind, avoiding functionality overlapping between layers. This architecture has been implemented on ROS and JADE, the most widespread RF and MAS frameworks, respectively, which offer the necessary base to develop a MARS. In addition, the use cases presented in this work contribute to illustrate how ATVs based on this architecture are able to collaborate with machines, charging stations or environmental sensors in the factory, efficiently responding to transportation service requests and adapting to context changes.



6 Conclusions and Future Work

This chapter summarises the accomplishments of the research process and outlines the key conclusions drawn from it. Additionally, it presents the published contributions and proposes some possible future steps and open questions that are yet to be solved.

6.1 Conclusions

Modern manufacturing is evolving rapidly to meet global market demands, leading to the concept of Industry 4.0, where simplified, modular, and efficient processes with decentralized control are favored. The MMR plays a vital role in intelligent manufacturing by replacing fixed conveyor belts with flexible transportation robots and intelligent machines. This enables quick and cost-effective production system reconfiguration and allows for unplanned or on-demand deliveries. RFs aid in component integration, but the development of logic, relationships between modules, and social abilities for intelligent interaction among diverse entities within Cyber Physical Production Systems remain significant.

The thesis began by posing several research questions that served as guiding principles throughout the investigation. These questions included exploring the challenges faced by companies in transitioning to Industry 4.0, identifying design paradigms that could facilitate this transition, and understanding the key concepts considered by reference architectures in the interpretation of Industry 4.0. The thesis also aimed to uncover the requirements for the design and development of MMRs in the context of Industry 4.0, as well as the necessary information models, functionalities, and architectural organization. Additionally, the research investigated the integration of multi-MMR systems in the factories on the future considering as a reference the Platfform Industrie 4.0.

A significant effort of the thesis involved an in-depth analysis of digitalization initiatives in leading industrialized countries, specifically focusing on MMRs. This analysis led to **one of the main results** of this thesis, **the definition of the main design requirements for a generic robotic system suitable for the flexible factory of the future**. Taking into account these requirements, various robotic, multi-agent and OPC UA frameworks have been analysed and their characteristics have been summarized in Table 3.1. The list includes frameworks with distributed and modular design (R2,R3), open-source (C6), multi-platform (C7) and, if possible, based on standards (C4). On the one hand, *Robotic Frameworks*, offer support of physical sensors and actuators (C1), robotic algorithms (C2), simulation and modelling tools (C3). Here, ROS emerged as the most popular and well-suited framework, with widespread adoption and support from industrial robot manufacturers. On the other hand, integrating the multi-agent paradigm into RFs could advance socialization (R5), communication management (R6), recovery (R7) and reconfiguration (R8) capabilities as well as the overall communication security (C5). Here, the JADE multi-agent framework was chosen for its high performance, ease of use, and adherence to widely recognized standards. Additionally, OPC UA frameworks were identified as complementary enablers for communication between machines, typically controlled by PLCs, robots and other CPPSs.

The **first major contribution** is centered on the **development of Robotframework, a generic control architecture enabling the integration of single robot capabilities for a wide range of mobile manipulation applications**. Combining the different robotic skills is an error prone work that requires experience in many robotic fields, usually deriving on ad-hoc solutions that are not reusable in other contexts. Robotframework aids at integrating different navigation, manipulation, perception, and high-decision modules in a standardized form, leading to a faster and simplified development of new robotic applications. The architecture includes generic real-time data collection tools, diagnosis and error handling modules, and user-friendly interfaces.

This architecture is considered generic because it is based on ROS, which provides tools for developing robotic skills in a generic and standardized manner to work with a wide variety of hardware and software. It also promotes the standardization of skills based on pluginlibs that together with the meta-model and JSON format, proposed for plans generation, enable interoperability and homogeneous execution of workflows across different robots and robotic systems. All this without changing the main core of the architecture. Thus, making use of Robotframework, any high-level planner, which is application dependent, can generate a plan for any robot, just being aware of the abilities provided by the concrete robot (implemented by means of the pluginlib concept) as well as the proposed meta-model guidelines and the JSON format.

Realistic use cases, including an industrial aileron inspection application (CRO-INSPECT) and a pest inspection and treatment application (Greenpatrol), validated the architecture's effectiveness and practicality. These use cases served to validate and demonstrate the generalization capacity of Robotframework, and to showcase its implementation possibilities for future applications. As a result, it can be concluded that the modular design and pluginlibs of Robotframework facilitate its reuse in different contexts without major modifications, as it abstracts application dependent modules from robot dependent modules, significantly reducing the time required to develop new mobile manipulation applications. Notably, the framework's flexibility is exemplified through various navigation modes and mobile manipulation tasks presented as plugins, offering templates for future applications. Figure 4.48 presents a summary of the already integrated applications with Robotframework and how the previously introduced applications could be integrated with it.

However, while Robotframework effectively manages heterogeneous MMRs, it lacks social features within its architecture. As a response to this limitation and guided by design principles outlined in industrial reference architectures such as RAMI 4.0, the thesis pursued the enhancement of social capabilities, that leads to the second contribution of this work.

The **second major contribution** of this thesis focuses on the **seamless integration of Robotframework into the factories of the future by means of Asset Administration Shell**. From a scientific point of view, this approach offers a new perspective where MMR integration is not treated as just another aspect of AAS implementation, but as its main concern. To that end, heterogeneous production entities in the smart factory have been identified and the service categories defined in the Functional View of the AAS have been mapped to them.

From a methodological point of view, the integration is addressed through a multi-layer approach that decouples integration-related concerns into concrete, manageable tasks and provides guidance on how to define, implement and deploy services. Moreover, the proposed integration practices adhere to RAMI 4.0 concepts to meet the requirements collected in 5.1, ensuring asset-related transportation services (SR1-SR2) while abstracting social abilities from the control functionalities, as well as other submodel services to notify significant events at social level (SR3) or to allow internal parameters reconfiguration and adaptation on context changes (SR4). Finally, the integration with MAS and industrial standards ensures the communication with any type of CPPS (SR5).

From an applied point of view, the development and evaluation of distributed task allocation and traceability mechanisms using Multi-Agent Systems (MAS) and the industrial OPC UA standard demonstrate the MMRs' adaptness in transporting products and replenishing raw materials within the factory. The integration showcases the adaptability and responsiveness of MMRs, effectively responding to transportation service requests and dynamically adapting to contextual changes. The implementation of this architecture has been successfully executed on ROS and

JADE, the widely adopted RF and MAS, respectively. The presented use cases illustrate how MMRs based on this architecture seamlessly collaborate with machines, charging stations, and environmental sensors within the factory. The MMRs efficiently respond to transportation service requests and adapt to changing contextual situations.

The consideration of the AAS concept and the conceptual service model of the ASS (i.e., the service categories of the ASS) ensures the integration architecture's interoperability, a prerequisite for facilitating their adoption by companies. Besides, the successful implementation and validation of this architecture further reinforce its applicability in future smart manufacturing systems. Overall, these contributions emphasize the significance of Robotframework and its potential to drive the advancement of Industry 4.0 principles, solidifying its position in the realm of industrial automation and smart factories.

The contributions described in this report have given rise to a set of results that have been published in journals with an impact index and presented at international conferences of recognized prestige in the field of research. The work carried out in relation to *the development of Robotframework, a generic robotic architecture for MMRs* has resulted in the following publications:

(a) **J. Martin**, A. Ansuategi, I. Maurtua, A. Gutierrez, D. Obregón, **O. Casquero**, and M. Marcos, "A Generic ROS-Based Control Architecture for Pest Inspection and Treatment in Greenhouses Using a Mobile Manipulator", in IEEE Access, vol. 9, pp. 94981-94995, 2021, doi: 10.1109/ACCESS.2021.3093978.

(b) M. Pattinson, S. Tiwari, Y. Zheng, D. Fryganiotis, M. Campo-Cossio, R. Arnau, D. Obregón, **J. Martin**, C. Tubio, I. Lluvia, O. Rey, J. Verschoore, D. Húska, L. Lenza, J.R. Gonzalez, "Galileo

Enhanced Solution for Pest Detection and Control in Greenhouses with Autonomous Service Robots", 2020 European Navigation Conference (ENC), Dresden, Germany, 2020, pp. 1-10, doi: 10.23919/ENC48637.2020.9317451.

(c) **J. Martin**, H. Engelhardt, M. Masannek, T. Scholz, K. Gillmann, B. Schadde (2019). "RoboCup@Work 2018 Team AutonOHM", Robot World Cup XXII. RoboCup 2018 Lecture Notes in Computer Science, vol 11374. Springer, Cham, doi: 10.1007/978-3-030-27544-0_41

(d) **J. Martin**, H. Engelhardt, T. Fink, M. Masannek, T. Scholz, (2018). "RoboCup@Work Winners 2017 Team AutonOHM", Robot World Cup XXI. RoboCup 2017 Lecture Notes in Computer Science(), vol 11175. Springer, Cham, doi: 10.1007/978-3-030-00308-1_41

(e) **J. Martin**, T. Fink, S. May, C. Ochs, and I. Cabanes, "An Autonomous Transport Vehicle in an existing manufacturing facility with focus on the docking maneuver task", in 2017 3rd International Conference on Control, Automation and Robotics, ICCAR 2017, 2017, pp. 365–370.

With respect to the integration of Robotframework on the factories of the future, the following publications can be highlighted:

(a) **J. Martin, O. Casquero**, B. Fortes and M. Marcos, , "A Generic Multi-Layer Architecture Based on ROS-JADE Integration for Autonomous Transport Vehicles", Sensors, 2019 19(1), 69, doi:10.3390/s19010069

(b) M. López, **J. Martin**, U. Gangoiti, **A. Armentia**, E. Estévez, M. Macos, "Tolerancia a fallos en sistemas de fabricación flexible basado en MAS" Jornadas de Automática, 2018, Badajoz. (Price for the best work in computers and control).

(c) M. López, **J. Martin**, U. Gangoiti, **A. Armentia**, E. Estévez, M. Macos, "Supporting Product Oriented Manufacturing: a Model Driven and Agent based Approach", 16th IEEE International Conference on Industrial Informatics (INDIN), 2018, Oporto.

(d) **J. Martin**, S. Endres, M. Stefan, and C. Itziar, "Decentralized Robot-Cloud Architecture for an Autonomous Transportation System in a Smart Factory," in 2nd International Workshop on Linked Data in Industry 4.0 (LIDARI). CEUR Workshop Proceedings (CEUR-WS.org), 2017.

6.2 Future Work

The work conducted in this thesis has laid a strong foundation for future research in the realm of mobile manipulation robotics and smart manufacturing systems. The development of the Robotframework architecture, validated through realistic use cases, showcases its potential for diverse applications and its seamless integration within the factories of the future. To further enhance its versatility, future work should focus on integrating navigation goals into tasks, expanding the architecture's applicability to a wider range of applications, conducting extensive testing with new use cases, and transitioning to ROS2 for improved communication mechanisms among robots. These endeavors promise to elevate the framework's adaptability, robustness, and scalability, contributing to the advancement of Industry 4.0 principles and the evolution of smart factories on a global scale.

Chapter 4 of this thesis introduced the targets-plan-operation mode and its correlated meta-model, which exhibits a generic and versatile nature applicable to a wide spectrum of mobile manipulation applications. However, there are potential improvements that could render it even more inclusive, such as integrating navigation goals into tasks. Initially focused on mobile robots, the operation was expected to always have a navigation goal, but as evident from the analysis, this may not always be the case. By merging navigation goals and tasks under the unified concept of "tasks," the framework's versatility and coherence can be further enhanced. This vision holds true for all the use cases presented in this work, including Robocup@work, Bosch production system, and cro-inspect, streamlining both the state machine's complexity and the development of new navigation and manipulation abilities.

Furthermore, the architecture holds the potential to extend its utility to a broader range of applications that might not necessarily involve navigation but could still benefit from the well-structured framework of Robotframework. By reusing several packages provided, such as the GUI, robot_manager, specific robotic skills, or the diagnostics and monitoring modules, these applications can capitalize on the framework's modularity and efficiency.

A future direction for enhancing the architecture involves its integration into ROS2, which holds the potential to address various issues related to a single master and elevate the quality of communication mechanisms among robots. This upgrade could further elevate the framework's performance, scalability, and compatibility with the latest advancements in robotics and automation technologies.

To reinforce the robustness of the architecture, it is essential to subject it to further testing with new use cases. By exploring additional scenarios, such as human-robot collaboration or transportation use cases requiring load manipulation, the architecture's adaptability and applicability can be thoroughly assessed and refined. For instance, a use case where the product agent triggers all the required transportation and manufacturing services while simultaneously performing traceability would be an intriguing avenue for exploration. Mobile robotics offers a wide range of possibilities, from ambitious applications to more modest ones, with the potential to be employed in various fields. Virtually any task can be streamlined and automated, either by a mobile robot operating independently or in coordination with others, resulting in faster, safer and more efficient processes. While significant progress has been made in this domain over the past decades, the forthcoming advancements are perhaps even more exciting and expected to have a profound impact on society.

Another interesting line of work would be related to the use of other MAS frameworks, such as Python Agent DEvelopment framework (PADE) [255] or Smart Python Agent Development Environment (SPADE) [256]. A Python-based MAS framework can provide access to a rich ecosystem of Machine Learning libraries that could be used to enhance agent capabilities.

Bibliography

[1] P. Iigo-Blasco, F. Diaz-Del-Rio, M. C. Romero-Ternero, D. Cagigas-Muiz, and S. Vicente-Diaz, “Robotics software frameworks for multi-agent robotic systems development,” *Robotics and Autonomous Systems*, vol. 60, no. 6, pp. 803–821, jun 2012, doi: 10.1016/j.robot.2012.02.004.

[2] A. Tiderko, “Multimaster_Fkie,” 2016, accessed: 2022-08-10. [Online]. Available: http://wiki.ros.org/multimaster_fkie/

[3] “OPC UA Client architecture,” accessed: 2023-07-05. [Online]. Available: https://reference.opcfoundation.org/Core/Part1/v104/docs/6.2

[4] “OPC UA Server Architecture,” accessed: 2023-07-05. [Online]. Available: https://reference.opcfoundation.org/Core/Part1/v104/docs/6.3

[5] K. Schwab, *La cuarta revolucion industrial*. Penguin Random House Grupo Editorial Espana, Nov. 2016, google-Books-ID: BRonDQAAQBAJ.

[6] “World Economic Forum 2020 - Fourth Industrial Revolution or Evolution?” accessed: 2023-06-22. [Online]. Available:

https://www.automation.com/en-us/articles/february-2020/world-economic-forum-2020-fourth-industrial-revolu

[7] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621–641, Jan. 2016, doi: 10.1016/j.cirp.2016.06.005.

[8] S. Zanero, "Cyber-Physical Systems," *Computer*, vol. 50, no. 4, pp. 14–16, Apr. 2017, doi: 10.1109/MC.2017.105.

[9] A. Kamagaew, J. Stenzel, A. Nettstrater, and M. Ten Hompel, "Concept of cellular transport systems in facility logistics," *ICARA 2011 - Proceedings of the 5th International Conference on Automation, Robotics and Applications*, pp. 40–45, 2011, doi: 10.1109/ICARA.2011.6144853.

[10] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE Spectrum*, vol. 45, no. 7, pp. 26–34, 2008, doi: 10.1109/MSPEC.2008.4547508.

[11] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: PROSA," *Computers in Industry*, vol. 37, no. 3, pp. 255–274, nov 1998, doi: 10.1016/S0166-3615(98)00102-X.

[12] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux, "Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution," *Computers in Industry*, vol. 66, pp. 99–111, Jan. 2015, doi: 10.1016/j.compind.2014.10.011.

[13] D. McFarlane, V. Giannikas, A. C. Wong, and M. Harrison, "Product intelligence in industrial control: Theory and practice," *Annual Reviews in Control*, vol. 37, no. 1, pp. 69–88, apr 2013, doi: 10.1016/j.arcontrol.2013.03.003.

[14] N. Matni, A. Tang, and J. C. Doyle, "A case study in network architecture tradeoffs," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. Santa Clara California: ACM, Jun. 2015, pp. 1–7, doi: 10.1145/2774993.2775011.

[15] J. Martin, O. Casquero, B. Fortes, and M. Marcos, "A generic multi-layer architecture based on ros-jade integration for autonomous transport vehicles," *Sensors*, vol. 19, no. 1, pp. 1–16, 2019, doi: 10.3390/s19010069.

[16] M. Lucas-Estañ, J. Maestre, B. Coll-Perales, J. Gozalvez, and I. Lluvia, "An experimental evaluation of redundancy in industrial wireless communications," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 1075–1078, doi: 10.1109/ETFA.2018.8502497.

[17] S. R. Fletcher, T. Johnson, T. Adlon, J. Larreina, P. Casla, L. Parigot, P. J. Alfaro, and M. del Mar Otero, "Adaptive automation assembly: Identifying system requirements for technical efficiency and worker satisfaction," *Computers & Industrial Engineering*, vol. 139, p. 105772, 2020, doi: 10.1016/j.cie.2019.03.036.

[18] J. Martin, A. Ansuategi, I. Maurtua, A. Gutierrez, D. Obregon, O. Casquero, and M. Marcos, "A Generic ROS-Based Control Architecture for Pest Inspection and Treatment in Greenhouses Using a Mobile Manipulator," *IEEE Access*, vol. 9, pp. 94 981–94 995, 2021, doi: 10.1109/ACCESS.2021.3093978.

[19] "Informe CODDII Industria 4.0," accessed: 2023-06-22. [Online]. Available: https://coddii.org/wp-content/uploads/2016/10/Informe-CODDII-Industria-4.0.pdf

[20] F. Blanco, J. M. Castro, R. Gayoso, and W. Santana, *Las claves de la Cuarta Revolución Industrial*. Libros de Cabecera, Aug. 2019, isbn: 978-84-949079-8-2.

[21] H. Kagermann, W. Wahlster, and J. Helbig, "Recommendations for implementing the strategic initiative INDUSTRIE 4.0," 2013, accessed: 2023-06-22. [Online]. Available: https://www.din.de/resource/blob/76902/e8cac883f42bf28536e7e8165993f1fd/recommendations-for-implementing-industry-4-0-data.pdf

[22] "Digital Transformation Monitor Germany: Industrie 4.0," accessed: 2023-06-22. [Online]. Available: https://ati.ec.europa.eu/sites/default/files/2020-06/DTM_Industrie%204.0_DE.pdf

[23] "What is Industrie 4.0?" accessed: 2023-06-22. [Online]. Available: https://www.plattform-i40.de/IP/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html

[24] P. Adolphs and U. Epple, "Status Report: Reference Architecture Model Industrie 4.0 (RAMI4.0)," accessed: 2023-06-22. [Online]. Available: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/januar/GMA_Status_Report___Archtitecture_Model_Industrie_4.0___RAMI_4.0_/GMA-Status-Report-RAMI-40-July-2015.pdf

[25] "IEC-62890:2020 | IEC Webstore," 2020, accessed: 2023-06-22. [Online]. Available: https://webstore.iec.ch/publication/30583

[26] "IEC-62264-6:2020 | IEC Webstore," 2020, accessed: 2023-06-22. [Online]. Available: https://webstore.iec.ch/publication/59706

[27] "IEC-61512-1:1997 | IEC Webstore," 1997, accessed: 2023-06-22. [Online]. Available: https://webstore.iec.ch/publication/5528

[28] “DIN SPEC 91345:2016-04, Referenzarchitekturmodell Industrie 4.0 (RAMI4.0),” Beuth Verlag GmbH, Tech. Rep., 2016, doi: 10.31030/2436156.

[29] X. Ye and S. H. Hong, “Toward Industry 4.0 Components: Insights Into and Implementation of Asset Administration Shells,” *IEEE Industrial Electronics Magazine*, vol. 13, no. 1, pp. 13–25, Mar. 2019, doi: 10.1109/MIE.2019.2893397.

[30] T. Miny, G. Stephan, T. Usländer, and J. Vialkowitsch, “Functional View of the Asset Administration Shell in an Industrie 4.0 System Environment,” 2021, accessed: 2023-06-22. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Functional-View.html

[31] “Industry IoT Consortium,” accessed: 2023-06-22. [Online]. Available: https://www.iiconsortium.org/

[32] M. Annunziata and P. Evans, “INDUSTRIAL INTERNET. A European perspective. Pushing the Boundaries of Minds and Machines,” Jun. 2013, accessed: 2023-06-22. [Online]. Available: https://www.ge.com/news/sites/default/files/5901.pdf?mod=article_inline

[33] “ISO/IEC/IEEE-42010:2022,” 2022. [Online]. Available: https://www.iso.org/standard/74393.html

[34] S.-W. Lin, E. Simmon, and S. Mellor, “The Industrial Internet Reference Architecture. Version 1.10,” 2022. [Online]. Available: https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/11/IIRA-v1.10.pdf

[35] S.-W. Lin, B. Murphy, E. Clauer, U. Loewen, R. Neubert, G. Bachmann, M. Pai, and M. Hankel, “Architecture Alignment and In-

teroperability. An Industrial Internet Consortium and Plattform Industrie 4.0 Joint Whitepaper," An Industrial Internet Consortium and Plattform Industrie 4.0 Joint Whitepaper, Tech. Rep., 2017.

[36] B. Boss, "Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0 An Industrial Internet Consortium and Plattform Industrie 4.0 Joint Whitepaper," 2020, accessed: 2023-06-22. [Online]. Available: https://www.iiconsortium.org/pdf/Digital-Twin-and-Asset-Administration-Shell-Concepts-and-Application-Joint-Whitepaper.pdf

[37] S. Wei, J. Hu, Y. Cheng, Y. Ma, and Y. Yu, "The essential elements of intelligent Manufacturing System Architecture," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017, pp. 1006–1011, doi: 10.1109/COASE.2017.8256234.

[38] "Industrial Valuechain Initiative," accessed: 2023-06-23. [Online]. Available: https://iv-i.org/en/en-top/

[39] Sino-German Industrie 4.0 / Intelligent Manufacturing Standardisation Sub-Working Group, "Alignment Report for Reference Architectural Model for Industrie 4.0/ Intelligent Manufacturing System Architecture," Apr. 2018, accessed: 2023-06-23. [Online]. Available: https://www.dke.de/resource/blob/1711304/2e4d62811e90ee7aad10eeb6fdeb33d2/alignment-report-for-reference-architectural-model-for-industrie-4-0-data.pdf

[40] "Scaling Fourth Industrial Revolution Technologies in Production," 2018, accessed: 2023-06-23. [Online]. Available: https://www3.weforum.org/docs/WEF_Technology_and_Innovation_The_Next_Economic_Growth_Engine.pdf

[41] S. Pfeiffer, "The Vision of "industrie 4.0 in the Making-a Case of Future Told, Tamed, and Traded," *NanoEthics*, vol. 11, no. 1, pp. 107–121, Apr. 2017, doi: 10.1007/s11569-016-0280-3.

[42] EFFRA, "Factories of the Future FoF 2020 Roadmap, EFFRA European Factories of the Future Research Association," pp. 1–89, 2012, accessed: 2018-01-26. [Online]. Available: www.effra.eu

[43] "Industria Conectada 4.0," accessed: 2023-06-23. [Online]. Available: https://www.industriaconectada40.gob.es/Paginas/index.aspx

[44] "Basque Digital Innovation HUB," accessed: 2023-06-23. [Online]. Available: https://bdih.spri.eus/es/

[45] M. Molnar, T. Frost, V. Lightner, and et. al., "2019 Manufacturing USA Strategic Plan," Advanced Manufacturing National Program Office, Tech. Rep. November, 2019. [Online]. Available: https://www.manufacturingusa.com/reports/manufacturing-usa-strategic-plan

[46] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621–641, jan 2016, doi: 10.1016/j.cirp.2016.06.005.

[47] "Plattform industrie 4.0, Aspects of the Research Roadmap in Application Scenarios I4.0," Federal Ministry for Economic Affairs and Energy (BMWi), Tech. Rep., 2016, accessed: 2018-07-14. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf?__blob=publicationFile&v=10

[48] "Horizon 2020 - work programme 2018-2020," European Commission, Tech. Rep., 2018. [Online]. Available: https://ec.europa.eu/research/participants/data/ref/h2020/wp/2018-2020/main/h2020-wp1820-leit-nmp_en.pdf

[49] C. Santos, A. Mehrsai, A. C. Barros, M. Araújo, and E. Ares, "Towards Industry 4.0: an overview of European strategic roadmaps," *Procedia Manufacturing*, vol. 13, pp. 972–979, 2017, doi: 10.1016/j.promfg.2017.09.093.

[50] S. Weyer, M. Schmitt, M. Ohmer, and D. Gorecky, "Towards industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems," *IFAC-PapersOnLine*, vol. 28, no. 3, pp. 579–584, 2015, doi: 10.1016/j.ifacol.2015.06.143.

[51] D. Park, Y. Hoshi, H. Mahajan, H. Kim, Z. Erickson, W. Rogers, and C. Kemp, "Active robot-assisted feeding with a general-purpose mobile manipulator: Design, evaluation, and lessons learned," *Robotics and Autonomous Systems*, vol. 124, 2019, doi: 10.1016/j.robot.2019.103344.

[52] L. Lu and J. T. Wen, "Baxter-on-wheels (BOW): An assistive mobile manipulator for mobility impaired individuals," in *Trends in Control and Decision-Making for Human-Robot Collaboration Systems*, 2017, pp. 41–63, doi: 10.1007/978-3-319-40533-9_3.

[53] L. Lu and J. Wen, "Human-directed coordinated control of an assistive mobile manipulator," *International Journal of Intelligent Robotics and Applications*, vol. 1, pp. 104–120, 2017, doi: 10.1007/s41315-016-0005-3.

[54] C. A. Smarr, T. L. Mitzner, J. M. Beer, A. Prakash, T. L. Chen, C. C. Kemp, and W. A. Rogers, "Domestic Robots for Older

Adults: Attitudes, Preferences, and Potential," *International Journal of Social Robotics*, vol. 6, no. 2, pp. 229–247, 2014, doi: 10.1007/s12369-013-0220-0.

[55] A. S. Kapusta, P. M. Grice, H. M. Clever, Y. Chitalia, D. Park, and C. C. Kemp, "A system for bedside assistance that integrates a robotic bed and a mobile manipulator," *PLoS ONE*, vol. 14, no. 10, p. e0221854, 2019, doi: 10.1371/journal.pone.0221854.

[56] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of the Research Platform of a Domestic Mobile Manipulator Utilized for International Competition and Field Test," in *IEEE International Conference on Intelligent Robots and Systems*, 2018, pp. 7675–7682, doi: 10.1109/IROS.2018.8593798.

[57] S. Elliott and M. Cakmak, "Robotic Cleaning Through Dirt Rearrangement Planning with Learned Transition Models," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1623–1630, doi: 10.1109/ICRA.2018.8460915.

[58] G. Pan, F. Yang, and M. Chen, "Kinematic Control of a Dual-Arm Humanoid Mobile Cooking Robot," in *Proceedings of the 12th International Convention on Rehabilitation Engineering and Assistive Technology*, ser. i-CREATe 2018. Midview City, SGP: Singapore Therapeutic, Assistive and Rehabilitative Technologies (START) Centre, 2018, pp. 308–311.

[59] H. Yu, L. Li, J. Chen, Y. Wang, Y. Wu, M. Li, H. Li, Z. Jiang, X. Liu, and T. Arai, "Mobile Robot Capable of Crossing Floors for Library Management," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2019, pp. 2540–2545, doi: 10.1109/ICMA.2019.8816274.

[60] A. Y. Mersha, R. De Kinkelder, and D. Bekke, “Affordable modular mobile manipulator for domestic applications,” in *Proceedings of the 2018 19th International Conference on Research and Education in Mechatronics, REM 2018*, 2018, pp. 141–146, doi: 10.1109/REM.2018.8421773.

[61] A. Mitrevski, A. Padalkar, M. Nguyen, and P. G. Plöger, “Lucy, Take the Noodle Box!: Domestic Object Manipulation Using Movement Primitives and Whole Body Motion,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11531 LNAI, pp. 189–200, doi: 10.1007/978-3-030-35699-6_15.

[62] C. Mucchiani, P. Cacchione, W. Torres, M. Johnson, and M. Yim, “Exploring Low-Cost Mobile Manipulation for Elder Care Within a Community Based Setting,” *Journal of Intelligent and Robotic Systems*, vol. 98, pp. 1–12, 2020, doi: 10.1007/s10846-019-01041-x.

[63] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, “Development of Human Support Robot as the research platform of a domestic mobile manipulator,” *ROBOMECH Journal*, vol. 6, no. 1, 2019, doi: 10.1186/s40648-019-0132-3.

[64] S. S. Srinivasa, D. Berenson, M. Cakmak, A. Collet, M. R. Dogar, A. D. Dragan, R. A. Knepper, T. Niemueller, K. Strabala, M. Vande Weghe, and J. Ziegler, “Herb 2.0: Lessons learned from developing a mobile manipulator for the home,” *Proceedings of the IEEE*, vol. 100, no. 8, pp. 2410–2428, 2012, doi: 10.1109/JPROC.2012.2200561.

[65] A. Khanna and S. Kaur, “Evolution of Internet of Things (IoT) and its significant impact in the field of Precision Agriculture,” *Computers and Electronics in Agriculture*, vol. 157, pp. 218–231, 2019, doi: 10.1016/j.compag.2018.12.039.

[66] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, “A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming,” *IEEE Access*, vol. 7, pp. 156 237–156 271, 2019, doi: 10.1109/ACCESS.2019.2949703.

[67] R. R. Shamshiri, C. Weltzien, I. A. Hameed, I. J. Yule, T. E. Grift, S. K. Balasundram, L. Pitonakova, D. Ahmad, and G. Chowdhary, “Research and development in agricultural robotics: A perspective of digital farming,” *International Journal of Agricultural and Biological Engineering*, vol. 11, no. 4, pp. 1–14, 2018, doi: 10.25165/ijabe.v11i4.4278.

[68] K. Hu, G. Coleman, S. Zeng, Z. Wang, and M. Walsh, “Graph weeds net: A graph-based deep learning method for weed recognition,” *Computers and Electronics in Agriculture*, vol. 174, p. 105520, 2020, doi: 10.1016/j.compag.2020.105520.

[69] “RHEA EU project,” accessed: 2023-07-02. [Online]. Available: http://www.rhea-project.eu/

[70] A. Ruckelshausen, P. Biber, M. Dorna, H. Gremmes, R. Klose, A. Linz, R. Rahe, R. Resch, M. Thiel, D. Trautz, and U. Weiss, “BoniRob: An autonomous field robot platform for individual plant phenotyping,” *in the 7th European Conference on Precision Agriculture, ECPA 2009*, vol. 9, pp. 841–847, jan 2009.

[71] S. Best, J. Baur, O. Ringdahl, R. Oberti, S. Evain, J. Bontsema, B. Debilde, H. Ulbrich, T. Hellström, M. Hočevar, A. Shapiro, M. Armada, W. Saeys, Y. Edan, W. Gauchel, J. Hemming, and E. Pekkeriet, “CROPS: Clever Robots for Crops,” *Engineering & Technology Reference*, 2015, doi: 10.1049/etr.2015.0015.

[72] X. Chen, “Statistical multipath model comparative analysis of different GNSS orbits in static urban canyon environment,” *Advances*

in Space Research, vol. 62, no. 5, pp. 1034–1048, 2018, doi: 10.1016/j.asr.2018.06.005.

[73] K. G. Fue, W. M. Porter, E. M. Barnes, and G. C. Rains, "An Extensive Review of Mobile Agricultural Robotics for Field Operations: Focus on Cotton Harvesting," *AgriEngineering*, vol. 2, no. 1, pp. 150–174, 2020, doi: 10.3390/agriengineering2010010.

[74] L. L. Wang, B. Zhao, J. W. Fan, X. A. Hu, S. Wei, Y. S. Li, Q. B. Zhou, and C. F. Wei, "Development of a tomato harvesting robot used in greenhouse," *International Journal of Agricultural and Biological Engineering*, vol. 10, no. 4, pp. 140–149, jan 2017, doi: 10.25165/j.ijabe.20171004.3204.

[75] B. Arad, J. Balendonck, R. Barth, O. Ben-Shahar, Y. Edan, T. Hellström, J. Hemming, P. Kurtser, O. Ringdahl, T. Tielen, and B. van Tuijl, "Development of a sweet pepper harvesting robot," *Journal of Field Robotics*, vol. 37, no. 6, pp. 1027–1039, 2020, doi: 10.1002/rob.21937.

[76] Q. Feng, W. Zou, P. Fan, C. Zhang, and X. Wang, "Design and test of robotic harvesting system for cherry tomato," *International Journal of Agricultural and Biological Engineering*, vol. 11, no. 1, pp. 96–100, jan 2018, doi: 10.25165/j.ijabe.20181101.2853.

[77] D. Obregón, R. Arnau, M. Campo-Cossio, J. G. Arroyo-Parras, M. Pattinson, S. Tiwari, I. Lluvia, O. Rey, J. Verschoore, L. Lenza, and J. Reyes, "Precise Positioning and Heading for Autonomous Scouting Robots in a Harsh Environment," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11487 LNCS, pp. 82–96, doi: 10.1007/978-3-030-19651-6_9.

[78] T. Hellström and O. Ringdahl, "A software framework for agricultural and forestry robots," *Industrial Robot*, vol. 40, no. 1, pp. 20–26, jan 2013, doi: 10.1108/01439911311294228.

[79] K. Jensen, M. Larsen, S. H. Nielsen, L. B. Larsen, K. S. Olsen, and R. N. Jørgensen, "Towards an open software platform for field robots in precision agriculture," *Robotics*, vol. 3, no. 2, pp. 207–234, jun 2014, doi: 10.3390/robotics3020207.

[80] Y. Zhang, Y. Li, H. Zhang, Y. Wang, Z. Wang, Y. Ye, Y. Yue, N. Guo, W. Gao, H. Chen, and S. Zhang, "Earthshaker: A mobile rescue robot for emergencies and disasters through teleoperation and autonomous navigation," *Journal of University of Science and Technology of China*, vol. 53, no. 1, p. 4, 2023, doi: 10.52396/JUSTC-2022-0066.

[81] I. Kruijff-Korbayová, L. Freda, M. Gianni, V. Ntouskos, V. Hlaváč, V. Kubelka, E. Zimmermann, H. Surmann, K. Dulic, W. Rottner, and E. Gissi, "Deployment of ground and aerial robots in earthquake-struck amatrice in italy (brief report)," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2016, pp. 278–279, doi: 10.1109/SSRR.2016.7784314.

[82] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma, "Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013, doi: 10.1002/rob.21439.

[83] I. D. Miller, A. Cohen, A. Kulkarni, J. Laney, C. J. Taylor, V. Kumar, F. Cladera, A. Cowley, S. S. Shivakumar, E. S. Lee, L. Jarin-Lipschitz, A. Bhat, N. Rodrigues, and A. Zhou, "Mine Tunnel

Exploration Using Multiple Quadrupedal Robots," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2840–2847, 2020, doi: 10.1109/LRA.2020.2972872.

[84] P. Koch, H. Engelhardt, S. May, M. Schmidpeter, J. Ziegler, and A. Nuchter, "Signed Distance Based Reconstruction for Exploration and Change Detection in Underground Mining Disaster Prevention," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 360–365, doi: 10.1109/SSRR50563.2020.9292618.

[85] N. Castaman, E. Tosello, M. Antonello, N. Bagarello, S. Gandin, M. Carraro, M. Munaro, R. Bortoletto, S. Ghidoni, E. Menegatti, and E. Pagello, "RUR53: an unmanned ground vehicle for navigation, recognition, and manipulation," *Advanced Robotics*, vol. 35, no. 1, pp. 1–18, 2021, doi: 10.1080/01691864.2020.1833752.

[86] L. Lu and J. T. Wen, "Human-directed robot motion/-force control for contact tasks in unstructured environments," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, 2015, pp. 1165–1170, doi: 10.1109/CoASE.2015.7294255.

[87] R. Mendonca, M. M. Marques, F. Marques, A. Lourenço, E. Pinto, P. Santana, F. Coito, V. Lobo, and J. Barata, "A cooperative multi-robot team for the surveillance of shipwreck survivors at sea," in *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*, sep 2016, pp. 1–6, doi: 10.1109/OCEANS.2016.7761074.

[88] M. Sereinig, W. Werth, and L. M. Faller, "A review of the challenges in mobile manipulation: systems design and RoboCup challenges: Recent developments with a special focus on the RoboCup," *Elektrotechnik und Informationstechnik*, vol.

137, no. 6, pp. 297–308, oct 2020, doi: 10.1007/s00502-020-00823-8.

[89] R. Memmesheimer, D. Müller, I. Kramer, Y. Wettengel, T. Evers, L. Buchhold, P. Schmidt, N. Schmidt, I. Germann, M. Mints, G. Rettler, C. Korbach, R. Bartsch, I. Kuhlmann, T. Weiland, and D. Paulus, "RoboCup@Home Open Platform League 2019 - homer@UniKoblenz (Germany)," 2019, doi: 10.13140/RG.2.2.15059.12328.

[90] A. Dizet, C. Bono, A. Legeleux, M. Neau, and C. Buche, "RoboCup@Home Education 2020 Best Performance: RoboBreizh, a modular approach," 2021, accessed: 2022-12-05. [Online]. Available: https://arxiv.org/abs/2107.02978

[91] J. Ziegler, J. Gleichauf, and C. Pfitzner, "RoboCup Rescue 2018 Team Description Paper AutonOHM," in *The robocup rescue 2018 TDP collection*, 06 2018.

[92] S. Zhu, H. Zhang, H. Lu, J. Xiao, C. Shi, C. Cheng, W. Deng, and X. Chen, "RoboCup Rescue 2019 Team Description Paper," pp. 1–6, 2019, doi: 10.13140/RG.2.2.30334.33601/1.

[93] W. Houtman, C. M. Kengen, P. H. van Lith, R. H. ten Berge, J. J. Kon, K. J. Meessen, M. A. Haverlag, Y. G. Douven, F. B. Schoenmakers, D. J. Bruijnen, W. H. Aangenent, J. J. Olthuis, M. Dolatabadi, S. T. Kempers, M. C. Schouten, R. M. Beumer, W. J. Kuijpers, A. A. Kokkelmans, H. C. van de Loo, and M. J. van de Molengraft, "Tech United Eindhoven Middle-Size League Winner 2019," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11531 LNAI, pp. 517–528, doi: 10.1007/978-3-030-35699-6_42.

[94] J. J. Olthuis, N. B. van der Meer, S. T. Kempers, C. A. van Hoof, R. M. Beumer, W. J. Kuijpers, A. A. Kokkelmans, W. Houtman, J. J. van Eijck, J. J. Kon, A. T. Peijnenburg, and M. J. van de Molengraft, "Vision-Based Machine Learning in Robot Soccer," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2022, vol. 13132 LNAI, pp. 327–339, doi: 10.1007/978-3-030-98682-7_27.

[95] N. Eltester, A. Ferrein, and S. Schiffer, "A Smart Factory Setup based on the RoboCup Logistics League," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, 2020, doi: 10.1109/ICPS48405.2020.9274766.

[96] P. Kohout, M. De Bortoli, J. Ludwiger, T. Ulz, and G. Steinbauer, "A multi-robot architecture for the RoboCup Logistics League," *Elektrotechnik und Informationstechnik*, vol. 137, no. 6, pp. 291–296, 2020, doi: 10.1007/s00502-020-00826-5.

[97] J. Martin, H. Engelhardt, M. Masannek, T. Scholz, K. Gillmann, and B. Schadde, "RoboCup@Work 2018 Team AutonOHM," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11374 LNAI. Cham: Springer International Publishing, 2019, pp. 500–511, doi: 10.1007/978-3-030-27544-0_41.

[98] T. Carstensen, J. Carstensen, A. Dick, S. Falkenhain, J. Hübner, R. Kammel, A. Wentz, S. Aden, J. Friederichs, and J. Kotlarski, "Staying on top at RoboCup@Work 2016," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9776 LNAI, 2017, pp. 601–612, doi: 10.1007/978-3-319-68792-6_50.

[99] "RoboCup Industrial @Work League," accessed: 2023-06-10. [Online]. Available: https://atwork.robocup.org/

[100] "European Robotics League- Industrial robots," accessed: 2023-06-22. [Online]. Available: https://eu-robotics.net/

[101] M. Shneier and R. Bostelman, "Literature Review of Mobile Robots for Manufacturing, NIST Interagency/Internal Report (NISTIR)," in *National Institute of Standards and Technology*, 2015, https://doi.org/10.6028/NIST.IR.8022.

[102] I. Karabegović, E. Karabegović, M. Mahmić, and E. Husak, "The application of service robots for logistics in manufacturing processes," *Advances in Production Engineering and Management*, vol. 10, no. 4, pp. 185–194, 2015, doi: 10.14743/apem2015.4.201.

[103] D. Pavlichenko, G. Martin Garcia, S.-Y. Koo, and S. Behnke, "Kittingbot: A mobile manipulation robot for collaborative kitting in automotive logistics: Proceedings of the 15th international conference ias-15," pp. 849–864, 01 2019, doi: 10.1007/978-3-030-01370-7_66.

[104] O. Madsen, "Robotics-enabled Logistics and Assistive Services for the Transformable Factory of the Future (TAPAS)," 2009, accessed: 2023-07-20. [Online]. Available: http://www.tapas-project.eu/

[105] A. Dömel, S. Kriegel, M. Kassecker, M. Brucker, T. Bodenmüller, and M. Suppa, "Toward fully autonomous mobile manipulation for industrial environments," *International Journal of Advanced Robotic Systems*, vol. 14, 07 2017, doi: 10.1177/1729881417718588.

[106] P. J. Koch, M. K. van Amstel, P. Dębska, M. A. Thormann, A. J. Tetzlaff, S. Bøgh, and D. Chrysostomou, “A Skill-based Robot Co-worker for Industrial Maintenance Tasks,” *Procedia Manufacturing*, vol. 11, pp. 83–90, 2017, doi: 10.1016/j.promfg.2017.07.141.

[107] K. Furmans, F. Schönung, and K. R. Gue, “Plug-and-work material handling systems,” *Progress in Material Handling Research*, no. 1, pp. 1–11, 2010.

[108] K. Furmans, C. Nobbe, and M. Schwab, “Future of Material Handling - modular, flexible and efficient,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[109] S. Mayer, “Development of a completely decentralized control system for modular continuous conveyors,” Ph.D. dissertation, Universitätsverlag Karlsruhe, 2009, DOI: 10.5445/KSP/1000011463.

[110] A. Wolf, “A review on robotics in life science automation,” *AIS 2019 : 14th International Symposium on Applied Informatics and Related Areas organized in the frame of Hungarian Science Festival 2019 by Obuda University*, Jan. 2019.

[111] J. Martin, T. Fink, S. May, C. Ochs, and I. Cabanes, “An Autonomous Transport Vehicle in an existing manufacturing facility with focus on the docking maneuver task,” in *2017 3rd International Conference on Control, Automation and Robotics, ICCAR 2017*, 2017, pp. 365–370, doi: 10.1109/ICCAR.2017.7942719.

[112] “evocortex,” accessed: 2023-06-08. [Online]. Available: https://evocortex.org/

[113] “symovo,” accessed: 2023-06-08. [Online]. Available: https://www.symovo.de/

[114] "Omron Adept Technology," accessed: 2018-07-22. [Online]. Available: https://www.adept.com/

[115] "Aethon," accessed: 2023-06-08. [Online]. Available: https://aethon.com/

[116] "I am robotics," accessed: 2018-07-22. [Online]. Available: https://www.iamrobotics.com/

[117] "Locus Robotics," accessed: 2018-07-22. [Online]. Available: https://www.locusrobotics.com/

[118] "Mobilee Industrial Robots," accessed: 2018-07-22. [Online]. Available: http://www.mobile-industrial-robots.com/

[119] R. Indigo, "Fetch Robotics," 2016, accessed: 2023-06-08. [Online]. Available: https://fetchrobotics.com/

[120] "magazino," accessed: 2023-07-05. [Online]. Available: https://www.magazino.eu/

[121] "AMR arculee," accessed: 2023-06-08. [Online]. Available: https://www.arculus.de/product/

[122] M. Yang, E. Yang, R. C. Zante, M. Post, and X. Liu, "Collaborative mobile industrial manipulator: A review of system architecture and applications," in *2019 25th International Conference on Automation and Computing (ICAC)*, 2019, pp. 1–6, doi: 10.23919/IConAC.2019.8895183.

[123] S. Thakar, S. Srinivasan, S. Al-Hussaini, P. Bhatt, P. Rajendran, Y. J. Yoon, N. Dhanaraj, R. Malhan, M. Schmid, V. Krovi, and S. Gupta, "A Survey of Wheeled Mobile Manipulation: A Decision Making Perspective," *Journal of Mechanisms and Robotics*, vol. 15, pp. 1–38, 2022, doi: 10.1115/1.4054611.

[124] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your Robot have Skills?" in *Proceedings of the 43rd International Symposium on Robotics*, 2012.

[125] F. Rovida, D. Wuthier, B. Grossmann, M. Fumagalli, and V. Krüger, "Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5964–5971, doi: 10.1109/IROS.2018.8594319.

[126] F. Rovida, M. Crosby, D. Holz, A. Polydoros, B. Großmann, R. Petrick, and V. Krueger, "SkiROS—A skill-based robot control platform on top of ROS," in *Studies in Computational Intelligence*, 2017, pp. 121–160, doi: 10.1007/978-3-319-54927-9_4.

[127] F. Rovida, D. Chrysostomou, C. Schou, S. Bøgh, O. Madsen, V. Krüger, R. S. Andersen, M. R. Pedersen, B. Grossmann, and J. S. Damgaard, "SkiROS: A four tiered architecture for task-level programming of industrial mobile manipulators," 2014.

[128] J. Saukkoriipi, T. Heikkilä, J. Ahola, T. Seppälä, and P. Isto, "Programming and control for skill-based robots," *Open Engineering*, vol. 10, pp. 368–376, 2020, doi: 10.1515/eng-2020-0037.

[129] J. Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty," *The International Journal of Robotics Research*, vol. 26, pp. 433–455, 2007, doi: 10.1177/027836490707809107.

[130] J. Huckaby and H. Christensen, "A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics," *AAAI Workshop - Technical Report*, pp. 94–101, 2012.

[131] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, “A framework for end-user instruction of a robot assistant for manufacturing,” *Proceedings - IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6167–6174, 2015, doi: 10.1109/ICRA.2015.7140065.

[132] M. Crosby, R. Petrick, F. Rovida, and V. Krueger, “Integrating Mission and Task Planning in an Industrial Robotics Framework,” *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, pp. 471–479, 2017, doi: 10.1609/icaps.v27i1.13857.

[133] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, “CoSTAR: Instructing collaborative robots with behavior trees and vision,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 564–571, doi: 10.1109/ICRA.2017.7989070.

[134] B. Ridge, T. Gaspar, and A. Ude, “Rapid state machine assembly for modular robot control using meta-scripting, templating and code generation,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017, pp. 661–668, doi: 10.1109/HUMANOIDS.2017.8246943.

[135] S. G. Brunner, P. Lehner, M. J. Schuster, S. Riedel, R. Belder, D. Leidner, A. Wedler, M. Beetz, and F. Stulp, “Design, Execution, and Postmortem Analysis of Prolonged Autonomous Robot Operations,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1056–1063, 2018, doi: 10.1109/LRA.2018.2794580.

[136] S. Hart, A. H. Quispe, M. W. Lanighan, and S. Gee, “Generalized Affordance Templates for Mobile Manipulation,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2022, pp. 6240–6246, doi: 10.1109/ICRA46639.2022.9812082.

[137] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp, "ROS commander (ROSCo): Behavior creation for home robots," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 467–474, doi: 10.1109/ICRA.2013.6630616.

[138] S. S. Yang, X. Mao, S. S. Yang, and Z. Liu, "Towards a hybrid software architecture and multi-agent approach for autonomous robot software," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, pp. 1–15, jul 2017, doi: 10.1177/1729881417716088.

[139] Z. Liu, X. Mao, and S. Yang, "AutoRobot: A multi-agent software framework for autonomous robots," *IEICE Transactions on Information and Systems*, vol. E101D, no. 7, pp. 1880–1893, jul 2018, doi: 10.1587/transinf.2017EDP7382.

[140] M. Li, Z. Cai, X. Yi, Z. Wang, Y. Wang, Y. Zhang, and X. Yang, "ALLIANCE-ROS: A software architecture on ros for fault-tolerant cooperative multi-robot systems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9810 LNCS. Cham: Springer International Publishing, 2016, pp. 233–242, doi: 10.1007/978-3-319-42911-3_19.

[141] A. W. Colombo, R. Schoop, and R. Neubert, "An agent-based intelligent control platform for industrial holonic manufacturing systems," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 1, pp. 322–337, 2006, doi: 10.1109/TIE.2005.862210.

[142] L. A. Cruz Salazar, D. Ryashentseva, A. Lüder, and B. Vogel-Heuser, "Cyber-physical production systems architecture based on multi-agent's design pattern—comparison of selected approaches

mapping four agent patterns," *The International Journal of Advanced Manufacturing Technology*, vol. 105, no. 9, pp. 4005–4034, Dec. 2019, doi: 10.1007/s00170-019-03800-4.

[143] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, Jan. 2015, doi: 10.1016/j.mfglet.2014.12.001.

[144] V. Mařík and D. McFarlane, "Industrial adoption of agent-based technologies," *IEEE Intelligent Systems*, vol. 20, no. 1, pp. 27–35, jan 2005, doi: 10.1109/MIS.2005.11.

[145] D. Sinha and R. Roy, "Reviewing Cyber-Physical System as a Part of Smart Factory in Industry 4.0," *IEEE Engineering Management Review*, vol. 48, no. 2, pp. 103–117, 2020, doi: 10.1109/EMR.2020.2992606.

[146] P. Leitão, A. W. Colombo, and S. Karnouskos, "Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges," *Computers in Industry*, vol. 81, pp. 11–25, 2016, doi: 10.1016/j.compind.2015.08.004.

[147] D. Trentesaux, T. Borangiu, and A. Thomas, "Emerging ICT concepts for smart, safe and sustainable industrial systems," *Computers in Industry*, vol. 81, pp. 1–10, Sep. 2016, doi: 10.1016/j.compind.2016.05.001.

[148] M. Moghaddam, M. N. Cadavid, C. R. Kenley, and A. V. Deshmukh, "Reference architectures for smart manufacturing: A critical review," *Journal of Manufacturing Systems*, vol. 49, pp. 215–225, Oct. 2018, doi: 10.1016/j.jmsy.2018.10.006.

[149] E. Trunzer, A. Calà, P. Leitão, M. Gepp, J. Kinghorst, A. Lüder, H. Schauerte, M. Reifferscheid, and B. Vogel-Heuser, “System architectures for Industrie 4.0 applications,” *Production Engineering*, vol. 13, no. 3, pp. 247–257, Jun. 2019, doi: 10.1007/s11740-019-00902-6.

[150] P. Leitão and J. Barbosa, “Building a Robotic Cyber-Physical Production Component,” in *Service Orientation in Holonic and Multi-Agent Manufacturing*, ser. Studies in Computational Intelligence, T. Borangiu, D. Trentesaux, A. Thomas, and D. McFarlane, Eds. Cham: Springer International Publishing, 2016, pp. 295–305, doi: 10.1007/978-3-319-30337-6_27.

[151] M. A. Pisching, M. A. O. Pessoa, F. Junqueira, D. J. dos Santos Filho, and P. E. Miyagi, “An architecture based on RAMI 4.0 to discover equipment to process operations required by products,” *Computers & Industrial Engineering*, vol. 125, pp. 574–591, Nov. 2018, doi: 10.1016/j.cie.2017.12.029.

[152] M. Yli-Ojanperä, S. Sierla, N. Papakonstantinou, and V. Vyatkin, “Adapting an agile manufacturing concept to the reference architecture model industry 4.0: A survey and case study,” *Journal of Industrial Information Integration*, vol. 15, pp. 147–160, Sep. 2019, doi: 10.1016/j.jii.2018.12.002.

[153] G. Zambrano Rey, C. Pach, N. Aissani, A. Bekrar, T. Berger, and D. Trentesaux, “The control of myopic behavior in semi-heterarchical production systems: A holonic framework,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 2, pp. 800–817, Feb. 2013, doi: 10.1016/j.engappai.2012.08.011.

[154] J. C. Bendul and H. Blunck, “The design space of production planning and control for industry 4.0,” *Computers in Industry*, vol. 105, pp. 260–272, Feb. 2019, doi: 10.1016/j.compind.2018.10.010.

[155] T. K. Jana, B. Bairagi, S. Paul, B. Sarkar, and J. Saha, "Dynamic schedule execution in an agent based holonic manufacturing system," *Journal of Manufacturing Systems*, vol. 32, no. 4, pp. 801–816, Oct. 2013, doi: 10.1016/j.jmsy.2013.07.004.

[156] H. Tang, D. Li, S. Wang, and Z. Dong, "CASOA: An Architecture for Agent-Based Manufacturing System in the Context of Industry 4.0," *IEEE Access*, vol. 6, pp. 12 746–12 754, 2018, doi: 10.1109/ACCESS.2017.2758160.

[157] T. Munkelt and M. Krockert, "AGENT-BASED SELF-ORGANIZATION VERSUS CENTRAL PRODUCTION PLANNING," in *2018 Winter Simulation Conference (WSC)*. Gothenburg, Sweden: IEEE, Dec. 2018, pp. 3241–3251, doi: 10.1109/WSC.2018.8632305.

[158] T. Borangiu, D. Trentesaux, A. Thomas, P. Leitão, and J. Barata, "Digital transformation of manufacturing through cloud services and resource virtualization," *Computers in Industry*, vol. 108, pp. 150–162, Jun. 2019, doi: 10.1016/j.compind.2019.01.006.

[159] "Eclipse Arrowhead – Eclipse Arrowhead framework and implementation platform," accessed: 2023-06-23. [Online]. Available: https://arrowhead.eu/

[160] Y. Lu and F. Ju, "Smart Manufacturing Systems based on Cyber-physical Manufacturing Services (CPMS)," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 883–15 889, Jul. 2017, doi: 10.1016/j.ifacol.2017.08.2349.

[161] P. Leitão and F. Restivo, "ADACOR: A holonic architecture for agile and adaptive manufacturing control," *Computers in Industry*, vol. 57, no. 2, pp. 121–130, Feb. 2006, doi: 10.1016/j.compind.2005.05.005.

[162] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, “Reference architecture for holonic manufacturing systems: PROSA,” *Computers in Industry*, vol. 37, no. 3, pp. 255–274, Nov. 1998, doi: 10.1016/S0166-3615(98)00102-X.

[163] S. Karnouskos, P. Leitao, L. Ribeiro, and A. W. Colombo, “Industrial Agents as a Key Enabler for Realizing Industrial Cyber-Physical Systems: Multiagent Systems Entering Industry 4.0,” *IEEE Industrial Electronics Magazine*, vol. 14, no. 3, pp. 18–32, Sep. 2020, doi: 10.1109/MIE.2019.2962225.

[164] P. Leitão, J. Queiroz, and L. Sakurada, “Collective Intelligence in Self-Organized Industrial Cyber-Physical Systems,” *Electronics*, vol. 11, no. 19, p. 3213, Jan. 2022, doi: 10.3390/electronics11193213.

[165] X. Ye, S. H. Hong, W. S. Song, Y. C. Kim, and X. Zhang, “An Industry 4.0 Asset Administration Shell-Enabled Digital Solution for Robot-Based Manufacturing Systems,” *IEEE Access*, vol. 9, pp. 154 448–154 459, 2021, doi: 10.1109/ACCESS.2021.3128580.

[166] A. D. Neal, R. G. Sharpe, K. van Lopik, J. Tribe, P. Goodall, H. Lugo, D. Segura-Velandia, P. Conway, L. M. Jackson, T. W. Jackson, and A. A. West, “The potential of industry 4.0 Cyber Physical System to improve quality assurance: An automotive case study for wash monitoring of returnable transit items,” *CIRP Journal of Manufacturing Science and Technology*, vol. 32, pp. 461–475, Jan. 2021, doi: 10.1016/j.cirpj.2020.07.002.

[167] P. Marcon, C. Diedrich, F. Zezulka, T. Schröder, A. Belyaev, J. Arm, T. Benesl, Z. Bradac, and I. Vesely, “The Asset Administration Shell of Operator in the Platform of Industry 4.0,” in *2018 18th International Conference on Mechatronics - Mechatronika (ME)*, Dec. 2018, pp. 1–5.

[168] S. Cavalieri and M. G. Salafia, “Asset Administration Shell for PLC Representation Based on IEC 61131–3,” *IEEE Access*, vol. 8, pp. 142 606–142 621, 2020, doi: 10.1109/ACCESS.2020.3013890.

[169] J. de las Morenas, A. García-Higuera, and P. García-Ansola, “Shop Floor Control: A Physical Agents Approach for PLC-Controlled Systems,” *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2417–2427, Oct. 2017, doi: 10.1109/TII.2017.2720696.

[170] F. Pethig, O. Niggemann, and A. Walter, “Towards Industrie 4.0 compliant configuration of condition monitoring services,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, Jul. 2017, pp. 271–276, doi: 10.1109/INDIN.2017.8104783.

[171] W. Dai, V. N. Dubinin, J. H. Christensen, V. Vyatkin, and X. Guan, “Toward Self-Manageable and Adaptive Industrial Cyber-Physical Systems With Knowledge-Driven Autonomic Service Management,” *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 725–736, Apr. 2017, doi: 10.1109/TII.2016.2595401.

[172] M. Wenger, A. Zoitl, and T. Müller, “Connecting PLCs With Their Asset Administration Shell For Automatic Device Configuration,” in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, Jul. 2018, pp. 74–79, doi: 10.1109/INDIN.2018.8472022.

[173] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, and O. Madsen, “Skill-based instruction of collaborative robots in industrial settings,” *Robotics and Computer-Integrated Manufacturing*, vol. 53, no. October 2018, pp. 72–80, 2018, doi: 10.1016/j.rcim.2018.03.008.

[174] J. D. Contreras, J. I. Garcia, and J. D. Diaz, “Developing of Industry 4.0 Applications,” *International Journal of Online and Biomed-*

ical Engineering (iJOE), vol. 13, no. 10, pp. 30–47, Nov. 2017, doi: 10.3991/ijoe.v13i10.7331.

[175] "IEEE Recommended Practice for Industrial Agents: Integration of Software Agents and Low-Level Automation Functions," *IEEE Std 2660.1-2020*, pp. 1–43, Jan. 2021, doi: 10.1109/IEEESTD.2021.9340089.

[176] L. Sakurada, P. ã, and F. D. la Prieta, "Agent-Based Asset Administration Shell Approach for Digitizing Industrial Assets," *IFAC-PapersOnLine*, vol. 55, no. 2, pp. 193–198, Jan. 2022, doi: 10.1016/j.ifacol.2022.04.192.

[177] P. Leitão, J. Barbosa, A. Pereira, J. Barata, and A. W. Colombo, "Specification of the PERFoRM architecture for the seamless production system reconfiguration," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2016, pp. 5729–5734, doi: 10.1109/IECON.2016.7793007.

[178] P. Leitão, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart Agents in Industrial Cyber–Physical Systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1086–1101, May 2016, doi: 10.1109/JPROC.2016.2521931.

[179] R. S. Peres, A. Dionisio Rocha, P. Leitao, and J. Barata, "IDARTS – Towards intelligent data analysis and real-time supervision for industry 4.0," *Computers in Industry*, vol. 101, pp. 138–146, Oct. 2018, doi: 10.1016/j.compind.2018.07.004.

[180] B. Vogel–Heuser and L. Ribeiro, "Bringing automated intelligence to cyber-physical production systems in factory automation," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 347–352.

[181] M. Bennulf, F. Danielsson, B. Svensson, and B. Lennartson, "Goal-Oriented Process Plans in a Multiagent System for Plug & Produce," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2411–2421, Apr. 2021, doi: 10.1109/TII.2020.2994032.

[182] I. Kovalenko, D. Tilbury, and K. Barton, "The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems," *Control Engineering Practice*, vol. 86, pp. 105–117, May 2019, doi: 10.1016/j.conengprac.2019.03.009.

[183] I. Kovalenko, D. Ryashentseva, B. Vogel-Heuser, D. Tilbury, and K. Barton, "Dynamic Resource Task Negotiation to Enable Product Agent Exploration in Multi-Agent Manufacturing Systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2854–2861, Jul. 2019, doi: 10.1109/LRA.2019.2921947.

[184] O. Shehory and A. Sturm, *Agent-oriented software engineering: Reflections on architectures, methodologies, languages, and frameworks*, 2014, vol. 9783642544, doi: 10.1007/978-3-642-54432-3.

[185] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004, doi: 10.1177/0278364904045564.

[186] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Studies in Computational Intelligence*, A. Koubâa and J. Martínez-de Dios, Eds. Cham: Springer International Publishing, 2015, vol. 604, pp. 31–51, doi: 10.1007/978-3-319-18299-5_2.

[187] R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Trans-*

actions on Computers, vol. C-29, no. 12, pp. 1104–1113, 1980, doi: 10.1109/TC.1980.1675516.

[188] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, “Intelligent Manufacturing in the Context of Industry 4.0: A Review,” *Engineering*, vol. 3, no. 5, pp. 616–630, oct 2017, doi: 10.1016/J.ENG.2017.05.015.

[189] M. K. Adeyeri, K. Mpofu, O. T. Adenuga, T. Adenuga Olukorede, and O. T. Adenuga, “Integration of agent technology into manufacturing enterprise: A review and platform for industry 4.0,” in *IEOM 2015 - 5th International Conference on Industrial Engineering and Operations Management, Proceeding*. IEEE, mar 2015, pp. 1–10, doi: 10.1109/IEOM.2015.7093910.

[190] J. Y. Zhao, Y. J. Wang, X. Xi, and G. D. Wu, “Simulation of steel production logistics system based on multi-agents,” *International Journal of Simulation Modelling*, vol. 16, no. 1, pp. 167–175, 2017, doi: 10.2507/IJSIMM16(1)CO4.

[191] A. Orebäck and H. I. Christensen, “Evaluation of architectures for mobile robotics,” *Autonomous Robots*, vol. 14, no. 1, pp. 33–49, jan 2003, doi: 10.1023/A:1020975419546.

[192] M. Mouad, L. Adouane, P. Schmitt, D. Khadraoui, and P. Martinet, “MAS2CAR architecture: Multi-agent system to control and coordinate teAmworking Robots,” in *ICINCO 2011 - Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics*, vol. 2, 2011, pp. 451–456, doi: 10.5220/0003649904510456.

[193] R. Rasch, A. Pörtner, M. Hoffmann, and M. König, “A decoupled three-layered architecture for service robotics in intelligent environments,” in *Proceedings of the 1st Workshop on Embodied*

Interaction with Smart Environments, EISE 2016, ser. EISE '16. ACM, 2016, pp. 1:1—-1:8, doi: 10.1145/3008028.3008032.

[194] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck, "Towards component-based robotics," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 3567–3572, doi: 10.1109/IROS.2005.1545523.

[195] B. Gerkey, R. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," *Proceedings of the International Conference on Advanced Robotics*, 08 2003.

[196] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 043–048, 2006, doi: 10.5772/5761.

[197] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, and T. Asfour, "The robot software framework ArmarX," *IT - Information Technology*, vol. 57, no. 2, pp. 99–111, 2015, doi: 10.1515/itit-2014-1066.

[198] "OpenRTM-aist Middelware," accessed: 2022-11-16. [Online]. Available: https://openrtm.org/openrtm/en

[199] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source Robot Operating System," *ICRA Workshop on Open Source Software*, vol. 3, pp. 1–6, 2009.

[200] O. Michel, "Cyberbotics Ltd. webots™: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004, doi: 10.5772/5618.

[201] E. Rohmer, S. P. Singh, and M. Freese, “V-REP: A versatile and scalable robot simulation framework,” in *IEEE International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326, doi: 10.1109/IROS.2013.6696520.

[202] Gazebo, “Gazebo Simulator Official Site,” 2016, accessed: 2022-10-05. [Online]. Available: http://gazebosim.org/

[203] “open-rmf,” accessed: 2022-11-16. [Online]. Available: https://www.open-rmf.org/

[204] A. D. Friedrich, “Anwendbarkeit von Methoden und Werkzeugen des konventionellen Softwareengineering zur Modellierung und Programmierung von Steuerungssystemen,” in *Embedded Systems 2*. kassel university press, 2009, p. 158.

[205] T. Hannelius, M. Salmenperä, and S. Kuikka, “Roadmap to adopting OPC UA,” *IEEE International Conference on Industrial Informatics (INDIN)*, pp. 756–761, 2008, doi: 10.1109/INDIN.2008.4618203.

[206] “Ignition OPC UA implementation - Inductive automation,” accessed: 2023-06-08. [Online]. Available: https://inductiveautomation.com/ignition/modules/ignition-opc-ua

[207] “SIEMENS - OPC UA,” accessed: 2022-11-28. [Online]. Available: https://new.siemens.com/global/en/products/automation/industrial-communication/opc-ua.html

[208] Open62541, “List of Open Source OPC UA Implementations,” 2016, accessed: 2022-11-22. [Online]. Available: https://github.com/open62541/open62541/wiki/List-of-Open-Source-OPC-UA-Implementations

[209] "open62541," accessed: 2022-11-22. [Online]. Available: http://www.open62541.org/

[210] "ROS-OPCUA FreeOpcUa C++ implementation," accessed: 2022-11-23. [Online]. Available: http://wiki.ros.org/ros_opcua_impl_freeopcua?distro=kinetic

[211] "Agent Builder," accessed: 2023-06-23. [Online]. Available: https://www.agentbuilder.com/

[212] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE: A software framework for developing multi-agent applications. Lessons learned," *Information and Software Technology*, vol. 50, no. 1-2, pp. 10–21, jan 2008, doi: 10.1016/j.infsof.2007.10.008.

[213] H. Gautam, M. Sharma, and D. Yadav, "Java Agent Development Framework," *Journal of Advance Research in Computer Science & Engineering (ISSN: 2456-3552)*, vol. 1, no. 3, pp. 06–09, 2014, doi: 10.53555/nncse.v1i3.515.

[214] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007, vol. 8, doi: 10.1002/9780470061848.

[215] O. Gutknecht and J. Ferber, "The MadKit agent platform architecture," in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 1887, 2001, pp. 48–55, doi: 10.1007/3-540-47772-1_5.

[216] E. Colon, "Robotics Frameworks," in *Introduction to Modern Robotics II*, 2013, p. 20.

[217] E. Tsardoulias and P. Mitkas, "Robotic frameworks, architectures and middleware comparison," vol. abs/1711.06842, 2017.

[218] A. T. Khan, S. Li, D. Chen, and Y. Li, "Open-source projects for autonomous robotics and systems: A survey," *Filomat*, vol. 34, no. 15, pp. 4953–4966, 2020, doi: 10.2298/FIL2015953K.

[219] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, 01 2015, doi: 10.18564/jasss.2661.

[220] R. Leszczyna, "Evaluation of agent platforms," *JRC Scientific and Technical Reports*, 2008, accessed: 2023-07-20. [Online]. Available: https://publications.jrc.ec.europa.eu/repository/bitstream/JRC47224/reqno_jrc47224.pdf

[221] "The Need For Robotics Standards," accessed: 2022-11-28. [Online]. Available: https://www.theconstructsim.com/need-robotics-standards/

[222] "ROS Metrics," 2020, accessed: 2022-11-28. [Online]. Available: https://metrics.ros.org/packages_rosdistro.html

[223] "ROS - Industrial," accessed: 2022-11-28. [Online]. Available: https://rosindustrial.org/

[224] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with JADE," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1986 LNAI, pp. 89–103, 2001, doi: 10.1007/3-540-44631-1_7.

[225] G. Villarrubia, J. F. De Paz, J. Bajo, and J. M. Corchado, "Ambient agents: Embedded agents for remote control and monitoring using the PANGEA platform," *Sensors*, vol. 14, no. 8, pp. 13 955–13 979, 2014, doi: 10.3390/s140813955.

[226] A. Koubâa, M. F. Sriti, H. Bennaceur, A. Ammar, Y. Javed, M. Alajlan, N. Al-Elaiwi, M. Tounsi, and E. Shakshuki, "Coros: A multi-agent software architecture for cooperative and autonomous service robots," in *Studies in Computational Intelligence*, 2015, vol. 604, pp. 3–30, doi: 10.1007/978-3-319-18299-5_1.

[227] G. S. Broten, D. Mackay, S. P. Monckton, and J. Collier, "The robotics experience: Beyond components and middleware," *IEEE Robotics and Automation Magazine*, vol. 16, no. 1, pp. 46–54, 2009, doi: 10.1109/MRA.2008.931632.

[228] "ROS-OPCUA IEC 62541 python implementation," accessed: 2022-11-23. [Online]. Available: http://wiki.ros.org/ros_opcua_impl_python_opcua?distro=kinetic

[229] A. V. Sandita and C. I. Popirlan, "Developing A Multi-Agent System in JADE for Information Management in Educational Competence Domains," *Procedia Economics and Finance*, vol. 23, pp. 478–486, 2015, doi: 10.1016/s2212-5671(15)00404-9.

[230] J. Lange, F. Iwanitz, and T. J. Burke, *OPC: von data access bis unified architecture*. VDE, 2010.

[231] G. K. Kraetzschmar, N. Hochgeschwender, W. Nowak, F. Hegger, S. Schneider, R. Dwiputra, J. Berghofer, and R. Bischoff, "RoboCup@Work: Competing for the factory of the future," in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*. Springer, Cham, 2015, vol. 8992, pp. 171–182, doi: 10.1007/978-3-319-18615-3_14.

[232] "Gmapping, laser-based SLAM," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/gmapping

[233] "Navigation Stack," accessed: 2020-10-05. [Online]. Available: http://wiki.ros.org/navigation

[234] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, no. May, pp. 1322–1328, 1999, doi: 10.1109/robot.1999.772544.

[235] M. A. Fischler and R. C. Bolles, "Random sample consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981, doi: 10.1145/358669.358692.

[236] J. Martin, H. Engelhardt, T. Fink, M. Masannek, and T. Scholz, "RoboCup@Work Winners 2017 Team AutonOHM," in *RoboCup 2017: Robot World Cup XXI*. Springer International Publishing, 2018, pp. 498–508, doi: 10.1007/978-3-030-00308-1_41.

[237] H. Zhou and S. Sakane, "Robot Localization," pp. 1270–1276, 2005, accessed: 2022-04-20. [Online]. Available: http://wiki.ros.org/robot_localization

[238] "CRO-INSPECT," accessed: 2023-07-05. [Online]. Available: https://cordis.europa.eu/project/id/716935/es/

[239] "Segway RMPv3 drivers ROS wiki," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/Robots/RMPv3

[240] "Velodyne laser drivers ROS wiki," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/velodyne

[241] "Omron os32c laser drivers ROS wiki," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/omron_os32c_driver

[242] "Kuka lbr iiwa drivers ROS wiki," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/kuka_lbr_iiwa_support

[243] "UEye cameras IDS drivers ROS wiki," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/ueye_cam

[244] I. Lluvia, A. Ansuategi, C. Tubio, L. Susperregi, I. I. Maurtua, E. Lazkano, C. Tubío, L. Susperregi, I. I. Maurtua, and E. Lazkano, "Optimal Positioning of Mobile Platforms for Accurate Manipulation Operations," *Journal of Computer and Communications*, vol. 07, no. 05, pp. 1–16, 2019, doi: 10.4236/jcc.2019.75001.

[245] "GreenPatrol EU project," accessed: 2023-07-05. [Online]. Available: http://www.greenpatrol-robot.eu/

[246] "Intel RealSens drivers ROS wiki," accessed: 2023-07-05. [Online]. Available: http://wiki.ros.org/RealSense

[247] A. Gutierrez, A. Ansuategi, L. Susperregi, C. Tubío, I. Rankić, and L. Lenža, "A Benchmarking of Learning Strategies for Pest Detection and Identification on Tomato Plants for Autonomous Scouting Robots Using Internal Databases," *Journal of Sensors*, vol. 2019, pp. 1–15, 2019, doi: 10.1155/2019/5219471.

[248] M. Lopez, J. Martin, U. Gangoiti, A. Armentia, E. Estevez, and M. Marcos, "Supporting Product Oriented Manufacturing: A Model Driven and Agent based Approach," in *Proceedings - IEEE 16th International Conference on Industrial Informatics, INDIN 2018*, jul 2018, pp. 133–139, doi: 10.1109/INDIN.2018.8472069.

[249] "Kobuki fleet repository," accessed: 2017-08-10. [Online]. Available: https://github.com/Jonmg/kobuki_fleet

[250] J. Martin, S. May, S. Endres, and I. Cabanes, "Decentralized robot-cloud architecture for an autonomous transportation system in a smart factory," in *CEUR Workshop Proceedings*, vol. 2063. CEUR Workshop Proceedings (CEUR-WS.org), 2017.

[251] "CODESYS for Raspberry Pi SL," accessed: 2023-02-10. [Online]. Available: https://store.codesys.com/codesys-control-for-raspberry-pi-sl.html

[252] "Wireshark," accessed: 2018-01-31. [Online]. Available: https://www.wireshark.org/

[253] "move-base ROS package," accessed: 2023-02-25. [Online]. Available: http://wiki.ros.org/move_base

[254] J. Martin and T. Scholz, "OPC UA - ROS based Autonomous Transportation System Video," 2018, accessed: 2023-07-10. [Online]. Available: https://youtu.be/EGsDS6ZymKU

[255] L. Melo, R. F. Sampaio, R. Leao, G. Barroso, and J. Bezerra, "Python-based multi-agent platform for application on power grids," *International Transactions on Electrical Energy Systems*, vol. 29, 2019, doi: 10.1002/2050-7038.12012.

[256] J. Palanca, A. Terrasa, V. Julián, and C. Carrascosa, "SPADE 3: Supporting the New Generation of Multi-Agent Systems," *IEEE Access*, vol. 8, pp. 182 537–182 549, 2020, doi: 10.1109/ACCESS.2020.3027357.