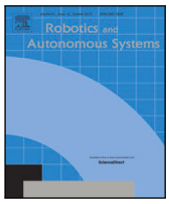
Robotics and Autonomous Systems 184 (2025) 104861

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



Ontological framework for high-level task replanning for autonomous robotic systems

Rodrigo Bernardo [a,b,*], João M.C. Sousa [a], Paulo J.S. Gonçalves [b]

[a] IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[b] IDMEC, Instituto Politécnico de Castelo Branco, Castelo Branco, Portugal

ARTICLE INFO

Keywords:
Semantic knowledge
Ontologies
Autonomous robotic systems
Replanning
Robot control platforms

ABSTRACT

Several frameworks for robot control platforms have been developed in recent years. However, strategies that incorporate automatic replanning have to be explored, which is a requirement for *Autonomous Robotic Systems* (ARS) to be widely adopted. Ontologies can play an essential role by providing a structured representation of knowledge. This paper proposes a new framework capable of replanning high-level tasks in failure situations for ARSs. The framework utilizes an ontology-based reasoning engine to overcome constraints and execute tasks through Behavior Trees (BTs). The proposed framework was implemented and validated in a real experimental environment using an *Autonomous Mobile Robot* (AMR) sharing a plan with a human operator. The proposed framework uses semantic reasoning in the planning system, offering a promising solution to improve the adaptability and efficiency of ARSs.

1. Introduction

Today, the industry is experiencing constant market changes [1]; so that the industry can be competitive, it needs to be highly responsive, flexible and able to reconfigure quickly. Implementing *Autonomous Robotic Systems* (ARSs), such as *Autonomous Mobile Robots* (AMRs), *Autonomous Mobile Manipulator Robots* (AMMRs), etc. can achieve these degrees of flexibility and responsiveness [2]. ARSs involve designing and developing robots that can move and operate autonomously in various environments. Although ARSs have advanced significantly in recent years, these advances still need to meet the requirements imposed by Industry 4.0 and 5.0 [3,4]. These advancements have led to a trend toward integrating and interoperating industrial systems, increasing heterogeneous data and making information management a complex task. Ontologies have been recognized as an effective tool for creating standardized and formal vocabularies that define concepts, enabling the capture and exchange of shared knowledge among various technology partners in contemporary industries [5,6]. Semantic knowledge has emerged as an effective strategy for representing the information required to enhance the flexibility and modularity of production systems, offering advantages over traditional approaches [7,8].

Several frameworks for the robot control platforms have been developed in recent years [9–12]; the ROSPlan [13] has been the most widely used planning system in the past decade. Recently strategies such as SkiROS2 [14] and the PlanSys2 [15] have been that merge task-level planning and execution, based on the use of *Behavior Trees* (BTs) [16]. None of these strategies includes modules or address strategies for replanning, using semantic information, in the event of plan failure. Examples of such failures: if the plan becomes unfeasible due to processing errors, if new information is sensed that violates the initial plan constraints if there is a failure to execute a task or action, if the initial goal changes, etc.

This paper introduces a framework (robot control platform) designed to autonomously replan and adjust a set of tasks for ARSs. The proposed framework is intended to be highly modular and adaptable, taking advantage of BTs, enabling ARSs to conduct high-level planning and task coordination. It emphasizes optimized execution based on BTs and an ontology-based reasoning engine to support logical reasoning. By utilizing semantic knowledge and reasoning about the tasks and actions of the robotic agent and the semantic map of the environment, the inferred knowledge is used to replan the initially defined tasks in the event of failure to achieve the goal. The ontological knowledge is derived from processing information collected by the sensors integrated into the robotic agent and the sensors in the environment where it operates. The proposed domain ontology is based on the *Core Ontology for Robotics and Automation* (CORA) [17] and the *Autonomous Robotic* (AuR) [18] ontologies from the field of automation and robotics. Additionally, the *Semantic Sensor Network* (SSN) [19] ontology, designed for IoT devices and sensor networks, was employed to characterize the sensing capabilities of the devices, their deployment in the physical

* Corresponding author at: IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Portugal.
E-mail address: rodrigo.f.bernardo@tecnico.ulisboa.pt (R. Bernardo).

https://doi.org/10.1016/j.robot.2024.104861
Received 22 August 2024; Received in revised form 15 October 2024; Accepted 8 November 2024
Available online 19 November 2024

environment, and the physical phenomena these devices can observe. The higher-level concepts in the proposed recovery module are based on some concepts proposed by Mohammed Diab et al. in [20] when proposing a FailRecOnt. The FailRecOnt ontology focuses on failure interpretation and recovery in a contingency-based task and motion planning framework, specifically for manipulator robots. These concepts have been used and expanded to encompass all categories of ARS to handle uncertainty, recover from failures, and manage human–robot interactions, etc. The proposed ontology incorporates the *Descriptive Ontology for Linguistic and Cognitive Engineering* (DOLCE) Upper Ontology as a foundational layer, facilitating the interpretation of key concepts and enabling the creation of a domain ontology with a high degree of flexibility [21]. In sum, the framework focuses on abstract operations, logical reasoning, and the system's ability to manage tasks dynamically and intelligently. Considering it a high-level structure, the ontological structure of the framework summarizes the fundamental concepts without going into detail, providing a generic framework that can be easily applied to various autonomous robotic systems (e.g., AMMRs, AMRs, etc.).

The experiment used an AMR and was performed in a simple industrial environment. The knowledge was inferred from semantic data using SPARQL (*SPARQL Protocol and RDF Query Language*) queries. In the replanning process, a basic optimization model was employed to select the optimal location for collecting a specific component from a set of available options, minimizing the travel distance for the mobile agent. Task planning relied on the tasks and actions defined within the ontology and the interpretation of the domain and problem described in the *Planning Domain Definition Language* (PDDL), formulated based on the ontological knowledge inferred through SPARQL queries.

This paper is organized as follows: The next section overviews related work and presents the research gaps. Section 3 introduces the proposed ontological framework. Section 4 presents an example of a practical validation of the proposed framework in a real environment. Finally, Section 5 provides the conclusions of the work and discusses challenges and future prospects.

2. Literature review

2.1. Ontologies in robotic systems

Ontologies are a robust solution for acquiring and sharing common knowledge by formally conceptualizing knowledge representation. They define the concepts and relationships within a specific domain, enabling a shared understanding among stakeholders and promoting semantic interoperability. This shared understanding is often described as *"sharing a common ontology is equivalent to sharing a common world view"* [22]. Depending on their level of generality, ontologies can be categorized into different types. Guarino proposes a classification system based on this generality, identifying four classes: (i) *Top-level* or *Upper ontologies*, which describe very general concepts such as space, time, events, or actions that are independent of any particular problem or domain; (ii) *Domain ontologies* or *domain-specific ontologies*, which focus on general concepts related to a specific domain; (iii) *Task ontologies*, which describe generic tasks or activities; and (iv) *Application ontologies*, which are tailored to specific applications and describe concepts relevant only within a particular domain and task [23].

In robotics, ontologies are frequently used to represent the different entities and concepts that a robotic agent may come across in its environment, including objects, locations, and tasks [24]. By utilizing ontological knowledge, robotic agents can comprehensively reason about these entities instead of treating each object as a separate and distinct entity. This approach improves the robot's understanding and interaction with its environment. Furthermore, ontologies enhance communication between robotic agents and humans, as well as among robotic agents themselves [25]. As robotics advances, ontologies are anticipated to be crucial in enabling autonomous operation in complex and dynamic environments [2]. There are numerous significant works aimed at standardizing knowledge representation within the robotics domain [12,26–30]. Additionally, various standards focus on the application of ontologies in robotics, which can enhance interoperability and support the development of more sophisticated robotic systems. One such initiative is the *IEEE RAS Ontologies for Robotics and Automation (R&A) Working Group* (ORA WG), established in 2011 to connect existing standards from ISO, IEC, and other organizations [26]. The most recent development from this group is CORA, an ontology created in 2015 to map and represent the most fundamental concepts and axioms in the Robotics and Automation (R&A) domain [31]. Furthermore, the *Ontology for Autonomous Robotics* (ROA) conceptual framework allows humans and robots to share information about robot architectures. It defines the core concepts and relationships related to robot architecture for autonomous systems, drawing from SUMO and CORA ontologies [22]. Similarly, the *Ontology for Robotic Architecture* (ORArch) expands on these ideas by detailing how hardware and software components can be jointly represented within mixed architecture descriptions.

2.2. Robot control platforms

In robotics, robot control platforms are crucial for integrating knowledge management, planning, and execution plans for real and simulated robots. The most widely adopted middleware for integrating strategies has been *Robotic Operative System* (ROS[1]). ROS and its successor, ROS 2,[2] were designed with a modular architecture based on packages, making them well-suited for architecture-oriented projects. They offer a collection of software frameworks for robot software development, which can be easily reused across different hardware platforms [32].

Planning is a branch of *Artificial Intelligence* (AI) that focuses on using autonomous techniques to address planning and scheduling problems. Symbolic planning has played a crucial role within AI, mainly due to the development of planning languages such as the *Stanford Research Institute Problem Solver* (STRIPS) [33] and its successor PDDL [34]. These languages provide a standardized framework that allows the integration of planning algorithms developed by the scientific community. PDDL is a formal language employed to define planning domains in AI. It details the actions, objects, and objectives a planner needs to consider when devising a plan.

The ROSPlan [13] framework has been ROS's reference. It provides a collection of packages and tools that allow high-level planning and task coordination of robots and autonomous systems. It is designed to facilitate the planning and execution of complex tasks, making it a valuable tool for various robotic applications. SKiROS [14] is another ROS framework that uses Planning. Besides, it has an ontology-based reasoning engine and recently launched a new version, SKIROS2 [35]. This approach introduces a layered, hybrid control structure for automated task planning and reactive execution underpinned by a knowledge base that facilitates reasoning about the world state and entities. PlanSys2 [15] is an example of a system that applies this structure to task planning in ROS 2. In PlanSys2, the plans generated by the planning algorithms are converted into Behavior Trees (BTs), allowing for optimized execution based on BTs. This optimization is achieved through a novel actions auction protocol and enhanced multi-robot planning capabilities. None of these frameworks proposed a strategy for replanning tasks in the event of failure; it is only mentioned as a future objective.

[1] https://www.ros.org.

[2] https://docs.ros.org/en/foxy/index.html.

2.3. Research gaps

Semantic knowledge in robotics offers a powerful means of acquiring and sharing common knowledge by providing a structured and standardized approach to representing information about robotic agents' environment, capabilities, and goals [22]. This structured knowledge can be utilized to reason about the world and make informed decisions [2]. An ontology can encapsulate information about the robot's sensors, their capabilities, and details about the surrounding objects and environmental properties, such as shape, color, and location. One of the key benefits of using ontologies in robotics is the enhanced ability for robots to communicate effectively with humans and other robots. Robots can exchange information more efficiently and accurately by sharing a common ontology, which is crucial in collaborative scenarios. Several ontology-based domain knowledge representations have been proposed [36], in order to increase the flexibility, re-usability [37], and adaptability of various robotic tasks [38,39] (i.e., recognition [40], navigation [41,42], planning [43], manipulation [44,45], and human–robot interaction [46]) in different environments.

Various methodologies have been proposed to create adaptive planners based on AI to deal with incomplete information to develop valid plans [47]. Manzoor et al. [24] presents a review of ontology-based representation and reasoning techniques, which are used to improve the efficiency of robots in complex tasks. However, strategies for resurfacing when the initial plan fails have yet to be the subject of much research [48]. Zhang et al. [49] introduced the concepts of plan explicability and predictability, which are utilized by autonomous agents, such as robots, to create plans that are both "explicable" and "predictable", making them easily understandable and foreseeable by humans. Bae et al. [50] proposes a hierarchical planning approach based on semantic knowledge for multi-robot systems. They were using knowledge of the relationships and properties of environmental elements. The approach allows robots to plan complex tasks efficiently, dealing with overlap and deadlock problems. Diehl et al. [51] presents a causal model that allows robots to predict and prevent failures. The model uses simulations to learn and predict the success of an action, identifying corrective actions if a failure is anticipated. Focusing on the domestic environment, Wang et al. [52] proposes a task planning mechanism that uses semantic knowledge and execution diagnostics to prevent failures. Ruiz-Celada et al. [53] presents a framework for robot manipulation that plans at the task and movement levels using geometric reasoning modules. It automatically generates BTs that can be edited in real-time, allowing the action plan to be adapted to environmental changes. Mohammed Diab et al. [20] proposes the FailRecOnt framework and introduces an ontology specifically designed for failure interpretation and recovery, utilizing a contingency-based task and motion planning framework. This enables robots to effectively manage uncertainty, recover from failures, and handle human–robot interactions. Hanheide et al. [54] propose a new method for robot task planning and explanation in open and uncertain worlds; however, when a task fails, it uses standard assumptions to infer new knowledge and circumvent the failure state. A goal is created to verify each assumption, and a human is necessary to verify the assumptions.

Future robotic deployments will necessitate that ARSs can repeatedly perform a wide range of tasks across various application domains. Recent high-level task replanning has been developed using semantic knowledge. Most of these works are only applied to particular domains, see e.g. [52,53]. Other strategies do not use an upper ontology to provide a stable base, and they are difficult to expand to more complex cases [49,51,54]. Mohammed Diab et al. [20] presents the richest work in terms of ontologies, but it was only applied to manipulator robotic systems. Therefore, this paper proposes a new framework (robot control platform) for autonomously replanning a plan (set of tasks). This framework uses a high-level knowledge base so that it is generic enough to adapt to the most diverse contexts in which robotic agents can be inserted and take advantage of BTs, enabling ARSs to conduct high-level planning and task coordination.

The proposed platform advances existing frameworks, e.g., ROSPlan, SkiROS2, and PlanSys2 are frameworks that are highly used by the community because they are open-source, are easy to apply to a wide variety of robotic systems and provide the user with high-level tools that allow for easy and effective robot control. However, none of these strategies include replanning strategies in the event of failures.

3. Proposed ontological framework - Robot control platform

As robot systems become increasingly complex, using higher-level representations and coordination has gained importance for enhancing software quality and maintainability [16,55]. These representations, such as abstraction layers, modular architectures, and domain-specific languages, help developers manage system complexity more effectively. By abstracting low-level implementation details, higher-level representations allow a more explicit focus on functionality and system goals, promoting modularity, reusability, and interoperability. Additionally, they enhance maintainability by simplifying updates and scaling while reducing errors through encapsulated, well-defined components [55].

BTs provide a visual representation of control logic for autonomous agents, serving as a versatile and powerful tool for managing robotic systems. Through the use of BTs, robots can navigate complex scenarios, adapt to dynamic environments, and ensure robustness and reliability [56]. BTs comprise four control nodes: fallback, sequence, parallel, and decorator; and two execution nodes: action and condition [56]. In the ROS 2 environment, the developers of the Navigation2 package [57] introduced the recovery node (*"RecoveryNode"*), which has been utilized in this context. As its name suggests, this node links a primary action with a recovery action. Typically, the first action represents the "primary" behavior, while the second action is designed to be executed if the primary behavior fails. The execution (or "ticking") of the second action can often increase the likelihood of the first action's success.

This framework utilizes semantic knowledge to summarize the key concepts in the field without delving into specifics, providing a general framework that can be easily applied to different autonomous robotic systems. By utilizing these high-level concepts, it becomes possible to create plans and adjust them in response to changes in the initial state to prevent potential failure situations. The use of BTs allows for high-level control of the entire system. A system with a subtree in a BT allows modularization, dividing complex tasks into smaller, reusable units (subtrees). These subtrees can be executed independently or combined to create hierarchical and flexible behavior patterns [58]. Based on that, two trees are used to implement a replanning system. The framework we present requires a main tree like the one shown in Fig. 2, which manages all systems at a high level. A second BT communicates with the robotic agent (See Fig. 5). This is launched in the main tree's "*Dispatch Plan*" node, and each action within the BT is specific to each system and is responsible for connecting with the low-level actions of the robotic agents. This second tree is the output of the framework *plan_manager* described in Section 3.1. Whenever there is a need to replan the initial plan, this second tree is "destroyed" and rewritten to satisfy a new plan (sequence of actions/tasks).

The proposed framework, depicted in Fig. 1, comprises five main modules: (i) Behavioral Management (Behavior Tree); (ii) The Knowledge Base; (iii) Plan Manager; (iv) Replanning; (v) Hardware-level (Autonomous Robotic Systems); The implementation of each module is detailed in the following sections.

The main module of our architecture is called *Behavioral Management*. The BT presented in Fig. 2 is designed to manage the ontological framework proposed in this paper. This contains customized retrieval behaviors in specific sub-contexts. Initially, the process waits for a task or a set of tasks (Plan). The plan is created based on the *plan_manager* framework discussed in Section 3.1 of this paper. The condition "Wait

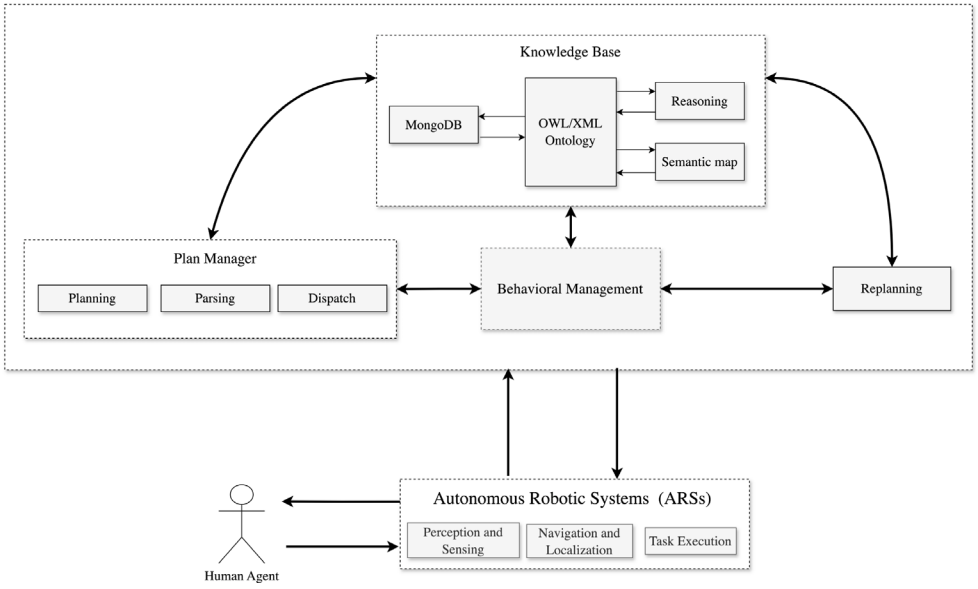


Fig. 1. Drawing of a proposed framework.

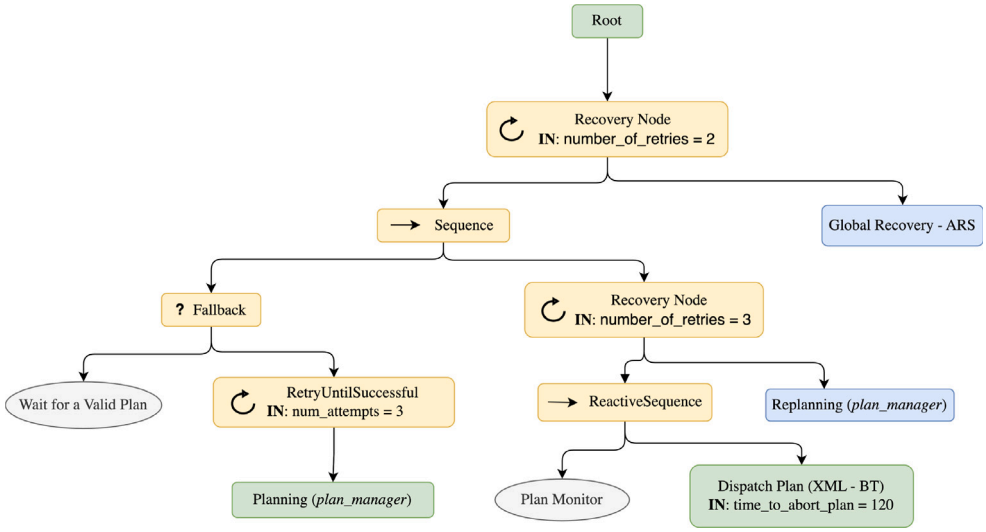


Fig. 2. The main tree of behavioral management shows control flow nodes highlighted in yellow. Nodes of the condition type, displayed in gray, define the conditions that need to be met. Nodes of the action type are depicted in green. Additionally, there is a recovery action shown in blue, responsible for recovering the failure state of the "primary" action of the control node (*RecoveryNode*). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

for a Valid Plan" checks if the tasks in the plan are feasible by the agent (e.g., it checks whether it has enough battery power, etc.). If there is no viable plan, the action *"Planning (plan_manager)"* is initiated up to three times. The process is stopped if the plan is still not feasible after the third attempt. Once a feasible plan is created, it is sent to the agent via the *"Dispatch Plan (XML - BT)"* node. In addition, the *"Plan Monitor"* node tracks all iterations to ensure the tasks that make up the plan remain feasible during the execution. If the plan becomes unfeasible, it is canceled, and the *"Replanning (plan_manager)"* node is called to automatically generate a new plan to address the issues that caused the initial plan to fail. Since the initial plan is sent to the agent, it can be readjusted three times if it becomes invalid before the process is aborted. If these contextual recoveries fail, BT enters the global recovery node (*"Global Recovery - ARS"*). This node is reserved for ARSs system-level failures to help resolve problems like the stuck robot. After each recovery from this node, the previously described process is rerun and has two attempts for the plan to execute; if it does not return SUCCESS, the entire process is addressed, and the whole system must be rebooted.

The diagram in Fig. 3 illustrates the sequence of steps in the proposed framework. The process begins with defining a goal. To achieve the goal, a list of tasks is created, for which a Domain and a Problem are written in PDDL (*Definition of Goal(s)*). These will be interpreted by the *plan_manager*, which is explained in detail in the next section. A module called *Plan Monitor* constantly checks the feedback from the *plan_manager*. If any of the tasks indicate a failure status, this status is passed on to a new module called *Replanning*. If the automatic generation of the plan fails, the failure is communicated to a human agent, who corrects the plan and creates a new, valid plan.

3.1. Plan manager

Robot planning systems integrate knowledge management, planning, and execution to enable robots to perform tasks efficiently. Advancements in planning have enabled robots to perform complex tasks with high accuracy and efficiency. This section presents a *plan_manager* framework used in validation.

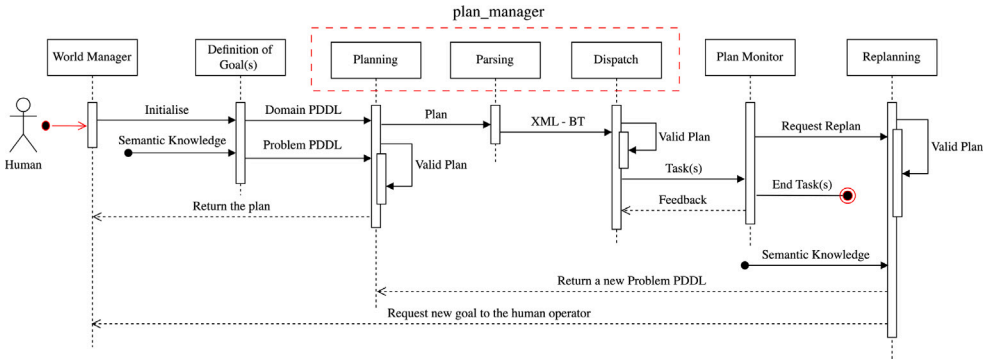


Fig. 3. Sequence diagram of a proposed framework.

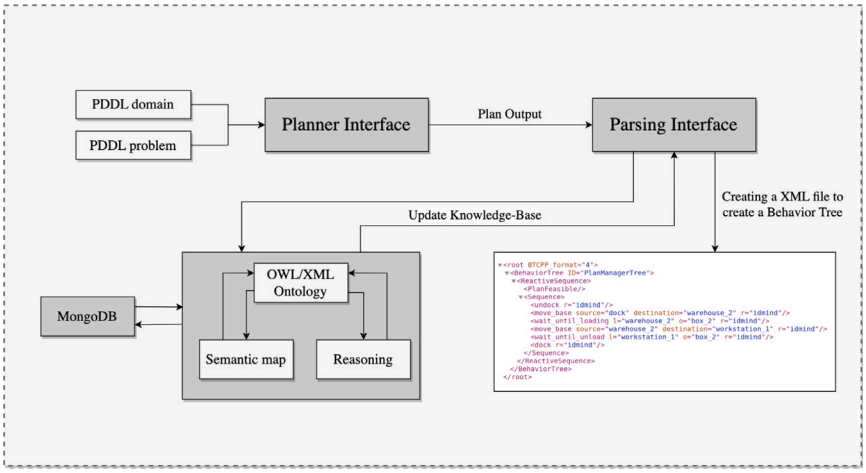


Fig. 4. Plan manager framework.

3.1.1. Design of plan_manager framework

A *plan_manager* framework was developed that uses all the potential BTs (See Figs. 4 and 5). The framework contains the POPF [59] and fast-forward (FF) [60], planners that can be used to generate plans. Note that other planners can be easily integrated into the framework. The main advantage of this new framework is that it uses a richer knowledge base, similar to the SkiROS framework [14,35].

The framework is initialized after generating a domain and a problem in PDDL, which can be generated automatically based on semantic reasoning as presented in a previous work [61], or created manually. The framework has a planner interface node for the *Artificial Intelligence* (AI) planner (e.g., FF). The interface planner is called and returns true if a solution has been found. After a positive response to the creation of a valid problem, a parsing interface node is used to convert the output of the planner interface into an XML file to create a BT (See Fig. 5). Information from the domain and the PDDL problem is added/updated to the knowledge base. The BT generated always comprises a reactive sequence consisting of a condition node where different conditions can be implemented to be checked for the plan to remain valid. And a sequence of actions that are defined based on the output of the planner interface (See Fig. 5). Actions can be added using a parallel node within the sequence node if the actions start at the same instant of time. This happens when temporal planners are used (e.g., POPF).

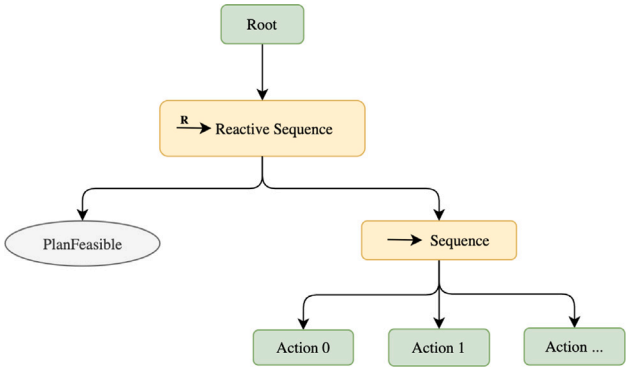


Fig. 5. Example of parsing interface output (BT). Control flow nodes are highlighted in yellow. Nodes of the condition type, displayed in gray, define the required conditions. Nodes of the action type are depicted in green. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.1.2. Implementation

The *plan_manager* framework makes use of several libraries. The core of the framework relies on BTs. The BehaviorTree.CPP[3] library, an open-source C++ library, was used to support type-safe asynchronous actions, composable trees, and provide logging and profiling infrastructure for development. To parse the PDDL files, the pddl[4] library has used, aims to be an unquestionable and complete parser for PDDL 3.1. TinyXML-2[5] is a small, simple, operating system-independent XML parser for the C++ was used to instantiate the BTs. The Owlready2

[3] https://www.behaviortree.dev.

[4] https://github.com/AI-Planning/pddl.

[5] https://github.com/leethomason/tinyxml2.

library was utilized to access the ontologies. Owlready2 is a Python module designed to work with OWL (Web Ontology Language) and RDF standards, allowing users to load, modify, save, and perform reasoning on ontologies. Unlike the standard Java-based API, Owlready2 offers transparent access to OWL ontologies directly in Python, making it a distinctive tool in the ontology management landscape [62]. SPARQL, a query language for accessing data stored in the RDF (Resource Description Framework) format [63], was used to craft complex queries and reason about the knowledge encapsulated within an ontology. By combining the capabilities of SPARQL queries with the functionality of Owlready2, users can efficiently explore and analyze ontologies, extract relevant information, and perform advanced reasoning tasks. Lastly, the MongoDB[6] database was chosen to store all the necessary information (e.g., the information for each one of the instances). MongoDB is a source-available, cross-platform, document-oriented database. As a NoSQL database, MongoDB stores data in JSON-like documents with optional schemas, allowing for flexible and scalable data management across various applications.

3.1.3. Basic usage demonstration

Let us assume that we need to supply the `box_1` to the `workstation_1`. Based on this, the domain and the problem are automatically written in PDDL, such as presented in a previous work [61]. In the example, the FF planning system was used to solve the problem (PDDL plan in Fig. 6(a)). Based on the information from the domain and the problem, the knowledge base (e.g., ontology and database) is updated. If the knowledge base does not contain information about these instances (i.e., `box_1` and `workstation_1`), the system encounters an error because the instances must exist in the knowledge base. If no error occurs and a plan is found, the BT is instantiated using XML format (Fig. 6(b)). Previously, all actions defined in the PDDL domain were implemented in actions of type: *BT::StatefulActionNode*, which is the preferred way to implement asynchronous actions in BTs.

Fig. 6 shows an example of interpreting the domain and problem in PDDL from the AI planner using a provided planner interface and creating the BT in XML format based on a provided parsing interface. Creating a BT follows the standard format shown in Fig. 5. If a temporary planner interface was used, parallel execution nodes *BT::ParallelNode* are used for actions executed at the exact moment inside a *BT::SequenceNode*.

3.1.4. Current limitations

The *plan manager* framework presented has some limitations. The framework only supports the PDDL 3.1, based on the specifications of the library used for the PDDL parser. In addition, for effective human–robot collaboration, a robot control platform must be able to coordinate multiple actuators concurrently in a multi-robot environment. To address these needs in future work, integrating the framework into ROS 2[7] is crucial. This will allow leveraging the new features of ROS 2, such as the real-time multicast communications enabled by DDS.[8]

3.2. Ontology description and knowledge-based reasoning engine

When an agent detects a failure, the main question is what to do about it. In order to infer what to do, the agent needs a causal explanation. For example, why can a component not be loaded at a specific location? What is the causal explanation? Does the object not exist in that location? Is there no human agent available to load the component? Could not reach the location? Some high-level concepts in the proposed recovery module are inspired by the work of Mohammed Diab et al. in [20]. Our ontology aims to define concepts to create these causal explanations in an industrial environment.

The ontology was created using Protégé software, version 5.6.1 [64], to ensure consistency. The reasoning was performed using the Pellet logic reasoner, version 2.2.0 [65]. Protégé, developed by Stanford University, is a free, open-source editor designed to assist tool builders, domain experts, and knowledge engineers in creating ontologies. SPARQL queries were used to query and retrieve information from the ontologies, while the Owlready2 library was utilized to access the developed OWL ontology.

3.2.1. Description of the proposed domain ontology

The proposed domain ontology is built upon the CORA (note the prefix "cora.") and AuR ontologies from the field of automation and robotics. The SSN ontology was used to characterize the capabilities of sensing devices and their deployment in the physical environment. These ontologies (CORA, AuR, and SSN) have the upper ontology DOLCE as a fundamental layer (note the prefix "dul."). In order to accompany the interpretation of some relevant concepts and thus define a domain ontology with a high-level of flexibility, DOLCE is used as a fundamental layer.

The CORA ontology does not incorporate the contextualization and interpretation of the behaviors of robot agents alongside other agents, such as human operators, with overall goals and processes. In other words, CORA does not account for humans acting as independent agents or collaborating with robotic agents to achieve a common goal(s). Human–robot collaboration is a key in modern systems. To address this limitation, the `HumanGroup` was added to the ontology to infer that different humans can cooperate. The `interactsWith` property was defined so that a `cora.RobotGroup` interacts with some individuals of type `HumanAgent` and `HumanGroup`. This extension allows for a more comprehensive representation of agents in the ontology, encompassing the interaction and collaboration between humans and robots to pursue common goals (See Fig. 7).

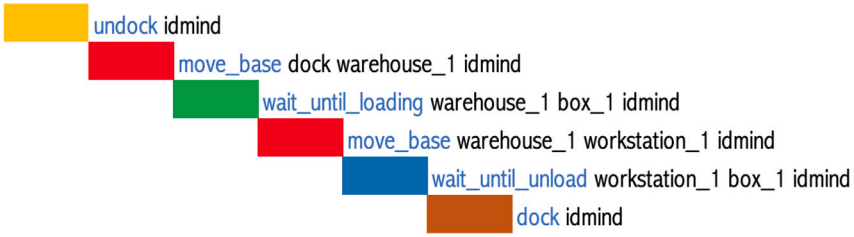
Some higher-level concepts in the proposed recovery module are based on concepts proposed by Mohammed Diab et al. in [20] when proposing a FailRecOnt (note the prefix "fro."). It uses concepts such as: `fro.TaskFailure`, `fro.FailureSympton`, `fro.FailureNarrative` and `fro.FailureDiagnosis`.

- `fro.TaskFailure` has described with *"A Situation which interprets a series of Events as the failed execution of some task(s)"* This situation is described by a *fro.FailureNarrative*.
- `fro.FailureSympton` has been defined as: *"A simple classification label which can be applied to events to interpret them as a failure of some sort"*.
- `fro.FailureNarrative` has been defined as: *"A descriptive context of situations"*.
- `fro.FailureDiagnosis` has been defined as: *"The process is triggered by a detected deviation from the expected behavior of a given system, whose aim is to identify the origin of the failure"*.

The concepts for generating explanatory narratives of the failures that can occur during the execution of the plan have been explored in other works in the literature [66]. Several concepts have been added, such as `RobotNavigationFailures`, `SchedulingFailures`, `PerceptionFailures`, `Human-RobotInteractionFailures`, etc. to identify the highest number of possible failures. These concepts are incorporated into the low-level actions of the agent(s) to provide feedback during their execution, making it easier to identify the type of failure (See. Fig. 8).

The `dul.Situation` is defined as *"A view, consistent with ("satisfying") a Description, on a set of entities."*. Several concepts are added; for example, a `dul.PlanExecution` is a context including some actions executed by agents according to certain parameters and expected tasks to be achieved from a Plan. From the Ontology for Collaborative Robotics and Adaptation (OCRA) [67] (note the prefix "ocra."),

[6] https://www.mongodb.com.
[7] https://docs.ros.org/en/foxy/index.html.
[8] https://www.omg.org/spec/DDS.



(a) PDDL plan.

```
<root BTCPP_format="4">
  <BehaviorTree ID="PlanManagerTree">
    <ReactiveSequence>
      <PlanFeasible/>
      <Sequence>
        <undock r="idmind"/>
        <move_base source="dock" destination="warehouse_1" r="idmind"/>
        <wait_until_loading l="warehouse_1" o="box_1" r="idmind"/>
        <move_base source="warehouse_1" destination="workstation_1" r="idmind"/>
        <wait_until_unload l="workstation_1" o="box_1" r="idmind"/>
        <dock r="idmind"/>
      </Sequence>
    </ReactiveSequence>
  </BehaviorTree>
</root>
```

(b) Tree in XML formal

Fig. 6. Parsing interface: Instantiate trees using XML format generated from PDDL plan.

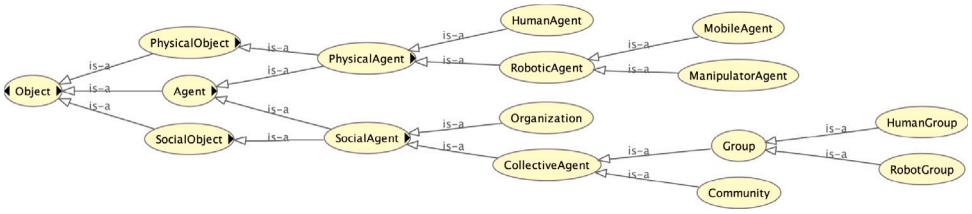


Fig. 7. Relationships of the class `dul.Agent`.

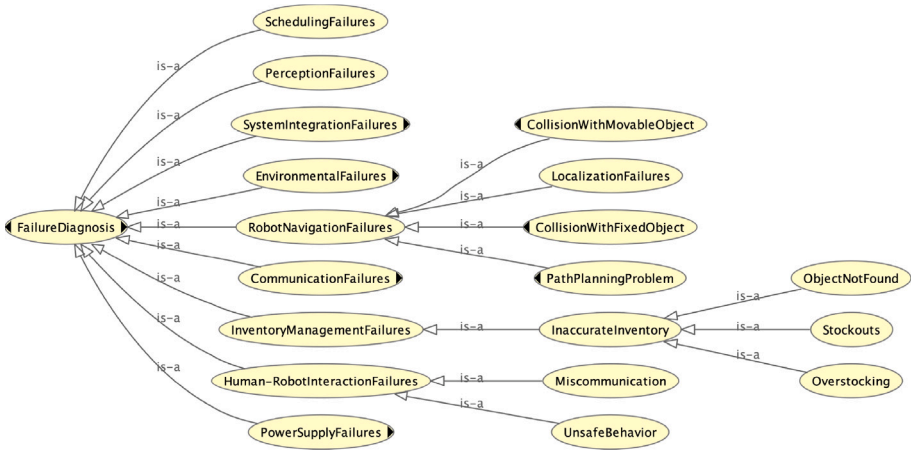


Fig. 8. Description of the some concepts for describing and identifying a failure.

the concept `ocra.PlanAdaptation` was used, which describes *"an event in which one (or more) agent, due to its evaluation of the current or expected future state, changes its current plan while executing it, into a new plan, to continuously pursue the achievement of the plan's goal"*. It was also used the predicate/relationship: `ocra.BetterPlan`, which relates two plans and a situation that makes one of the plans better to achieve a goal, and define similarly `ocra.WorsePlan` as its inverse predicate.

3.2.2. Recovery strategies

The class `RecoveryStrategy` has been added as a subclass of `dul.Task`. This class defines different tasks or actions that an agent

Table 1
Example of some rules that can be adapted based on Priority, TimeConstraint, and ResourcesAvailability of a task.

Rule	hasTaskPriority	hasTimeConstraint	hasResourceAvailability	RecoveryStrategy
1	HighPriority	TightSchedule	ResourcesAvailable	ChangeForEquivalentTask
2	HighPriority	TightSchedule	ResourcesNotAvailable	CallHelp
3	HighPriority	FlexibleSchedule	ResourcesAvailable	RepeatLastAction
4	LowPriority	TightSchedule	ResourcesAvailable	RepeatLastAction
5	LowPriority	TightSchedule	ResourcesNotAvailable	CallHelp
6	LowPriority	FlexibleSchedule	ResourcesNotAvailable	IgnoreFailure

can take to recover from a failure state. A recovery strategy is a method for repairing or reconstructing a plan that initially failed during execution.

- `RepeatLastAction`: When the cause is unknown, one strategy is to repeat the last action, which could be a one-off system failure.
- `ChangeForEquivalentTask`: In a strategy where a specific component needs to be delivered to a location, if the original pick-up location is inaccessible, the initial task can be modified by changing the pick-up location of the component. This change is based on the understanding that there are more components of the same type in other location(s).
- `RemoveFailureTask`: In a complex plan, a strategy can involve removing a task that leads to failure and proceeding with the remaining tasks outlined in the plan, unless that task is a prerequisite for the following tasks. For instance, when distributing multiple components to different locations, if it is not feasible to deliver the components to a specific location, the plan can proceed to transport the remaining components to their respective locations.
- `IgnoreFailure`: A failure is not necessarily a problem for a plan to continue. For example, if the situation arises, it does not prevent the plan from continuing. For example, suppose a human is working with an MMR and only the mobile base is used in the task. If a failure is received from the manipulator, it can be ignored, as it does not need to be addressed for the task.
- `CallHelp`: As a last resort, when all attempts at automatic replanning are unsuccessful, a human agent can be called in to help replan the tasks.

The `dul.Diagnosis` class was used to define the different output states that can come from the tasks/actions the agents can carry out; these states are the outputs of the actions implemented in the *plan_manager*. In the figure, it can seen some of the possible states of `PlanStatus`, `AgentStatus`, `TaskStatus`, etc. The `AgentStatus` class defines the states of the agents at each time they operate, for example, `CandidateForMission`, `NonCandidateForMission`. And it is used the `TaskState` class to define the status of the different constituent of the task (`SpecificFromAction`), for example: `AtPoseState`, `CollisionState`, `PerceptionState`. Different types of `Priority` are also defined for the tasks to facilitate the decision in the reasoner process, as well as several concepts to define the status of the task (`GenericStatus`) (See Fig. 9).

Table 1 presents some rules that have been created to guide the recovery process based on `Priority`, `TimeConstraint`, and `ResourcesAvailability` of a task.

In order to choose the most effective recovery strategy for replanning after a failure, it is essential to define the connections between different concepts and set up guidelines for choosing the appropriate recovery strategy depending on the type of failure and its context. Table 2 presents some rules for selecting the best recovery strategy based on a `fro.FailureNarrative`.

Table 2
Some rules for selecting the best strategy of recovery based on a `fro.FailureNarrative`.

Rule	fro.FailureNarrative	RecoveryStrategy
1	`fro.FailureDiagnosis` is incomplete	RepeatLastAction
2	Initial task cannot be performed due to constraints, but an equivalent task exists	ChangeForEquivalentTask
3	The task can be removed without affecting subsequent tasks	RemoveFailureTask
4	The failure is irrelevant to the continuation of the plan	IgnoreFailure
5	All other strategies have been attempted and failed	CallHelp
6	Lack of necessary resources	CallHelp
7	An environmental constraint (e.g., an obstacle in the path)	ChangeForEquivalentTask
8	A temporary issue (e.g., network outage)	RepeatLastAction
9	A critical system error	CallHelp
10	MechanicalError	CallHelp
11	A SoftwareError	CallHelp

3.3. Real world implementation

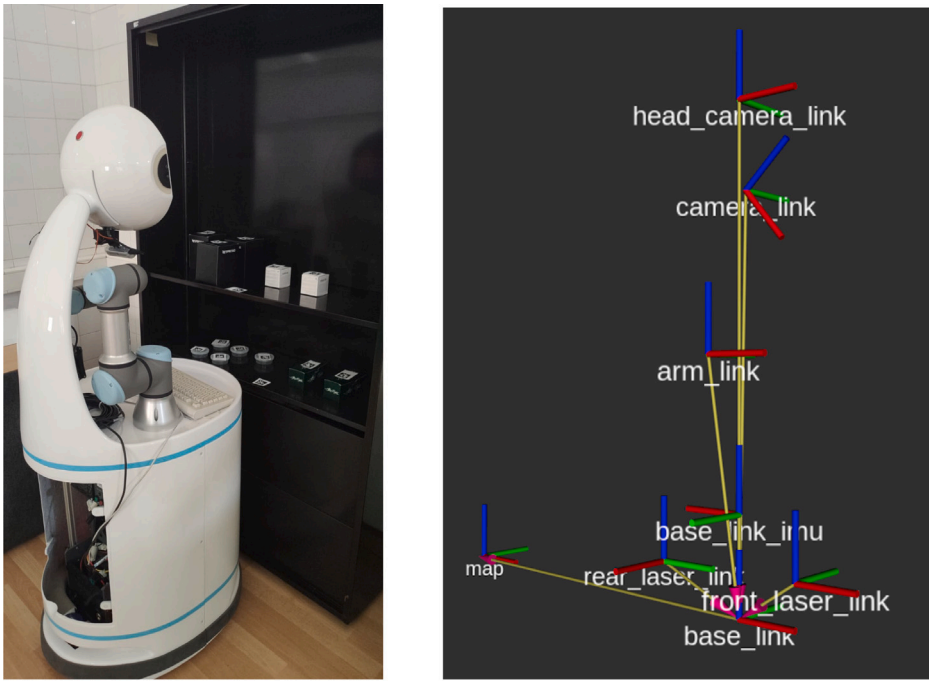
In this study, an AMMR is used to validate the proposed framework. The AMMR consisting of a mobile base and a Universal Robot (UR3[9]) equipped with a Robotiq 2f-140 gripper[10] are used (see Fig. 10). The mobile base has various sensors, including rear and front lasers and cameras, to enable autonomous navigation and obstacle detection. To move between points while avoiding both static and dynamic obstacles, the AMMR uses a Nonlinear Model Predictive Control (NMPC) approach [68]. However, this paper specifically focuses on the mobile base of the agent, known as the Autonomous Mobile Robot (AMR).

The experimental validation was in an environment within the robotics laboratory at Instituto Politécnico de Castelo Branco. Fig. 11 shows the constructed environment, which simulates a compact industrial setting. This setup consists of two warehouses (represented by cabinets with shelves) and three workstations (tables visible in the background).

The environment was characterized through a semantic map; this was built based on the sensors present in the agent (AMMR) presented previously; the strategy used was the one already presented in other previous works of this working group [45,69], where a deep neural network was trained for detecting and classifying the objects present in the environment where the robotic agent is located. In addition, in this work, a Raspberry Pi 4 Model B and a Raspberry Pi V2 camera were installed to observe the working environment. The microcontroller runs the Ubuntu Server 20.04.03 LTS operating system and the ROS Noetic, and the same deep neural network was used to detect and classify

[9] https://www.universal-robots.com/pt/produtos/ur3-robot/.
[10] https://robotiq.com/products/2f85-140-adaptive-robot-gripper.

Fig. 9. The sub classes of the class `dul.SocialObject`.



(a) Robotic agent (AMMR). (b) Frames of the robotic agent.

Fig. 10. Autonomous Mobile Manipulator Robot (AMMR) used in experimental validation and your transform frames (tf).

the objects. In both strategies, the object detection algorithm YOLO v3 [70]. The Darknet for ROS was used for object detection [71]. The YOLO object detector is often acknowledged as one of the fastest deep learning-based object detectors. It can achieve a higher FPS rate compared to more computationally intensive two-stage detectors like Faster R-CNN and some single-stage detectors like RetinaNet [72]. However, despite its speed, YOLO still needs to be faster to run efficiently on embedded devices such as the Raspberry Pi [73]. To maintain system fluidity, the camera image obtained was published to a topic and analyzed on a computer with greater processing power.

The framework presented has been implemented and tested within Robot Operating System (ROS) Noetic and Ubuntu 20.04.4 LTS operating system with an Intel®Core™ i7-7740X CPU @ 3.30 GHz × 8 processor, 16 GB RAM, and Quadro P2000/PCIe/SSE2 Graphics.

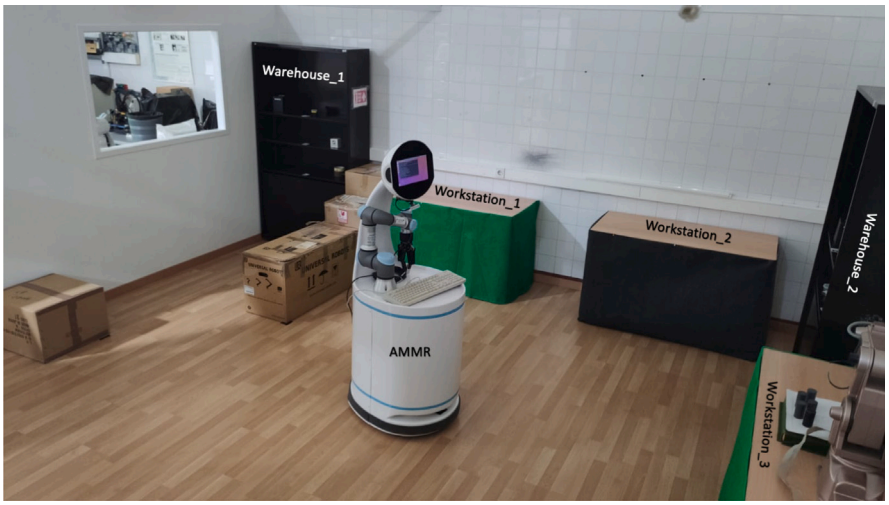


Fig. 11. The simulated industrial environment was created in the robotics laboratory of the Instituto Politécnico de Castelo Branco. It is possible to visualize the environment the AMMR used in the study.

4. Validation of the proposed framework

This section presents two practical case studies representing two of the most common internal logistics failures in order to validate the proposed framework. Fig. 8 shows some of the most common failures that occur in AMRs during navigation (`RobotNavigationFailures`), such as `CollisionWithMovableObject`, `LocalizationFailures`, `CollisionWithFixedObject`, and `PathPlanningProblem`. The first case study addresses issues related to `InacurateInventory`, illustrating how the framework mitigates the impact of failures like `ObjectNotFound` or `Stockouts`. The second case study focuses on `RobotNavigationFailures`, demonstrating how the framework helps resolve problems related to collision avoidance, and path planning during the robot's navigation tasks.

4.1. Failure in a `InacurateInventory`: `ObjectNotFound`

In order to make the validation in a `InacurateInventory` situation more understandable, the section has been subdivided into four sections. Section *Description of the Initial Problem* describes the initial problem. Section *Semantic Replanning* describes how semantic knowledge is used to generate a new valid plan. Section *Simple Optimization Model*, presents a simple case of optimization is presented, which, in this specific case, is used to select the best option from those available for creating the new plan, minimizing the distance the mobile agent has to travel. Section *Recovery Plan* shows the newly formulated plan.

4.1.1. Description of the initial problem

Various factors can contribute to the failure of internal logistics processes in industries [74]. Several failures have been enumerated and included in the ontology presented in this paper (See Section 3.2.2). One type of failure is the absence of a component/object ("`ObjectNotFound`") (See Fig. 8). This leads to the appearance of "critical components", a "critical component" is interpreted as a component in low quantity or even missing in a workstation, which may interrupt the production of the workstation until it is reposted. The present practical case aims to present a solution to overcome failures caused by inventory errors in the warehouse (`Inaccurate Inventory`) (See Fig. 8).

A component is given as critical for `workstation_1`. The responsible operator of the workstation generates an order for the component of type `box` to be provided to his workstation (`workstation_1`), an order that our robotic agent will interpret. A domain and a problem in PDDL are created to satisfy the order. (Fig. 12), a plan is generated. The automatic plan generation has been covered in [61].

4.1.2. Semantic replanning

Fig. 13 simplifies how semantic knowledge is used in this specific case and in which situations it is used to infer new knowledge or oven to check the plan's validity. When the initial plan is generated (Fig. 12), the robot leaves its loading dock; however, before moving to the warehouse, it checks if the component that it is going to get exists in the warehouse, if it exists in the warehouse, it then moves to the warehouse, requesting its arrival to the warehouse operator to load component to its platform, etc.

Fig. 14 shows the graphical interface created for the human–robot interaction; its purpose is to provide information about the task to be performed by the operator, containing information such as which component to load/unload, its location (e.g., in the warehouse), its destination location (e.g., `workstation_1`), quantity, etc. It also allows the operator to alter the defined order, such as the loaded quantity, change the unloading location, etc. The operator can even invalidate the robot's order, aborting the initial plan. All the information the operator can insert is then added/updated to the knowledge base to be used later in the semantic reasoning. It is also equipped with several warning alerts to the operator, such as: "*Move away, the robot will move*".

During execution, the planning agent reformulates the PDDL problem by replanning when a task fails. The recovery mode provides knowledge to interpret the failures, as listed in Section 3.2.2. In this case, the failure has been identified as "`ObjectNotFound`", meaning that the object to be collected is not in the expected location. A replan is therefore required. Semantic knowledge is then used to infer a new plan automatically. The ontology is then queried to determine where components are located to culminate the failure (Listing. 1).

The missing component in `workstation_1` is found in three places. See Table 3. Based on production plans, we need to assess whether we can pick up these components without disrupting the production at their respective locations. This assessment should consider specific metrics such as cycle time and hourly utilization rate. Different cardinalities have been defined, such as the minimum number of components that each location must have so that the production of that location is not affected. A new query is issued (Listing 2) to determine the minimum number of components each location must have to avoid affecting its production.

As a result, two solutions for generating the plan are possible. This follows from the interpretation of the data in Table 3. Collecting the components at `workstation_3` or `workstation_2`. Both options are feasible. However, to make the best choice, an optimization algorithm was developed to choose the best location by minimizing the distance the mobile agent has to travel. The optimization algorithm is presented in the next section.

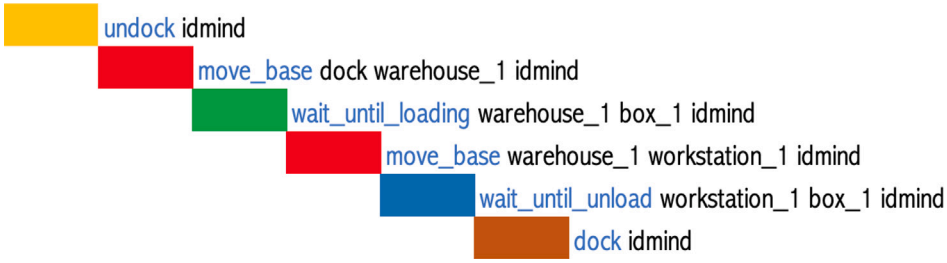


Fig. 12. Generated initial plan.

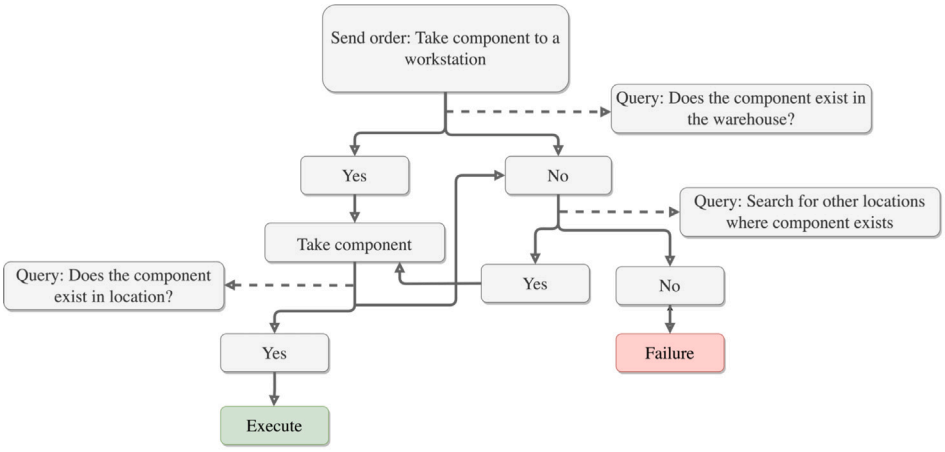


Fig. 13. Flowchart of overview of semantic knowledge use.

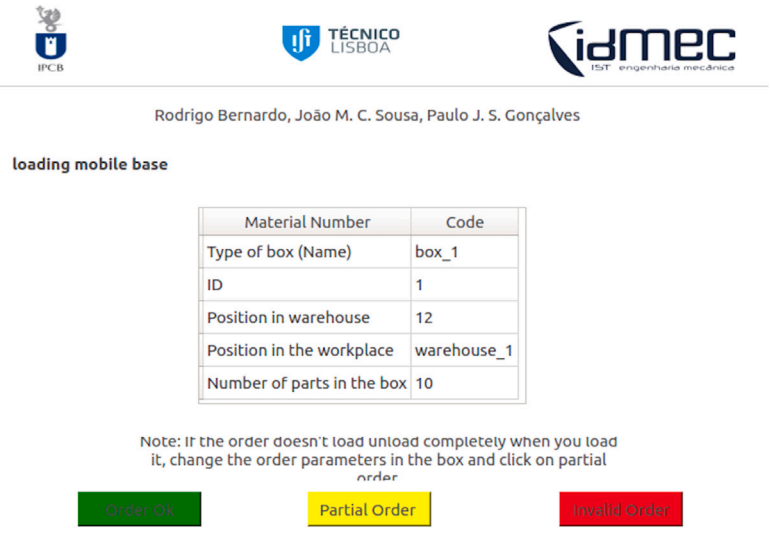


Fig. 14. Interface Human–Robot.

Listing 1: SPARQL query to know the locations and quantity of the critical component.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX on: <http://www.semanticweb.org/idmind#>
    SELECT ?location (COUNT(?component) as ?count)
    WHERE {{
        ?component rdf:type on:box .
        ?component on:hasLocation ?location .
    }}
    GROUP BY ?location
```

Listing 2: SPARQL query about the hourly usage rate of a component in a specific workstation.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX on: <http://www.semanticweb.org/idmind#>
    SELECT ?component ?usage
    WHERE {{
        ?component rdf:type on:{location_class} .
        ?component on:MinimimNumberComponets ?usage .
    }}
```

Table 3
Response to the queries of Listening 1 and 2. When Locations is represented by *?location*, the number of components in locations is represented by *?count* and the minimum number of components is represented by *?usage*.

?location	*?count*	*?usage*
workstation_3	10	5
workstation_2	15	10
workstation_1	5	20

Table 4
Parameters of the mathematical model.

Type	Symbol	Definition
Sets	L	Set of locations nodes, $\{1 \dots n\}$
Indices	i, j	Index of nodes
Parameters	s_i	Semantic location at ith node
	c_i	Number of components available at ith node.
	w_i, w_j	Minimum number of components of ith and jth nodes
	(x_i, y_i)	Metric coordinates of ith node
	$(x_{\text{robot}}, y_{\text{robot}})$	Current location of the robotic agent
	r_{loc}	Semantic location of the robotic agent (e.g., dock)
Decision variables	u_i, u_j	Binary decision variable indicating whether a location is selected (1) or not (0), for ith and jth nodes

4.1.3. Simple optimization model

This section presents a simple optimization model for choosing the best location to pick up a given component from a set of n available locations. The goal is to minimize the distance the robotic agent travels. The constraints ensure that the model returns only the optimal location, the starting point is the agent's current location, and the destination location must be different from the current location, assuming that no components have been observed at the current location and only locations with a number of components greater than the minimum number of components in the localization are valid. The Table 4 shows the parameters of the mathematical model are presented.

The mathematical optimization model is formulated to minimize the distance; see Eq. (1). A Euclidean distance was used to calculate the distance between the current location of the robotic agent and all available locations.

$$\text{Minimize} \quad \sum_{i \in L} u_i \times \sqrt{(x_{\text{robot}} - x_i)^2 + (y_{\text{robot}} - y_i)^2}, \tag{1}$$

subject to

$$\sum_{\substack{i,j \in L \\ w_i \leq w_j}} u_i \times c_i \geq w_j \times u_j, \quad \forall i, j \in L, \tag{2a}$$

$$\sum_{i \in L} u_i = 1, \quad \forall i \in L, \tag{2b}$$

$$\sum_{i \in L, c_i = r_{loc}} u_i = 0, \quad \forall i \in L, \tag{2c}$$

where constraint (2a) ensures that each selected location (e.g., workstation) has enough components available to satisfy the requirement, constraint (2b) ensures that only one location is selected. The constraint (2c) ensures that the robot must go to a different location than the current one, assuming it has not observed any components in the current location.

The authors coded the mathematical model in Python 3.9.6 and have used Gurobi[11] v11.0 to solve the model. The Gurobi optimizer was used to solve the *Mixed-Integer Programming* (MIP) model using a linear-programming based *branch-and-bound* (BB) algorithm.

4.1.4. Recovery plan

Based on the reasoning, the components on `workstation_2` and `workstation_3` meet the requirements; however, based on the output of the optimization model, the `workstation_2` is closer to the current location of the agent, being the best option to minimize the distance of agent need to perform. A new order can be sent to pick up the components without risking the workstation's production. A new plan is then generated to satisfy the initially defined plan (Fig. 15).

The framework proved effective in solving the proposed problem; It has proven to be an easily adaptable strategy to solve other situations that may invalidate an initially defined plan. Failure situations may depend on the environment in which the agent or agents are present. With this practical case, we wanted to prove that combining optimization algorithms and semantic knowledge is an excellent option for obtaining the optimal plan. This is an area that should be explored in future work.

4.2. Failure in a *RobotNavigationFailures: PathPlanningProblem*

In this section, we illustrate a scenario in which the initial plan depicted in Fig. 12 returns a failure of type `PathPlanningProblem` during execution. The failure occurs when the card boxes representing the walls near `warehouse_1` (see Fig. 11) were repositioned to block access to it. This failure (`PathPlanningProblem`) arises from the NMPC's inability to generate a valid path to reach the warehouse. Additionally, there is a change in tasks, and a new task (`HighPriority` task) is added to the agent's workload. The agent must now transport a `Cylinder` to `workstation_2`. The knowledge base is consulted to determine the locations of the `Cylinder` type component (A query similar to 1 was used). The response indicates that the component is available in `warehouse_2`. However, the initial plan still needs to be resolved, and based on the same assumptions explained in Section 4.1, it can be pick up from `workstation_2`. Subsequently, Fig. 16 illustrates the new plan generated to fulfill all the aforementioned requirements.

Replanning a robotic agent's initial plan is essential when a higher priority order arises due to the need to respond efficiently and effectively to new demands or changes in the production environment. It increases the system's flexibility and robustness, allowing for better resource management and a faster response to unforeseen situations.

[11] https://www.gurobi.com.

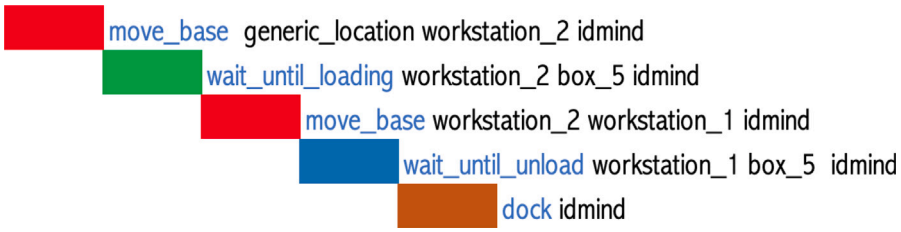


Fig. 15. Plan generated after replan.

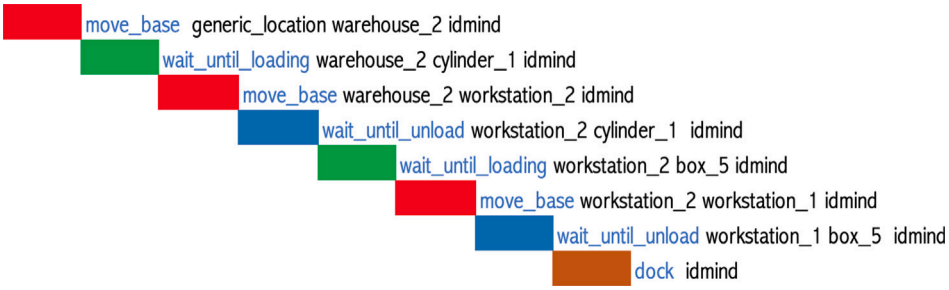


Fig. 16. A new plan to avoid a failure situation and to satisfy a new priority.

4.3. Current limitations

The development of the proposed framework is an ongoing process, with some limitations, such as the organization of the software packages developed so that they can be made available to the entire community, the framework only supports the PDDL 3.1, based on the specifications of the library used for the PDDL parser. This can be limiting based on the type of problems it is applied to. A strategy for adding more domain-specific concepts can be time-consuming. In addition, for effective human–robot collaboration, a robot control platform must be able to coordinate multiple actuators concurrently in a multi-robot environment. To address these needs in future work, integrating the framework into ROS 2[12] is crucial.

5. Conclusions and future work

Several frameworks for robot control platforms have been developed in recent years. However, strategies that incorporate automatic replanning have to be explored, which is a requirement for ARSs to be widely adopted. This paper has proposed a framework (robot control platform) designed to autonomously replan or adjust tasks for ARSs. The proposed framework is intended to be highly modular and adaptable, enabling ARSs to conduct high-level planning and task coordination. The framework utilizes an ontology-based reasoning engine to overcome the constraints and execute tasks through BTs. A domain ontology was proposed based on CORA and AuR ontologies from the automation and robotics field, and the SSN ontology was also employed. The high-level concepts in the proposed recovery module are derived from concepts in the FailRecOnt ontology. The domain ontology was built on top of the Upper ontology DOLCE to have a stable theoretical foundation that promotes clear structuring and disambiguation of new concepts and related relationships. The ontological knowledge was generated by processing the information from the sensors that equip the robotic agent and the sensors in the environment where it acts. Different SPARQL queries were created to infer new knowledge.

The proposed framework was implemented in a real scenario, and its potential was proven through a real logistics problem. Writing new PDDL problems automatically according to the replanning situations proved effective. That allows robots and autonomous systems to perform high-level planning and replanning; it can highlight the optimized execution based on BTs, an ontology-based reasoning engine to support aspects such as logical reasoning and create the possibility to replan. The task planning is based on the tasks and actions defined in the ontology and the interpretation of the PDDL domain and problem written based on the ontological knowledge inferred from SPARQL queries. The validation proved that semantic knowledge integrated with optimization models is a promising strategy and, as future work, is proposed to be explored in larger industrial scenarios. In summary, the presented framework proved to be a strategy with great potential for ARSs in solving problems that, in a usual case, would lead to a failure state.

For future work, continued efforts can be made to add further concepts to the proposed ontology. This will allow the framework to identify the most diverse failures in ARSs and automatically find solutions to avoid failures. New concepts can be added, along with further validation experiments, e.g., for robotic manipulators, expanding the work in [45]. Executing tasks and motion planning deals with complex robotic challenges, such as geometric considerations. For instance, geometric issues like being unable to find a path or finding an inverse kinematic solution to reach an object can make manipulation tasks infeasible. Experiments using AMMRs should also be carried out to enhance replanning strategies by incorporating the use of the mobile base. For example, to address the inability of the manipulator to pick up an object in a table. The mobile base can be moved to other location where the object. The framework will also be implemented in the most recent and increasingly adopted middleware, ROS 2, to validate it, for example, when multiple robots and human agents collaborate on a shared plan.

CRediT authorship contribution statement

Rodrigo Bernardo: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **João M.C. Sousa:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization. **Paulo J.S. Gonçalves:** Writing – review & editing, Supervision, Resources, Methodology, Funding acquisition, Conceptualization.

[12] https://docs.ros.org/en/foxy/index.html.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is financed by national funds through FCT - Foundation for Science and Technology, I.P., through IDMEC, under LAETA, project UIDB/50022/2020. The work of Rodrigo Bernardo was supported by the PhD Scholarship BD/6841/2020 from FCT. This work has indirectly received funding from the European Union's Horizon 2020 programme under StandICT.eu 2026 (under Grant Agreement No.: 101091933).

Data availability

No data was used for the research described in the article.

References

[1] S. Kumar, G. Bawge, B. Kumar, An overview of industrial revolution and technology of industrial 4.0, Int. J. Res. Eng. Sci. 9 (2021) 64–71.
[2] R. Bernardo, J.M. Sousa, P.J. Gonçalves, Survey on robotic systems for internal logistics, J. Manuf. Syst. 65 (2022) 339–350.
[3] T. Masood, P. Sonntag, Industry 4.0: Adoption challenges and benefits for SMEs, Comput. Ind. 121 (2020) 103261.
[4] J. Leng, W. Sha, B. Wang, P. Zheng, C. Zhuang, Q. Liu, T. Wuest, D. Mourtzis, L. Wang, Industry 5.0: Prospect and retrospect, J. Manuf. Syst. 65 (2022) 279–295.
[5] V.R.S. Kumar, A. Khamis, S. Fiorini, J.L. Carbonera, A.O. Alarcos, M. Habib, P. Goncalves, H. Li, J.I. Olszewska, Ontologies for industry 4.0, Knowl. Eng. Rev. 34 (2019).
[6] J.I. Olszewska, J. Bermejo-Alonso, R. Sanz, Special issue on ontologies and standards for intelligent systems, Knowl. Eng. Rev. 37 (2022).
[7] S.K. Chandrasegaran, K. Ramani, R.D. Sriram, I. Horváth, A. Bernard, R.F. Harik, W. Gao, The evolution, challenges, and future of knowledge representation in product design systems, Comput.-Aided Des. 45 (2) (2013) 204–228.
[8] P. Turaga, R. Chellappa, V.S. Subrahmanian, O. Udrea, Machine recognition of human activities: A survey, IEEE Trans. Circuits Syst. Video Technol. 18 (11) (2008) 1473–1488.
[9] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, The claraty architecture for robotic autonomy, in: 2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542), Vol. 1, IEEE, 2001, pp. 1–121.
[10] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, N. Thanh-Hung, Designing autonomous robots, IEEE Robot. Autom. Mag. 16 (1) (2009) 67–77.
[11] M. Stenmark, J. Malec, Knowledge-based instruction of manipulation tasks for industrial robotics, Robot. Comput.-Integr. Manuf. 33 (2015) 56–67.
[12] M. Tenorth, M. Beetz, KnowRob: A knowledge processing infrastructure for cognition-enabled robots, Int. J. Robot. Res. 32 (5) (2013) 566–590.
[13] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, M. Carreras, Rosplan: Planning in the robot operating system, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 25, 2015, pp. 333–341.
[14] F. Rovida, M. Crosby, D. Holz, A.S. Polydoros, B. Groß mann, R.P. Petrick, V. Krüger, SkiROS—a skill-based robot control platform on top of ROS, in: Robot Operating System (ROS) the Complete Reference (Volume 2), Springer, 2017, pp. 121–160.
[15] F. Martín, J.G. Clavero, V. Matellán, F.J. Rodríguez, Plansys2: A planning system framework for ros2, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2021, pp. 9742–9749.
[16] R. Ghzouli, S. Dragule, T. Berger, E.B. Johnsen, A. Wasowski, Behavior trees and state machines in robotics applications, 2022, arXiv preprint arXiv:2208.04211.
[17] I. Robotics, A. Society, IEEE standard ontologies for robotics and automation, IEEE Stan. 1872 (2015) 1–60.
[18] P. Goncalves, A. Olivares Alarcos, J. Bermejo Alonso, S. Borgo, M. Diab, M.K. Habib, H. Nakawala, V.R.S. Kumar, R. Sanz Cortiella, E. Tosello, et al., IEEE standard for autonomous robotics ontology [standards], IEEE Robot. Autom. Mag. 28 (3) (2021) 171–173.
[19] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al., The SSN ontology of the W3C semantic sensor network incubator group, J. Web Semant. 17 (2012) 25–32.
[20] M. Diab, M. Pomarlan, S. Borgo, D. Bebler, J. Rosell Gratacòs, J. Bateman, M. Beetz, FailRecOnt-an ontology-based framework for failure interpretation and recovery in planning and execution, in: Proceedings of the 2nd International Workshop on Ontologies for Autonomous Robotics, 2021, pp. 1–14.
[21] S. Borgo, C. Masolo, Ontological foundations of DOLCE, in: Theory and Applications of Ontology: Computer Applications, Springer, 2010, pp. 279–295.
[22] J.I. Olszewska, M. Barreto, J. Bermejo-Alonso, J. Carbonera, A. Chibani, S. Fiorini, P. Goncalves, M. Habib, A. Khamis, A. Olivares, et al., Ontology for autonomous robotics, in: 2017 26th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN, IEEE, 2017, pp. 189–194.
[23] N. Guarino, Formal Ontology in Information Systems: Proceedings of the First International Conference (FOIS'98), June 6-8, Trento, Italy, vol. 46, IOS Press, 1998.
[24] S. Manzoor, Y.G. Rocha, S.-H. Joo, S.-H. Bae, E.-J. Kim, K.-J. Joo, T.-Y. Kuc, Ontology-based knowledge representation in robotic systems: A survey oriented toward applications, Appl. Sci. 11 (10) (2021) 4324.
[25] E.T. Matson, J. Taylor, V. Raskin, B.-C. Min, E.C. Wilson, A natural language exchange model for enabling human, agent, robot and machine interaction, in: The 5th International Conference on Automation, Robotics and Applications, IEEE, 2011, pp. 340–345.
[26] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, E. Miguelanez, An IEEE standard ontology for robotics and automation, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2012, pp. 1337–1342.
[27] M. Stenmark, J. Malec, Knowledge-based industrial robotics, in: SCAI, 2013, pp. 265–274.
[28] B. Bruno, N.Y. Chong, H. Kamide, S. Kanoria, J. Lee, Y. Lim, A.K. Pandey, C. Papadopoulos, I. Papadopoulos, F. Pecora, et al., The CARESSES EU-Japan project: making assistive robots culturally competent, in: Italian Forum of Ambient Assisted Living, Springer, 2017, pp. 151–169.
[29] M. Waibel, M. Beetz, J. Civera, R. d'Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, et al., Roboearth, IEEE Robot. Autom. Mag. 18 (2) (2011) 69–82.
[30] A. Saxena, A. Jain, O. Sener, A. Jami, D.K. Misra, H.S. Koppula, Robobrain: Large-scale knowledge engine for robots, 2014, arXiv preprint arXiv:1412.0691.
[31] E. Prestes, J.L. Carbonera, S.R. Fiorini, V.A. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Goncalves, M.E. Barreto, M. Habib, et al., Towards a core ontology for robotics and automation, Robot. Auton. Syst. 61 (11) (2013) 1193–1204.
[32] P. Estefo, J. Simmonds, R. Robbes, J. Fabry, The robot operating system: Package reuse and community dynamics, J. Syst. Softw. 151 (2019) 226–242.
[33] R.E. Fikes, N.J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artif. Intell. 2 (3–4) (1971) 189–208.
[34] C. Aeronautiques, A. Howe, C. Knoblock, I.D. McDermott, A. Ram, M. Veloso, D. Weld, D.W. SRI, A. Barrett, D. Christianson, et al., Pddl| the Planning Domain Definition Language, Technical Report, Tech. Rep., 1998.
[35] M. Mayr, F. Rovida, V. Krueger, SkiROS2: A skill-based robot control platform for ROS, in: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2023, pp. 6273–6280.
[36] A. Olivares-Alarcos, D. Beß ler, A. Khamis, P. Goncalves, M.K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, et al., A review and comparison of ontology-based approaches to robot autonomy, Knowl. Eng. Rev. 34 (2019) e29, http://dx.doi.org/10.1017/S0269888919000237.
[37] E.A. Topp, M. Stenmark, A. Ganslandt, A. Svensson, M. Haage, J. Malec, Ontology-based knowledge representation for increased skill reusability in industrial robots, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2018, pp. 5672–5678.
[38] I. Kostavelis, A. Gasteratos, Semantic mapping for mobile robotics tasks: A survey, Robot. Auton. Syst. 66 (2015) 86–103.
[39] R. Bernardo, R. Farinha, P.J. Gonçalves, Knowledge and tasks representation for an industrial robotic application, in: Iberian Robotics Conference, Springer, 2017, pp. 441–451.
[40] H. Azevedo, J.P.R. Belo, R.A. Romero, OntPercept: A perception ontology for robotic systems, in: 2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education, WRE, IEEE, 2018, pp. 469–475.
[41] S.-H. Joo, S. Manzoor, Y.G. Rocha, S.-H. Bae, K.-H. Lee, T.-Y. Kuc, M. Kim, Autonomous navigation framework for intelligent robots based on a semantic environment modeling, Appl. Sci. 10 (9) (2020) 3219.
[42] J. Crespo, J.C. Castillo, O.M. Mozos, R. Barber, Semantic information for robot navigation: A survey, Appl. Sci. 10 (2) (2020) 497.
[43] S. Manzoor, S.-H. Joo, Y.G. Rocha, H.-U. Lee, T.-Y. Kuc, A novel semantic SLAM framework for humanlike high-level interaction and planning in global environment, in: Proceedings of the 1st International Workshop on the Semantic Descriptor, Semantic Modeling and Mapping for Humanlike Perception and Navigation of Mobile Robots Toward Large Scale Long-Term Autonomy (SDMM1), Macau, China, Vol. 8, 2019.
[44] M. Ersen, E. Oztop, S. Sariel, Cognition-enabled robot manipulation in human environments: requirements, recent work, and open problems, IEEE Robot. Autom. Mag. 24 (3) (2017) 108–122.

[45] R. Bernardo, J.M. Sousa, P.J. Gonçalves, A novel framework to improve motion planning of robotic systems through semantic knowledge-based reasoning, Comput. Ind. Eng. (ISSN: 0360-8352) 182 (2023) 109345, http://dx.doi.org/10.1016/j.cie.2023.109345, URL https://www.sciencedirect.com/science/article/pii/S0360835223003698.
[46] E. Fernández-Rodicio, Á. Castro-González, J.C. Castillo, F. Alonso-Martin, M.A. Salichs, Composable multimodal dialogues based on communicative acts, in: International Conference on Social Robotics, Springer, 2018, pp. 139–148.
[47] Z. Wang, G. Tian, Hybrid offline and online task planning for service robot using object-level semantic map and probabilistic inference, Inform. Sci. 593 (2022) 78–98.
[48] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, W. Ruml, Replanning for situated robots, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 29, 2019, pp. 665–673.
[49] Y. Zhang, S. Sreedharan, A. Kulkarni, T. Chakraborti, H.H. Zhuo, S. Kambhampati, Plan explicability and predictability for robot task planning, in: 2017 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2017, pp. 1313–1320.
[50] S. Bae, S. Joo, J. Choi, J. Pyo, H. Park, T. Kuc, Semantic knowledge-based hierarchical planning approach for multi-robot systems, Electronics 12 (9) (2023) 2131.
[51] M. Diehl, K. Ramirez-Amaro, A causal-based approach to explain, predict and prevent failures in robotic tasks, Robot. Auton. Syst. 162 (2023) 104376.
[52] Z. Wang, G. Tian, X. Shao, Home service robot task planning using semantic knowledge and probabilistic inference, Knowl.-Based Syst. 204 (2020) 106174.
[53] O. Ruiz-Celada, P. Verma, M. Diab, J. Rosell, Automating adaptive execution behaviors for robot manipulation, IEEE Access 10 (2022) 123489–123497.
[54] M. Hanheide, M. Göbelbecker, G.S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek, et al., Robot task planning and explanation in open and uncertain worlds, Artificial Intelligence 247 (2017) 119–150.
[55] D. Kortenkamp, R. Simmons, D. Brugali, Robotic systems architectures and programming, in: Springer Handbook of Robotics, Springer, 2016, pp. 283–306.
[56] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and AI: An Introduction, CRC Press, 2018.
[57] S. Macenski, F. Martín, R. White, J. Ginés Clavero, The marathon 2: A navigation system, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2020, URL https://github.com/ros-planning/navigation2.
[58] S.S.O. Venkata, R. Parasuraman, R. Pidaparti, Kt-bt: A framework for knowledge transfer through behavior trees in multirobot systems, IEEE Trans. Robot. (2023).
[59] A. Coles, A. Coles, M. Fox, D. Long, Forward-chaining partial-order planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 20, 2010, pp. 42–49.
[60] J. Hoffmann, FF: The fast-forward planning system, AI Mag. 22 (3) (2001) 57–57.
[61] R. Bernardo, J.M. Sousa, P.J. Gonçalves, The use of semantic knowledge in task planning for robotic agents, minimising human error, in: Iberian Robotics Conference, Springer, 2023, pp. 3–13.
[62] J.-B. Lamy, Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies, Artif. Intell. Med. 80 (2017) 11–28.
[63] E. Sirin, B. Parsia, et al., SPARQL-DL: SPARQL query for OWL-DL, in: OWLED, Vol. 258, 2007.
[64] Protégé, Protégé, 2022, URL https://protege.stanford.edu.
[65] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical owl-dl reasoner, J. Web Semant. 5 (2) (2007) 51–53.
[66] A. Olivares-Alarcos, A. Andriella, S. Foix, G. Alenyà, Robot explanatory narratives of collaborative and adaptive experiences, in: 2023 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2023, pp. 11964–11971.
[67] A. Olivares-Alarcos, S. Foix, S. Borgo, G. Alenyà, OCRA–An ontology for collaborative robotics and adaptation, Comput. Ind. 138 (2022) 103627.
[68] R. Bernardo, J.M. Sousa, M.A. Botto, P.J. Gonçalves, A novel control architecture based on behavior trees for an omni-directional mobile robot, Robotics 12 (6) (2023) 170.
[69] R. Bernardo, J. Sousa, P.J. Gonçalves, Planning robotic agent actions using semantic knowledge for a home environment, Intell. Robot. 1 (2) (2021) 116–130.
[70] J. Redmon, A. Farhadi, YOLOv3: An incremental improvement, 2018, arXiv.
[71] M. Bjelonic, YOLO ROS: Real-time object detection for ROS, 2016–2018, https://github.com/leggedrobotics/darknet_ros.
[72] T. Diwan, G. Anirudh, J.V. Tembhurne, Object detection using YOLO: Challenges, architectural successors, datasets and applications, Multimedia Tools Appl. 82 (6) (2023) 9243–9275.
[73] G. Lan, J. Benito-Picazo, D.M. Roijers, E. Domínguez, A. Eiben, Real-time robot vision on low-performance computing hardware, in: 2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV, IEEE, 2018, pp. 1959–1965.
[74] L. Barreto, A. Amaral, T. Pereira, Industry 4.0 implications in logistics: an overview, Procedia Manuf. 13 (2017) 1245–1252.

Rodrigo Bernardo received a master's degree in industrial Automation Engineering from the University of Aveiro, Aveiro, Portugal, in 2019 and is currently a Ph.D. candidate in Mechanical Engineering at Instituto Superior Técnico, Lisbon, Portugal. His research interests are Robotics and Automation, Ontologies, and Semantic Knowledge and Reasoning.

João M. Costa Sousa is Full Professor with the Dept. Mechanical Engineering, Instituto Superior Técnico (IST), Universidade de Lisboa, Portugal. He is the Coordinator of the Center of Intelligent Systems, IDMEC. He received the M.Sc. in mechanical engineering from IST in 1992, and the Ph.D. in electrical engineering from Delft University of Technology, the Netherlands, 1998. He has authored one book and more than two hundred papers in journals and conference proceedings. He has supervised more than 100 Ph.D. and M.Sc. students. He participated in more than 20 research projects, being the Principal Researcher in seven, five of them with industry in intelligent automation and machine learning. He was the PI of an Industry 5.0 project with the pharmaceutical industry. He is Associate Editor of IEEE Transactions on Fuzzy Systems, Mathematics and Computers in Simulation, Sensors, and Fuzzy Sets and Systems. He is past Chair of the Fuzzy Systems Technical Committee, IEEE Computational Intelligence Society.

Paulo J. Sequeira Gonçalves (Senior Member, IEEE) was born in Covilhã, Portugal, in 1972. He received the M.Sc. and Ph.D. degrees in mechanical engineering (control, automation, robotics, and industrial informatics scientific area) from the University of Lisbon, Lisbon, Portugal, in 1998 and 2005, respectively.

He is currently an Associate Professor with the Department of Electrotechnical and Industrial Engineering, Polytechnical University of Castelo Branco (IPCB), Castelo Branco, Portugal. He is a Senior Researcher with the Institute of Mechanical Engineering, Instituto Superior Técnico, University of Lisbon. He is responsible for the Robotics and Intelligent Equipment Laboratory of IPCB. His main research interests include ontologies, computational intelligence, robotics, industrial automation, computer vision, and in particular visual servo control of robots. He is the author or coauthor of over 150 journal articles, books, book chapters, and conference articles across the various domains described above.

Prof. Sequeira Gonçalves received the "Scientific Merit Award" from IPCB in 2017 and 2022, for his merits. He is currently serving as the Vice-Chair of the IEEE RAS Portuguese Chapter. He also served as the chair of the IEEE Computational Intelligence, Portuguese Chapter. He is currently the Chair of the IEEE P1872.3 Standard for Ontology Reasoning on Multiple Robots, and was an Officer of the working groups who worked toward IEEE RAS related standards, P1872, P1872.2 and P7007. IEEE standards (1872 and 7007) received the IEEE SA "Emerging Technology Award" in 2015 and 2021. He is an Active Member of the IEEE Standards Association and IEEE Robotics and Automation Society.