applied sciences

MDPI

Review

Robot Operating System 2 (ROS2)-Based Frameworks for Increasing Robot Autonomy: A Survey

Andrea Bonci *, Francesco Gaudeni, Maria Cristina Giannini and Sauro Longhi

Dipartimento di Ingegneria dell'Informazione (DII), Università Politecnica delle Marche, 60131 Ancona, Italy; s1097348@studenti.univpm.it (F.G.); m.c.giannini@staff.univpm.it (M.C.G.); sauro.longhi@univpm.it (S.L.)
* Correspondence: a.bonci@univpm.it; Tel.: +39-071-220-4666

Featured Application: This work aims to review the use of ROS2 as middleware to integrate heterogeneous hardware and software components in order to enable fixed-base robots to perform complex tasks by increasing their autonomy and flexibility. It shows how the open-source framework can be used in industry to overcome the limitations of commercially available cobots. An extensive review of the features and tools currently provided by ROS2, as well as its main fields of application, is provided. Moreover, as a proof of concept, a high-level modular architecture to increase autonomy in industrial operations is first proposed and then applied to one of the various commercially available industrial cobots.

Abstract: Future challenges in manufacturing will require automation systems with robots that are increasingly autonomous, flexible, and hopefully equipped with learning capabilities. The flexibility of production processes can be increased by using a combination of a flexible human worker and intelligent automation systems. The adoption of middleware software such as ROS2, the second generation of the Robot Operating System, can enable robots, automation systems, and humans to work together on tasks that require greater autonomy and flexibility. This paper has a twofold objective. Firstly, it provides an extensive review of existing literature on the features and tools currently provided by ROS2 and its main fields of application, in order to highlight the enabling aspects for the implementation of modular architectures to increase autonomy in industrial operations. Secondly, it shows how this is currently potentially feasible in ROS2 by proposing a possible high-level and modular architecture to increase autonomy in industrial operations. A proof of concept is also provided, where the ROS2-based framework is used to enable a cobot equipped with an external depth camera to perform a flexible pick-and-place task.

Keywords: Robot Operating System 2 (ROS2); Robot Operating System (ROS); collaborative robotics; industrial robotics; flexibility; autonomy

Citation: Bonci, A.; Gaudeni, F.; Giannini, M.C.; Longhi, S. Robot Operating System 2 (ROS2)-Based Frameworks for Increasing Robot Autonomy: A Survey. *Appl. Sci.* **2023**, *13*, 12796. https://doi.org/10.3390/app132312796

Academic Editor: Adel Razek

Received: 8 October 2023
Revised: 21 November 2023
Accepted: 22 November 2023
Published: 29 November 2023

1. Introduction

Achieving high levels of autonomy for robotic systems, especially in manufacturing, is a current challenge for the industry. The reasons can be many and often with significant impacts on business productivity. The industry's current need is to move from traditional production systems to so-called cyber-physical systems (CPSs) to improve overall productivity, reduce environmental impact, and enable more human-centered production—in other words, to promote more sustainable economic development. In these systems, software, robots, sensors, machines, and/or devices cooperate with each other to achieve the desired result. Robots are required to participate in CPSs [1–3]. The high heterogeneity of the devices comprising CPS systems requires expertise in various scientific domains, making the development of CPS a resource and time-consuming process. Conversely, despite the extensive use of new enabling technologies, humans are always involved at different levels of production processes and can interact physically with systems, e.g., by

loading or unloading parts from machines, or by interacting through interfaces, e.g., by intervening in the production schedule. In addition, global competition, rapidly evolving markets, and mass customization will replace traditional mass production, requiring new production challenges such as increased product variability and quality [4], and the need to reduce production and delivery times for both SMEs [5] and large companies [6]. All these challenges will require automation systems with robots that are increasingly autonomous, flexible, and hopefully equipped with learning capabilities. While humans are flexible and adaptable by nature and thus able to react to both small disturbances and large changes, the same does not apply to robots or traditional automation systems. The flexibility of production processes can be increased by using a combination of a flexible human worker, preferably with assessable performance [7], and collaborative robots (cobots) [8]. Although robots have gradually replaced humans in performing simple repetitive tasks in dangerous environments [9], cobots and robot automation are usually limited to monotonous and structured tasks (i.e., pick and place, grasping, moving workpieces, welding, and painting), while robots employed in assembly processes require dexterous human assistance, and for complex assembly tasks, workers still rely on manual operations. Mixed human–machine industrial environments, where operations can be performed differently, i.e., collaboratively, cooperatively, or individually, depending on production needs, require non-traditional perception, planning, and control systems. To ease the integration of collaborative robots into intelligent automation systems, it has become very useful to resort to middleware platforms that enable hardware abstraction and communication between heterogeneous devices and components. Robot Operating System (ROS), developed and managed by Open Robotics [10] has become a de facto standard in academia and research for many robotic systems for at least a decade. Its second version ROS2 [11], was subsequently developed to fill the gaps in ROS, identified by industry and academia. As a result, the adoption of middleware paves the way for the use of ROS2-based architectures in real-world industrial automation systems to leverage robots, automation systems, and humans to work together on tasks that require greater adaptability and flexibility.

In this context, this paper has a twofold objective: (a) starting from an extensive review of existing literature, the features and tools currently provided by ROS2, as well as its main fields of application, are examined, with the aim of highlighting the enabling aspects for the implementation of modular architectures to increase autonomy in industrial operations; (b) it is shown how this is currently potentially feasible in ROS2 by proposing a possible high-level modular architecture to increase autonomy in industrial operations, supporting the conceptualization with a proof of concept developed on one of the various commercially available industrial cobots. Although the approach is general for robots, in this work, the applications will focus on robotic manipulators and, in particular, on collaborative robots (cobots), for which demand is expected to increase in the manufacturing sector in large and especially small- and medium-sized enterprises, for which it is particularly relevant to ensure flexibility with reduced cost and setup time achievable with increased autonomy. A further reason to address autonomy issues using cobots lies in being able to apply them even in tasks involving the presence of humans, as well as in understanding and overcoming some of their limitations due to their inherent safety systems.

Cobots can indeed operate safely in close proximity to humans, although they generally move more slowly than their industrial counterparts. Despite their greater proximity, with regard to safety, cobots do not require additional information from humans in case of new or unknown events, as they are designed for safety and can stop without harming humans in the event of contact. However, the inherent characteristics of cobots alone are not enough to require robots and humans to work together or in close proximity to perform more complex tasks such as assembly or other operations that require adaptability and flexibility. In tasks requiring a high degree of adaptability, cobots must become fully independent in adapting to different scenarios or tasks in handling interaction with both humans and dynamic environments.

To cope with these demands, capabilities of environment perception, object recognition, trajectory planning, and re-planning are required in a dynamic and changing environment, even with possible human interaction. These features are not yet all available simultaneously in collaborative robots. In addition to a hardware platform to perform tasks safely, a cobot also needs a complex software architecture that incorporates algorithms that enable it to act possibly autonomously. Furthermore, as perception technologies improve, artificial intelligence would have the potential to help robots and humans work together on physical tasks, such as assembly. It follows that a single technology, be it robotics, rather than AI, sensors, or intelligent automation, is not sufficient on its own to guarantee high levels of adaptability. A middleware is needed to enable interaction and communication between them. The use of perception technologies, together with learning techniques and especially a middleware layer that orchestrates actions and information, has the potential to help robots and humans work together on complex physical tasks, such as assembly or others. This could help meet the challenge of customized production.

Hence, this paper deals with the use of ROS2 as middleware to integrate heterogeneous hardware and software components in order to enable commercially available fixed-base robots to perform complex tasks by increasing their autonomy and flexibility. Specifically, the main twofold objectives above mentioned can be further detailed as follows: (a.1) provide an overview of the aspects of ROS2 most widely addressed in literature; (a.2) provide an extensive survey of the features and tools provided by ROS2 that can be used to implement high-level modular architectures, given the lack of dedicated literature on the topic; (b.1) propose an high-level modular architecture to enhance the autonomy of fixed-base cobots; and (b.2) show a proof of concept of an ROS2-based framework to enable a cobot equipped with an external depth camera to perform a flexible pick-and-place task.

The remainder of the paper is organized as follows to address these specific objectives: Section 2 gives a literature review on ROS2, Section 3 describes the architecture for increasing autonomy, Section 4 deepens the architecture implementation in ROS2, Section 5 presents an application case study, and Section 6 concludes the paper.

2. Related Work

2.1. ROS: Origin, Potential, and Limits

The ROS framework has been widely used in academia and research for nearly fifteen years, since the original paper [12] was published by the robotics incubator Willow Garage [13], which developed it almost entirely from scratch and has maintained it with the OSRF (Open-Source Robotics Foundation) [10] since 2007. It has been used to orchestrate heterogeneous systems and tasks in a variety of robotics applications to simulate and implement complex control, planning, and coordination tasks. Autonomy has been essential in both robotics and automotive; since the 2007 DARPA Urban Challenge [14], autonomous driving has attracted attention and helped spread the growing development of new solutions in both fields. ROS open-source middleware has undergone rapid development [15] and has been widely used for robotics applications, autonomous systems, and intelligent machines. This has happened mainly thanks to both the set of libraries useful for many types of robots (e.g., Open-Source Computer Vision (OpenCV) [16,17] and Point Cloud Library (PCL) [18]), and the ecosystem of algorithm packages for sensors, control, and connection made available by community contributions that help to improve productivity [19].

Although ROS solves many of the complexity problems inherent in robotics, it is nevertheless lacking in several aspects, such as lacking real-time execution requirements, running only on a few OSs (operative systems), not being able to guarantee deadlines or process synchronization nor fault-tolerance, and having no built-in security mechanisms. As research uses ROS for increasingly complex tasks, the foundations of the ROS research platform began to show its limitations. However, industrial robotics, whether mobile or manipulative, can also benefit from the adoption of ROS solutions in industrial environments to enable intelligent and flexible automation that can tackle not only standard and

repetitive tasks, but also complex and dynamic tasks, which are still ongoing challenges for the factories of the future. Examples are cooperative mobile manipulators where it is necessary to coordinate the moving base, manipulator, and interaction forces with the moved objects, which for complex tasks may require nonlinear controls as in [20] with guaranteed execution times [21].

2.2. ROS2: Origin and Performance Analysis

To overcome the limitations of ROS and meet the needs of both the industry and the broader ROS community by providing features such as higher efficiency, reliability, security, and real-time processing, after an initial announcement in 2014, in 2018 Open Robotics released ROS2 [11,22], the second generation of ROS. Given the intrusive nature of the changes required to achieve these benefits, in order to keep ROS working and to be unaffected by the new developments, the ROS community decided to redesign the middleware from scratch.

One of the most significant changes in ROS2 is the switch from the TCP/UDP network protocol used in ROS to DDS (Data Distribution Service) as the default middleware for communication between nodes. DDS is an industry standard published by the Object Management Group (OMG) [23] that defines a middleware for real-time and reliable data distribution according to the publish/subscribe paradigm, thus enabling ROS2 to meet the safety, security, resilience, and fault-tolerant of distributed systems. Specifically, ROS2 is compatible with multiple DDS vendors (e.g., the default eProsima [24], RTI [25], OpenDDS [26], ADLINK [27], and others).

An early study exploring the performance of ROS2 was conducted by Maruyama et al. [28] and dates back to 2016. The authors conducted proof-of-concept experiments to clarify the capabilities of ROS2 and evaluated the performance characteristics of DDS [29,30] in ROS2 in various situations and for different aspects: latency, throughput, number of threads, and memory consumption. The performance of DDS was further analyzed in [31], showing that it can be used in some classes of industrial control systems.

In more recent years, numerous research papers have been devoted to the evaluation of ROS2 performance related to various aspects: DDS, security, and distributed robotic systems. Without claiming to be exhaustive (a mapping of software engineering research on ROS can be found in [32]), several attempts to clarify DDS issues can be found in [33], which conducted analyses on ROS2 and several DDS providers. The authors of [34] evaluated the response time of processing chains in ROS2, and a latency analysis of ROS2 was provided in [35]. More recently, in [36], the authors presented an experimental evaluation of the execution of ROS2's smallest schedulable entities, namely callbacks. They identified and showed the differences in callback execution using different executor versions (i.e., the ROS2 callback scheduler) and illustrate the negative impact of the new executor on the periodic execution of timer callbacks with a real-world use-case on a multi-agent robotic system. Distributed robotic systems were addressed in [37,38], which evaluated real-time ROS2 characteristics for multi-agent robotic systems and control performance in time-synchronized distributed networks, respectively. In the same domain, the network performance of ROS2 under varying QoS (quality of service) profiles and security settings was studied in [39]. Finally, some works related to the security issues of ROS2 can be found in [40–42]. As addressed in [43,44], finding a trade-off between performance and security is not an easy task.

Most of these evaluations conclude that an ROS2-based architecture, if properly developed, can satisfy the real-time constraints that are often required in industrial applications to meet safety and/or performance goals, unlike the previous ROS version. This is made possible, at a communication level, by employing DDS that guarantees more stable communications with lower message loss and mainly lower communication latency in both an idle network environment and with network traffic. In order to meet the constraints of real time as a whole, the communication level alone is not sufficient; however, ROS2 exhibits real-time performance even in the development environment by supporting RTOS

(Real-Time Operating System) such as the Linux kernel patched with PREEMPT_RT. The use of RTOS potentially enables real-time scheduling at both node and callback levels, ensuring lower task latency (i.e., time spent by the task in the ready state waiting for CPU) and scheduling jitter (i.e., the unwanted variation of the release times of a periodic task) and then determining and meeting deadlines. Nevertheless, it is crucial to note that these real-time capabilities of ROS2 are potential and not guaranteed in every application developed. It is the developer's responsibility to ensure the availability of an RTOS underlying and to manage scheduling effectively using executors.

2.3. ROS2: Industrial Robotics Applications

Regarding the use of ROS2 in industrial robotics tasks to increase autonomy, flexibility, and human–machine cooperation, in short toward intelligent automation systems, again, the literature is not very extensive and has focused on solving specific problems. In [45,46], the challenges of achieving flexible automation in an industrial plant consisting of a collaborative robotic assembly station are described. ROS was used to reuse old ROS packages to integrate smart devices and algorithms, then a communication architecture based on ROS2 with communication bridges was developed to pass messages between ROS and ROS2. In [47], ROS2 was proposed as a software framework to control the automation equipment of a reliable and robust automation system consisting of a motion control system (MCS) and an OpenPLC. A review of ROS2-compatible simulation software for manipulation with robotic arms, along with a comparison of them, can be found in [48]. A review highlighting the key changes of ROS2 in robotics is given in [49]. The article also shows through case studies the influence that ROS2 and its adoption have had in accelerating the development of real and reliable robotic systems, even in challenging environments. The spread of robots in unstructured and dynamic environments and the issue of path re-planning are gaining importance in robotics. Several works in ROS were devoted to this issue (e.g., [50–52] and references therein), but at present, however, no scientific work seems to be available dealing with reliable native re-planning tools in ROS2.

Staying within the same application domain but relaxing the constraint of using ROS2, the literature is more extensive, with several works where architectures/frameworks enabling robots to perform more advanced tasks through the use of middleware are proposed. In [53], an application that integrates a 3D perception module, a multi-layer motion planning module, and a real-time motion control module to autonomously perform a complex robotic assembly task is proposed without specifying the middleware used for its implementation. Similarly, in [54], a three-layer architecture (perception, planning, and control) is introduced for a human–robot cooperative assembly task. Conversely, other works explicitly specify the use of different middleware for the software design. In [55], a mobile pick and place in unstructured environments is achieved with a modular framework built on ROS to facilitate both the porting to other mobile manipulators and the integration of alternate components for planning, control, grasping, or perception. Other ROS-based frameworks can be found in [56,57]. They both rely on ROS to integrate several functionalities and focus on behavior planning but using, respectively, a behavior tree and Prolog for knowledge-based reasoning.

2.4. ROS2: Other Applications

ROS and its further evolution were also analyzed for automated driving projects in [58]. In the automotive sector, even more than in robotics, at least two other aspects have gained relevance, namely communication between real-time systems (e.g., for driver assistance) [59,60] and active vehicle safety systems [61,62] that autonomous driving systems must take into account. The latter aspect is even more critical for two-, three-, or all-wheel-drive vehicles that require more complex behavioral models, as discussed in [63–65], respectively. Thus, the trend is to require software to have an increasing ability to integrate heterogeneous and real-time systems. Specifically, the AUTOSAR (Automotive Open System Architecture) standard software architecture [66] is proposed in the automotive sector

to eliminate dependence on hardware and increase software scalability. A comparative study of ROS2 and AUTOSAR as architectural platforms for future autonomous vehicles (AVs) was recently addressed in [67], pointing out that some AUTOSAR APIs and service functionalities can be completely fulfilled by ROS2, even though the standard implementation of ROS2 does not provide all the automotive functionalities and additional packages need to be added.

Furthermore, studies on ROS2 can also be found in other domains. For instance, the use of ROS2 for edge-cloud architectures was analyzed in [68]; in [69], an ROS2-based distributed control architecture for unmanned aircraft systems (UASs) was proposed; while in [70], a toolbox for distributed cooperative robotics named ChoiRbot is presented.

Although the still limited literature on ROS2 is mainly concerned with new aspects of ROS2, an analysis of the requirements and tools of ROS2 for increasing the autonomy of robots in industrial environments is not as thorough. An architectural and implementation framework for improving the autonomy and flexibility of industrial robotics using ROS2 is then provided below.

3. General Architecture for Enhancing Cobot Autonomy

Based on the above state-of-the-art analysis, here, a general high-level and modular architecture to enhance the autonomy of fixed-base cobots, by integrating environmental information provided by additional external sensors, is extracted and presented. Specifically, this section focuses on the main modules that this architecture should provide and on how they should be interconnected to achieve this goal, while Section 4 will show how to decline this general framework resorting to ROS2 as the middleware and taking advantage of the tools it provides. Therefore, here, general information about cobots is included, but this discussion serves a specific purpose, as it outlines the requirements and features that middleware must fulfill for the implementation of individual tasks and what aspects require attention.

3.1. Motivation

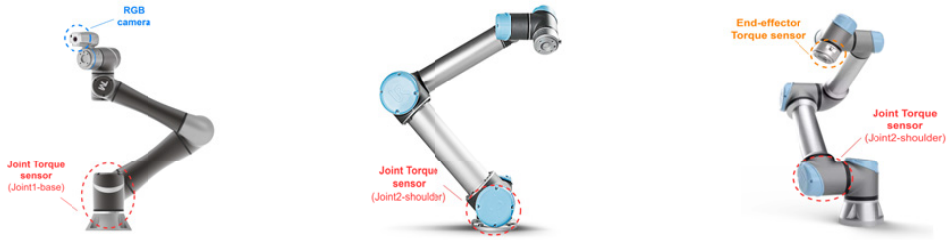
The reason behind the need for such an architecture is that the inherent characteristics of cobots alone are not yet sufficient to perform complex tasks in dynamic industrial scenarios that require a higher degree of autonomy. This is because industrial collaborative robots are currently on the market equipped with few exteroceptive sensors, and as a result, they cannot fully perceive the external environment and thus cannot be used at their full potential. These limitations are also due to the closed interfaces with limited functionality that most cobot manufacturers provide for users to control and program their products. They typically make available both a GUI (graphical user interface), very intuitive and user friendly but with standard predefined and non-customizable functionalities, and APIs (application program interfaces), specifically provided for further user-customized functionalities but to be properly integrated into the user application.

To better understand the limitations of the cobots currently on the market, the following is a brief overview of the main sensors they are equipped with and the typically available functionality enabled by them:

- Torque and force sensors: These are the sensors that allow collision detection and subsequent emergency stop of the robot. Typically, the GUI provided by commercial cobot manufacturers allows users to set safety stop criteria in terms of maximum values of parameters such as TCP (tool center point) force and joint torques. In this way, along with speed limitations, collisions are not anticipated and avoided, but they are made not dangerous. To handle them differently (e.g., with trajectory re-planning), different sensor types such as vision systems or distance sensors are needed.
- RGB camera: Most collaborative robots are also on the market with models equipped with an RGB camera mounted on the end of the manipulator (i.e., eye-in-hand arrangement). The integration of this vision system makes it possible to teach the robotic arm to recognize and grasp specific, identical objects from the same plane. Since depth

information is not available, it is necessary to manually reprogram the robot if the objects to be grasped or the plane on which they are located changes. Again, the use of different types of sensors (e.g., depth cameras) could overcome these limitations.

As a concrete example of the aforementioned, Figure 1 shows sensors integrated into some commercially available collaborative robots, from which it emerges that, while sensor types remain consistent, manufacturers can make different choices regarding which and how many sensors to integrate into their individual products.



(**a**) Techman TM5-900 (**b**) Universal Robot UR5/CB3 (**c**) Universal Robot UR5e

Figure 1. Examples of sensors integrated into commercial cobots.

3.2. Design and Requirements

A system to increase the autonomous capabilities of robots must necessarily be modular, as it must handle heterogeneous systems. Modules can be associated with the single, basic tasks that the robot must perform in order to increase its autonomy. Complex tasks require a high degree of interdependence between these modules. Like most autonomous systems, the main tasks to be performed by a fixed-base cobot with increased autonomy are shown in Figure 2 and described below:

1. Perception: The overall architecture needs to use sensors (being exteroceptive or proprioceptive) to gather the necessary information to become aware of the state both of the cobot and of the surrounding environment. This task is then responsible for acquiring and pre-processing these data. The robot state is usually available without the need for additional sensors in all commercialized cobots, which provide access to variables such as joint positions, speeds, and torques or the pose of the TCP. With reference to the latter instead, the often limited environment perception could be easily enhanced by resorting to vision, distance, and/or proximity sensors, such as depth cameras, LIDAR, and sonar. Moreover, this task can be performed by either using a single sensor or by the handling of several, even heterogeneous, sensors, whose data are integrated by means of sensor fusion algorithms to get a better single model of the surrounding environment.
2. Recognition: This task goes beyond mere perception, involving higher-level processing and interpretation of the raw data acquired by sensors. As an example, this is a task dealing with scene understanding, environmental reconstruction, and real-time object recognition/classification.
3. Behavior planning: This is a task responsible for identifying the type of behavior and/or the sequence of actions to be performed. It is potentially a highly advanced task that, however, can be approached at different levels depending on the desired level of autonomy. At a lower level, there is a fairly standard sequence of actions required to perform a specific and predefined task with minimal variations to handle unexpected situations such as obstacles. At the highest level, this allows the robot to autonomously understand the task to be executed based on the surrounding.
4. Trajectory planning: Another essential part of the robotic system is the path-planning algorithm that, based on the desired behavior of the cobot, generates a 3D path of the entire kinematic chain to move the end-effector from the initial pose to the desired final pose. The planned trajectory must be feasible, satisfying the physical constraints of the robot, as well as the constraints imposed by the environment due to

the presence of static obstacles (i.e., fixed and already present in the environment at the time of planning).

5. Trajectory re-planning: This represents a further fundamental planning module to increase the robot's autonomy in case of dynamic obstacles (i.e., moving obstacles that can appear within the scene at any time). It has two subtasks responsible for detecting collisions with dynamic obstacles and for avoiding them. Specifically, the first performs continuous collision checking between the cobot and the surrounding environment along the path, while the second re-plans a new path to avoid the detected collisions. This latter can be managed both with a global planner that plans a new path from the current state to the goal taking into account the updated environment and, more efficiently, with a local planner that updates the plan only around the potential detected collision.
6. Motion control: This is the low-level task responsible for executing the trajectory planned by the high-level tasks on the real cobot, controlling the robot's actuators to follow the planned trajectory. Alternatively, it can also handle the execution of this plan in simulation.
7. Manipulation: This refers to the ways robots physically interact with and modify the environment and objects around them with a wide range of actions such as grasping, packaging, assembling, or cutting, depending on the end-effector attached to the end of the robotic arm. All these actions require an appropriate control of the robot's end-effector. This is a task difficult to generalize because it heavily depends on the specific required manipulation action and on the type of end-effector used to accomplish it. As an example, the same grasping action for a pick-and-place task can be done both with suction cups or with a two-finger gripper. This choice, which should be based on the type of object to be grasped, determines a different tool control interface (e.g., suction/release commands instead of open/close) and a different grasp planning (e.g., suction cups require as contact point a flat facet near the object's center of mass to maintain stability, while a two-finger gripper requires two parallel facets at an appropriate distance).

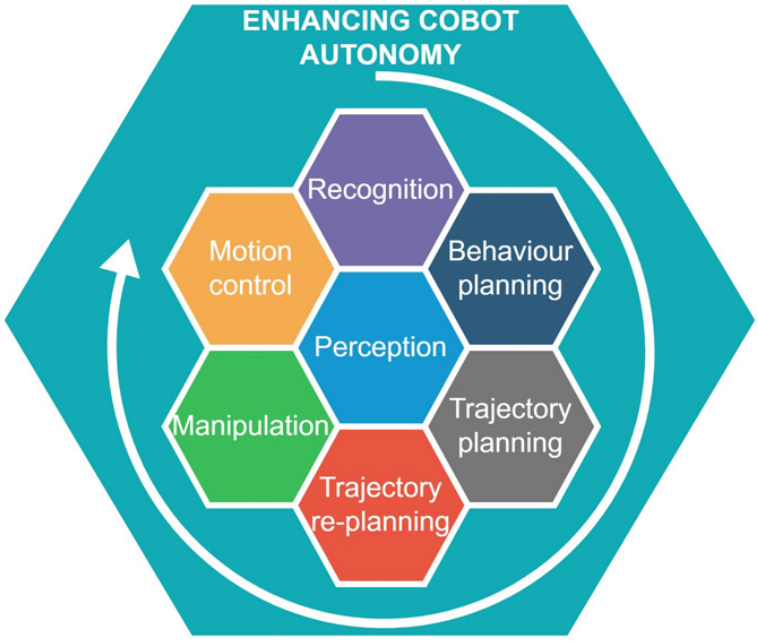


Figure 2. Main tasks for enhancing cobot autonomy.

Each of these tasks individually handles a specific problem, but none of them alone can ensure the desired enhanced autonomy achievable only by appropriately interconnecting

them, for example, as shown in Figure 3, in which task interconnection and consequent grouping into layers is proposed for a fixed-base robot.

In Figure 3, the starting layer is a perception layer resulting from the combination of the perception task with the recognition task; the latter, if necessary, interprets the raw data pre-processed (e.g., filtered and/or fused) by the former to extract the information needed for the execution of the other tasks.

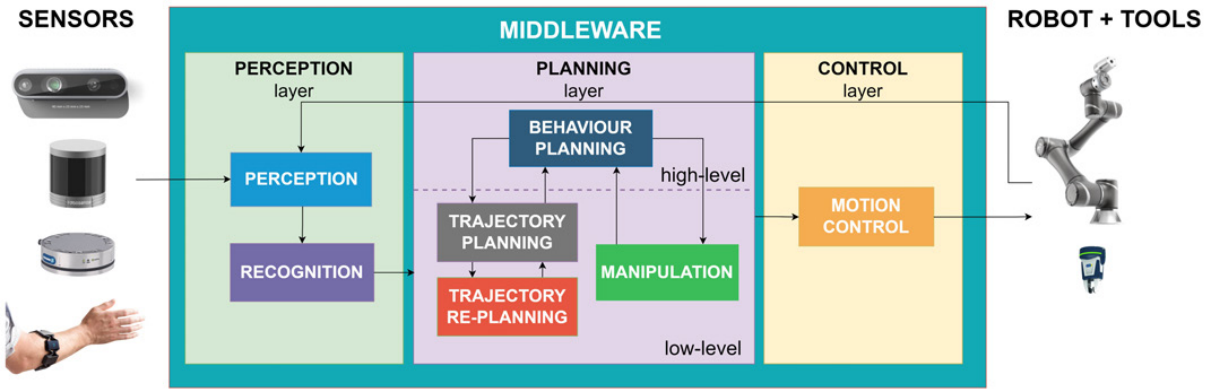


Figure 3. Main layers and module interconnections of a fixed-base robot.

The planning layer is the core layer of the architecture, responsible for making decisions on how the robot should behave at each level, from the type of action it should perform (the high-level behavior task) to the details of how the action must be executed in terms of motion of both the robot and its end-effector (the low-level planning, re-planning and manipulation tasks) adapting to the current scenario and the surrounding environment. This layer requires the information provided by the perception layer (and specifically by the recognition task) and provides as output a motion plan that must satisfy the robot's physical limitations, the perceived environmental constraints, and some goals that could be either predefined or externally requested or inferred from perception. This is the most complex layer responsible for the logic ensuring the desired higher level of autonomy.To ensure the highest level of autonomy, the behavior task should not be limited to a predefined sequence of actions depending on a fixed required goal, but it should use the perception to decide autonomously both the goal and the best sequence of actions to achieve it. Then, for each scheduled action, the lower-level planning is entrusted to the other three tasks and specifically to the planning and re-planning ones, whose close interaction allows the robot to react to static and/or dynamic obstacles, preventing collisions. Specifically, the planning task is ideally invoked only once to plan a feasible trajectory from the starting pose to the goal position required by the current action, while the re-planning task is executed continuously to monitor potential collisions and, if necessary, takes action to locally modify the trajectory, finally invoking the global planner again if the local re-planning fails. The proposed architecture also includes a manipulation task to emphasize the need for a particular action, such as manipulation. Even if this action could be managed by the other modules, a dedicated task should be allocated for it for its complexity and specificity (e.g., it concerns not only the planning of the robot but also of the end-effector). Further details on this topic, which will not be explored in this paper for the sake of brevity, can be found in [71,72].

The last layer is the control layer with the motion control task that ensures the execution of the plan computed by the planning layer. Where the robot and its end-effector are provided with high-level interfaces this task can be performed by a mere interface that sends commands to the robot and its end-effector; otherwise, the implementation of an appropriate feedback controller is required, which will also require the robot state provided by the perception layer.

3.3. Industrial Application Overview

Implementing at different levels the architecture described above, currently nontrivial or even impossible tasks for commercial cobots become feasible. This section briefly describes some of the more interesting industrial applications, focusing on those made possible by the implementation of a perception module that uses depth cameras to perceive the surrounding environment in 3D. More extensive surveys are available in [9,73,74]. For some specific applications, a single camera may be sufficient, but it could be affected by occlusion when certain areas in the scene are hidden from the camera's field of view, leading to incomplete depth information. Increasing the number of cameras is a simple and feasible solution, but it involves increasing the complexity of the perception layer [75].

Access to depth information about the surrounding environment and manipulated objects enables cobots to successfully complete their tasks despite dynamic environments and changing scenarios. Specifically, the recognition of the spatial pose of the object and of its size enables tasks such as flexible grasping [76–78] and flexible palletization/packaging [79]. While commercial robots are primarily trained to recognize and grasp specific, identical objects from the same surface (usually a flat surface), with additional information about the position and size of objects, these can be used to adapt tasks from static conditions to changing conditions (e.g., change in the type of object, its spatial location or palleting/packaging) without the need of manual reprogramming. This obviously requires adequate development of the perception layer, to perceive and recognize the objects with which the robot must interact, and of the planning layer (i.e., behavior, planning, and manipulation) to adapt the robot's behavior to changes in the properties of these objects. Similarly, the same depth information, but about the surrounding environment, is the key to enabling proper trajectory planning that can avoid static obstacles in an unknown and unstructured environment as well as online trajectory re-planning to deal with dynamic obstacles [77,80]. Again, the perception layer is essential for reconstructing the external environment, keeping it up to date, and for estimating the movement of dynamic obstacles, while the tasks of the planning layers involved are mainly planning and re-planning, which are responsible for managing static and dynamic obstacles.

Thanks to these enhanced capabilities, enabled by relatively limited additional hardware and a proper middleware implementation, cobots become capable of performing standard tasks but in a flexible manner, having the ability to adapt to changing scenarios and to react to dynamic environments. Moreover, this is also the basis for enabling more complex tasks, for example, in the field of human–robot collaboration (HRC), where direct interaction between the human operator and the cobot is required. An overview of the main industrial applications where HRC is advantageous can be found in [73], where standard tasks such as handling [81], welding [82], and assembly [83] are discussed. Besides these, there are also other less common tasks whose efficiency can be improved with HRC such as disassembly with HRCD (human–robot collaborative disassembly) [84]. In general, applications of this kind require a higher level of autonomy than simple human–robot Coexistence, in which the human and robot do not work together but only have to share the same workspace, and thus the human can be treated as a generic obstacle to be avoided. While this second scenario can be handled with the use of RGB-D cameras that provide both depth and image information, tasks requiring a more advanced level of HRC [8] require the integration of other sensors (e.g., wearable sensors [85]) and/or a more complex implementation of the layers described above [75,86], but can still be made feasible by adopting the architecture proposed here. In the following, and especially in the case study described in Section 5, we will focus only on the first type of these applications, i.e., on how otherwise standard tasks can be made flexible using additional depth information from an RGB-D camera.

4. ROS2-Based Architecture for Enhancing Cobot Autonomy: Current Capabilities and Limitations

Implementation of the modular architecture described above requires the use of middleware to easily integrate heterogeneous hardware (e.g., different sensors, robots, and end-effectors), as well as heterogeneous software components (e.g., those needed to accomplish individual tasks). For this purpose, several open-source robotic frameworks are available as alternatives to ROS, such as YARP (Yet Another Robot Platform) [87] for humanoid robots, OROCOS (Open Robot Control Software) [88] originally applied to both robots and drones [89,90], and MOOS (Mission Oriented Operating Suite) [91], applied to the domains of mobile robotics and marine vehicles; an overview of the frameworks can be found in [92]. Each of them has its own characteristics that make it more or less suitable for specific types of applications. For example, MOOS was originally developed to support operations with autonomous marine vehicles and continues to be primarily utilized in the field of marine robotics. OROCOS was created before ROS as a general-purpose framework for advanced machine and robot control, and it was the go-to framework when real-time requirements were in place until the development of ROS2. YARP is another framework older than ROS that has been mainly used in the domain of humanoid robots where skills such as visual, auditory, and tactile perception and legged locomotion are central, whereas ROS has a higher focus on mobile robots and provides more tools on navigation and planning. Moreover, even if both of them are under active development and provide features for general-purpose robots, ROS supports more robots and remains the most widely used framework among all of them. As a result, ROS, and in particular ROS2, is proposed as the most suitable middleware to implement the framework described in Section 3 not only because of its spread, but also because it provides the features previously described in Section 2, such as the structured communication layer and hardware abstraction, along with libraries, tools, and utilities that implement state-of-the-art algorithms and facilitate functionality (e.g., visualization and debugging) useful to rapidly build robotic applications. This choice is also motivated by the large and active ROS community that contributes to its development and maintenance, as well as to the creation and distribution of packages and resources solving classical problems encountered in robotics applications, including some of those that the proposed architecture aims to address. The disadvantage of this choice lies in the fact that ROS2 is relatively recent (the first Ardent distribution was released in 2017). It has not yet fully replaced its previous version (to which henceforth we will refer as ROS1 for clarity, using instead ROS to indicate the middleware in general and regardless of the specific version), and new applications are still being developed in ROS1 because the migration of all the components implemented in ROS1 to ROS2 is ongoing and documentation about it is still lacking.

This section is devoted to analyzing how ROS2 can be used to develop the proposed general high-level architecture, showing what is currently implemented and available in ROS2 for the development of each layer and task and what their potential limitations are, with an emphasis on similarities and differences from ROS1. The general idea is to have a single ROS2 application, in which each module in Figure 3 is implemented by one or more nodes grouped in one or more packages and communicates with others using the standard communication mechanisms provided by ROS. Specifically, each module exposes as an interface a set of ROS topics, services, and/or actions to offer information and/or functionality to other modules and to gather the necessary data and/or request services from other modules. In the following, each module is analyzed independently while also dealing with its interfaces exposed to the rest of the ROS application.

In detail, only the basic tasks necessary to achieve a general level of autonomy that is not strictly task-specific will be explored. Both behavior planning and manipulation, which are the most task-dependent and the least generalizable in ROS, will therefore not be thoroughly investigated. They will have to be the subject of further specific investigations. Indeed, as far as the manipulation task is concerned, ROS offers specific functionalities worth exploring, but this problem deserves a dedicated investigation given its extent and

complexity (see, e.g., [72]). Similar reasoning applies to behavior planning, at least when it is addressed at the highest level and is not limited to predefined sequences of actions (see, e.g., [93]), as will instead be addressed later.

4.1. Perception

This module is responsible for gathering from the robot and other external sensors the raw data needed to know the state of the robot and its surroundings. As for the data to be acquired directly from the robot, manufacturers usually allow access to the robot's state, but there is no standard either on the type of information accessed or the interfaces exposed; they often provide more than one way to obtain the same data. For example, in the aforementioned OMRON Techman TM-series cobot, variables such as joint angles, speeds and torques, or TCP pose, can be read either using the Modbus protocol [94], accessing dedicated registers, or by connecting to a dedicated TCP socket server (i.e., *Ethernet Slave*) that allows clients to read the variables specified by the programmer in the dedicated *Data Table* and use them for different purposes. For example, in [95], it was used to perform cobot fault detection by applying signal-based diagnosis [96], while in [97] it was used for anomaly detection of abnormal cobot behavior. On the other hand, the same cobot does not make the data from the integrated eye-in-hand RGB camera accessible from the outside. As a result, generally, the acquisition of the robot's state cannot be entrusted to standard ROS packages, and this submodule of the perception task must be handled by a specific low-level module that serves as the cobot driver, the same module that is also responsible for sending commands to the physical robot and that will be used in the Section 4.5. Apart from this hardware-specific part, which is not necessary when working in simulation, ROS2 provides tools for managing and using the robot's state that is essentially unchanged from ROS1. While the use of this information will be addressed in later modules, for the perception task only, the only aspect to highlight is the need to have an */joint_states* topic, which is the standard ROS interface for publishing the current state of the robot (real or simulated) [98]. This is an ROS channel where standard *sensor_msgs/msg/JointState* messages are sent containing the state of a set of joints (rotating or prismatic) in terms of position, velocity, and torques applied to the joint. This real-time information, combined with the prior knowledge of the physical parameters of the robot (i.e., dimensional, geometric, kinematic, and dynamic parameters) provided to ROS with the unified robot description format (URDF) [99], is all that is needed to have complete information of the robot's state in terms of pose, as will be shown in the recognition module. In summary, the only thing to be handled in this module, as far as this first part of perception is concerned, is to ensure the publication of the subject */joint_states*. The publisher should be the robot driver of a real manipulator or the simulator (e.g., Gazebo [100]) in a simulation scenario.

Regarding the perception entrusted to additional external sensors, the same concept applies, with ROS/ROS2 providing standard interfaces that must be implemented by the low-level modules handling real/simulated sensors. Again, sensors from different manufacturers, even more if of different types, require hardware-specific drivers that can be already available (sometimes provided by the same vendors) or to be developed. Despite the specific implementation, what should be standard in order to exploit existing ROS functionality are the topics where sensor data are published or rather their message types. In ROS2, this is ported from ROS1. *sensor_msgs* is a standard ROS package containing the definition of many messages relating to sensor devices such as cameras, inertial measurement units (IMUs), and laser range-finders [101]. For example, for an RGB camera, there is *sensor_msgs/msg/Image*, a message type intended to contain an uncompressed image with a header with fields such as timestamp, *frame_id* of the camera, image dimensions in pixels and encoding (e.g., *rgb8*), and a body that is the actual data matrix. Similarly, for a depth camera there is the message *sensor_msgs/msg/PointCloud2* (previous *PointCloud* is deprecated since ROS2 Foxy) with information including timestamp, *frame_id*, dimensions in pixels (point-cloud data may be organized 2D, i.e., image-like, or 1D, i.e., unordered), number of fields with their dimensions (e.g., x, y, z coordinates and RGB value) and the

actual point cloud. The implementation of these topics allows, for example, the immediate visualization of these RGB and depth data in RViz2 (i.e., the default 3D visualization tool for ROS2) as shown in Figure 4. Just selecting the desired standard topic enables RViz to subscribe to it and to display its data in real time.

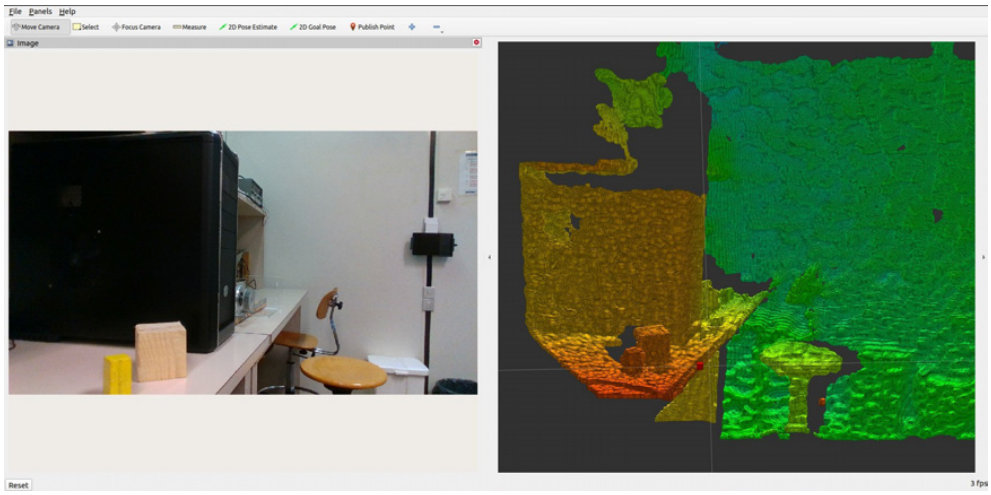


Figure 4. Visualization of *Image* and *PointCloud2* messages in RViz2.

The last aspect to analyze is the eventual pre-processing of the acquired raw data that could make use of state-of-the-art algorithms already implemented in ROS. Regarding this, the porting to ROS2 is still ongoing, and it is still not uncommon to find algorithms working in ROS1 that are not yet available for ROS2. As an example, the *imu_filter_madgwick* is a package available in ROS1 to fuse angular velocities, accelerations, and optionally magnetic fields from a generic IMU into an orientation given as a quaternion and to publish the result on a *sensor_msgs/msg/Imu* topic that was ported to ROS2 Eloquent only in 2020. Thus, summarizing, the main advantages of using ROS for the perception module are as follows: standard interfaces allowing, once implemented, to easily replace sensors (e.g., switching from a virtual sensor to a real one), and the availability of open and customizable packages to solve standard pre-processing problems and utilities such as RViz. On the other hand, the main limitation lies in the need for specific and not always available drivers to handle equally specific hardware, but this is a problem inherent to the absence of standardized interfaces on cobots and sensors that is not caused by ROS.

4.2. Recognition

The recognition module, being in charge of further processing the pre-processed data published by the perception module to extract all the information required by the following tasks, is highly dependent on the features planned for the other modules. Limiting this analysis to the application on which this work focuses (Section 3.3), recognition is related to the following aspects:

- Cobot state;
- Environment reconstruction;
- Object recognition/classification.

Concerning the state of the cobot, as anticipated, there is a standard ROS package, the *Robot State Publisher*, responsible for publishing the state of the robot, thus making it available for all the components [99]. Specifically, it retrieves the kinematic tree model of the robot from the URDF contained in the parameter *robot_description*, updates it with the joints state by subscribing to the */joint_states* topic, calculates the resulting 3D poses (position and orientation given as a quaternion) of all the links with the forward kinematics, and publishes them to tf2 using the */tf* topic dedicated to the movable joints. Tf2 is Transform Library Version 2 (previous tf version is deprecated since ROS1 Hydro), an ROS package

dedicated to keeping track of multiple coordinate frames over time in a tree structure and to allow the user to easily perform transformations between these coordinate frames at any desired time, as long as they are connected in the tree. This last package should be used to handle not only the robot frames but also the coordinate systems of sensors such as cameras, where knowing the image reference frame is crucial for its correct interpretation. The difference is that sensors often involve static coordinate transformations related to fixed joints that are handled by a different node, *static_transform_publisher*, published on a different topic, the */tf_static*, but with the same message type. As an example, Figure 5 shows how RViz2 visualizes all the frames in the tree structure, including the one related to an external camera mounted on an external support.

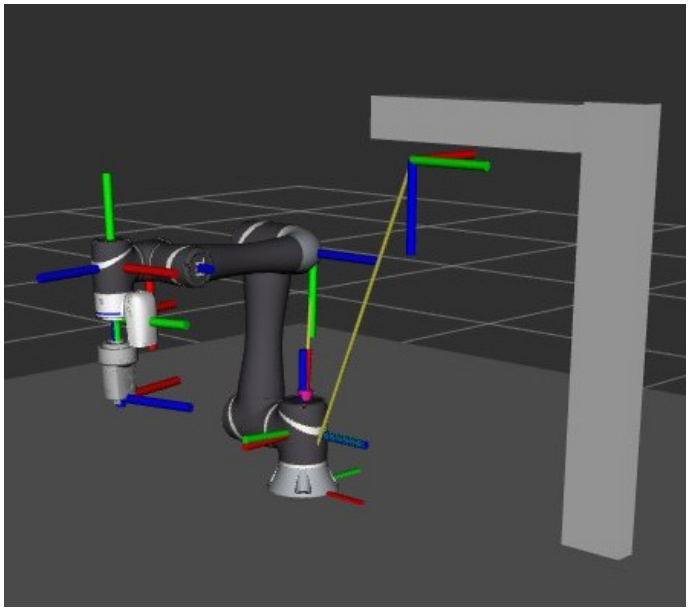


Figure 5. Visualization of the coordinate frames in RViz2.

More complex but still largely handled by ROS is the environment reconstruction. ROS provides several functionalities to automatically import information retrieved by the sensor into the virtual 3D scene where the robot is modeled and that is used for the trajectory planning [102]. Restricting again to our specific application in which the perception of the environment is entrusted to a single depth camera in an eye-to-hand arrangement, the problem is how to update the 3D environment surrounding the cobot using data provided by an RGB-D camera with the *sensor_msgs/msg/PointCloud2* type messages described in the previous section. On ROS2 this issue is handled by the *PointCloud Occupancy Map Updater*. This is a plugin of MoveIt2 (see Section 4.3) that takes as input the standard point-cloud message mentioned above and uses it to update the representation of the external environment, knowing the pose of the camera reference frame in the world. In doing so, the plugin can also eventually apply self-filtering to identify and remove the robot itself from the point cloud. This process is also called self-identification and is necessary when the camera is used in an eye-to-hand arrangement that provides a wider viewing angle, but involves the problem of segmenting the robot from the environment during the point-cloud processing. This segmentation is required in order to remove the robot shape from the 3D scene and thus to avoid the robot itself being considered as an obstacle [103]. This can be accomplished using the robot state (i.e., the above-described combination of URDF and joint states provided by this module) to know its occupancy space. Figure 6 shows the effect of this kind of filtering by placing side by side the RGB image, the original point cloud published by the camera driver, and its filtered version published on a different topic by the *PointCloud Occupancy Map Updater* mainly for debugging. Note that in the last figure, the color scale is associated with the distance of the framed object from the camera.

(**a**) RGB image (**b**) Point cloud (**c**) Filtered point cloud

Figure 6. Processing the point cloud with self-filtering.

Once the filter is applied, the new filtered point cloud is used to build a 3D occupancy grid using OctoMap, an ROS library based on octrees and on probabilistic occupancy estimation [104]. This map, composed of fixed-sized cubes (and settable size) called voxels that explicitly represent not only occupied space but also free and unknown areas, can be used to keep the *PlanningScene* subcomponents related to the external environment up-to-date. The *PlanningScene* is the object used in ROS for storing the representation of both the world around the robot and the state of the robot itself, which are needed to perform collision checking and to compute motion plans (see Sections 4.3 and 4.4). Figure 7, on the other hand, shows the operation of the *Planning Scene Monitor*, which is a standard ROS2 component, belonging to the MoveIt2 framework (see Section 4.3) but ported from ROS. It is responsible for maintaining, updating, and publishing the state of the *PlanningScene* using different sources of information, including the robot state and the occupancy map [102]. In particular, the updated *PlanningScene* is published on the topic *monitored_planning_scene* having type *moveit_msgs/msg/PlanningScene* that is heavily used by the planning and re-planning modules.

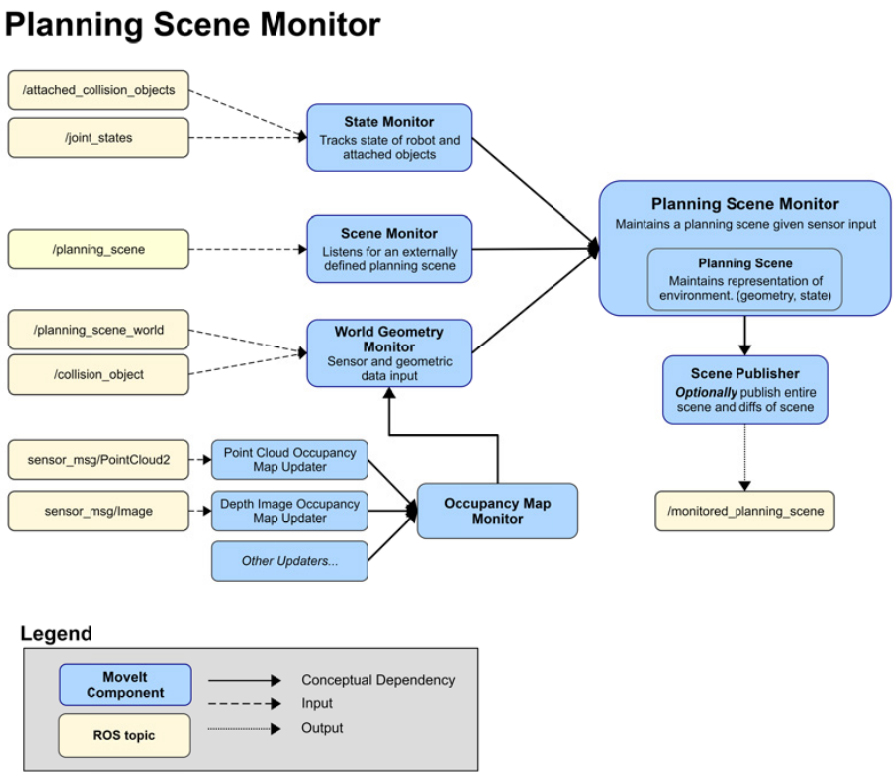
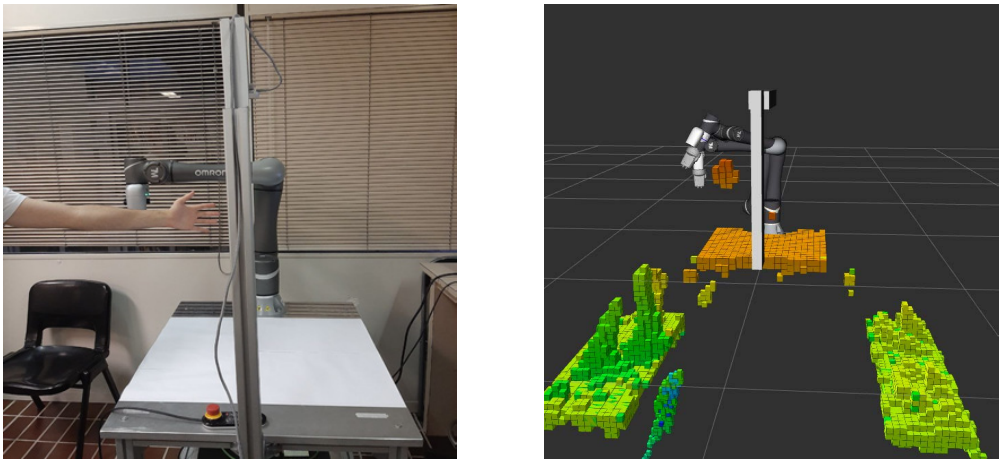


Figure 7. Planning scene monitor [105].

As an example, Figure 8 shows a *PlanningScene* reconstructed by combining the prior knowledge of the environment, the state of the robot, and the occupancy map extracted from the filtered point cloud obtained from the external camera, as described above.

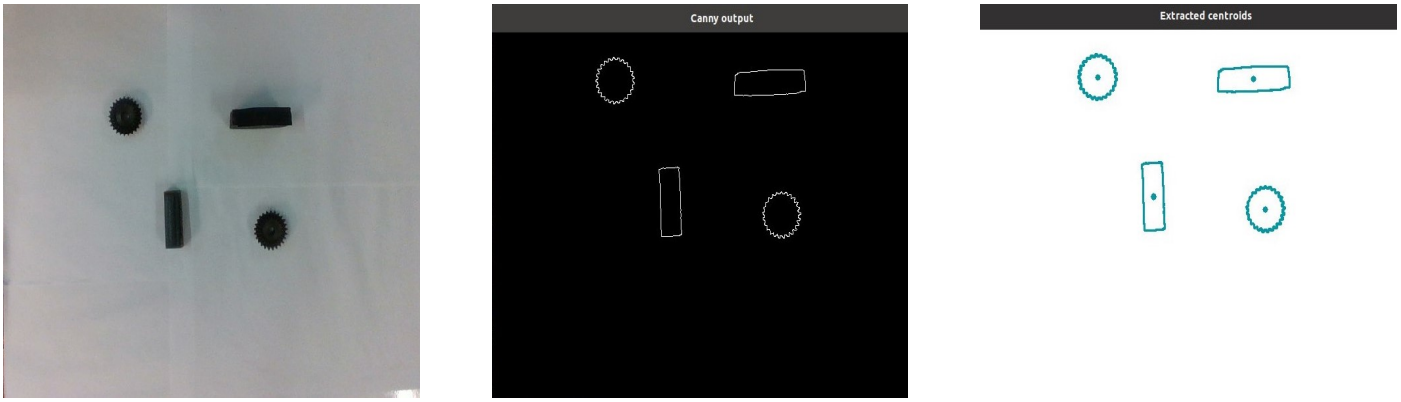


(**a**) Real scene (**b**) Planning scene on RViz2

Figure 8. Reconstruction of the planning scene.

Finally, the problem of object detection must be addressed; it involves using the RGB image provided by the camera to classify and recognize specific objects in the scene with which the robot must interact (e.g., to grasp or avoid them), or of which it must be aware of the presence (e.g., markers or human workers). This problem, unlike the previous ones, does not have a standard solution even in ROS1. However, there are unofficial GitHub repositories that deal with it, such as *darknet_ros*, an ROS package (also tested for ROS2 Foxy) for using YOLOV3 (You Only Look Once Version 3 software library) within ROS [106]. YOLO currently represents a state-of-the-art library for object detection that is known for its speed and accuracy. This library applies a single fully convolutional neural network (CNN) to the entire RGB image to detect and classify objects on which its model has been pre-trained. In particular, the repository [106] makes it easy to configure the trained model as well as the topic from which images will be taken, and it publishes detection results on dedicated topics in both image and list form. Already standard, on the other hand, is the interfacing of ROS with OpenCV, an open-source software library for computer vision, machine learning, and image processing written in C/C++ also having interfaces in Python and Java [16,17]. In ROS1, there exists *vision_opencv*, composed of the packages *cv_bridge* (i.e., the bridge between ROS image messages and image representation used by OpenCV) and *image_geometry* (i.e., a collection of methods for dealing with image and pixel geometry), this is the standard that implements this interface, and it is currently reported that it is partially ported to ROS2, although the main features are already available. Figure 9 shows an example of image processing to compute the centers of contour/shape regions thanks to the functions provided by OpenCV, such as *apply_canny*, which implements the Canny algorithm for edge detection [107], and *findContours*, which finds curves on edge joining all the continuous points with the same color or intensity.

In summary, ROS2 makes available most of the features required by a recognition module, thanks to the abstraction introduced by the perception module, although not everything has yet been ported from ROS1, and currently implemented algorithms can be improved and/or easily replaced thanks to the modularity of ROS.



(**a**) RGB image (**b**) Edges detected by Canny (**c**) Contours and centers

Figure 9. Example of image processing with OpenCV.

4.3. Trajectory Planning

In ROS2, the trajectory planning module, which is responsible for planning a feasible global trajectory to move the robot from a starting point to a goal pose, can be implemented by leveraging MoveIt 2, the direct porting in ROS2 of the previous version MoveIt [102]. It is a set of ROS packages that constitute a motion planning framework that provides APIs and graphical tools to help solve standard problems relating to motion planning, collision avoidance, and also manipulation. Its main strength lies in providing both off-the-shelf solutions that implement state-of-the-art algorithms and a general framework that can be customized by developing its own algorithms. The core of this framework is the node *move_group*, responsible for pulling all the individual subcomponents together and for interfacing them with the external world, encompassing, on the one hand, the cobot and additional sensors (i.e., perception, recognition, and motion control modules) and, on the other hand, those requiring MoveIt services/actions, i.e., the user, through a dedicated GUI on RViz2, or the rest of the ROS application (e.g., behavior planning module) through dedicated interface packages. A simplified schematic of MoveIt2 architecture is shown in the Figure 10, illustrating how MoveIt services (i.e., user interface) can be called, how the framework acquires the state of the robot and environment and requests plan execution to the robot controllers (i.e., robot interface), and some of its main subcomponents. Concerning the latter, we can observe the *Planning Scene Monitor* described in the previous section, the *Collision Detection*, which is responsible for the robot's self-collision and environment collision checks, and the *Planning Interface*, which takes charge of the motion plan request to move the robot from a start pose to a different one (expressed in joint space or operational space) and responds with a motion plan result that satisfies all the imposed constraints. Both the *Collision Detection* and the *Planning Interface* components rely on external libraries to implement collision detectors and motion planners. Specifically, MoveIt provides default libraries that allow the framework to be easily and quickly tested, which can be easily customized and/or replaced, making MoveIt highly adaptable. The default motion planning library is OMPL (Open Motion Planning Library), implementing many state-of-the-art sampling-based algorithms such as RRT (Rapidly Exploring Random Tree), PRM (Probabilistic Roadmap Method), EST (Expansive Space Trees), and KPIECE (Kinodynamic Motion Planning by Interior-Exterior Cell Exploration) [108,109]. Other available planners are CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [110], STOMP (Stochastic Trajectory Optimization for Motion Planning) [111], and Pilz Industrial Motion Planner. Concerning collision detectors, the default is FCL (Flexible Collision Library) [112], while an available alternative if continuous collision detection is required is the Bullet library [113]. Finally, it should be noted that, while modification of the motion planner is relatively straightforward, especially if working with OMPL, replacement of the collision detector requires modification and recompilation of MoveIt2 packages.

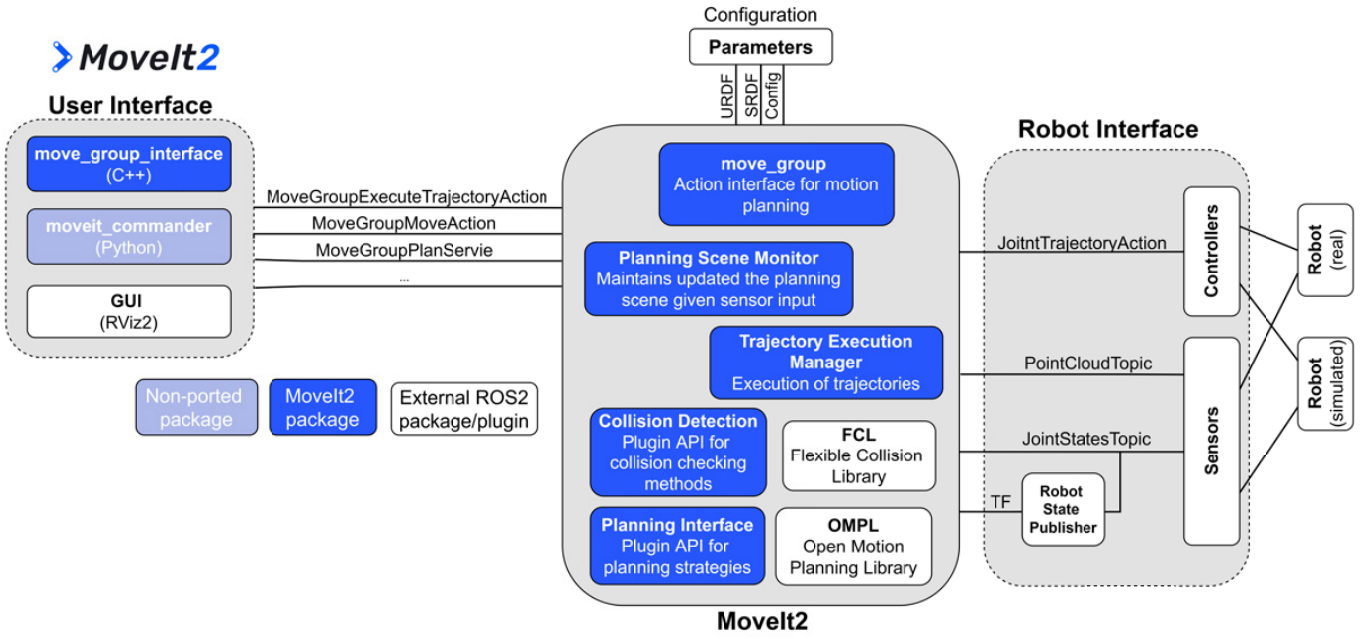
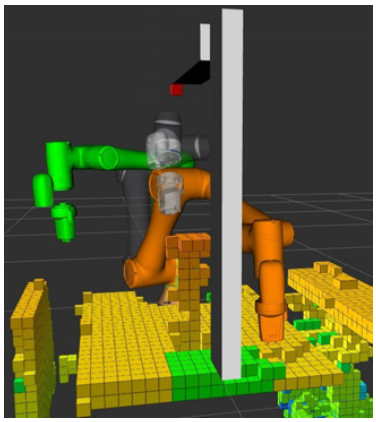


Figure 10. MoveIt2 architecture.

In conclusion, ROS2 through MoveIt2 offers a powerful framework for motion planning that, besides being highly modular and customizable, even in its standard implementation, offers advanced features such as planning to avoid collisions with the external environment perceived by an exteroceptive sensor like a depth camera. As an example of this, Figure 11b shows how MoveIt2 plans a feasible trajectory (in gray a snapshot) from a start state (in green) to a goal state (in orange) in an environment reconstructed in the form of an OctoMap from the point cloud acquired by a camera (the red voxel) mounted on an external support above the table. The real environment and the camera positioning can be seen in Figure 11a. This trajectory planning was achieved by using the default configuration of MoveIt2, i.e., FCL as the collision detector and RRTConnect as the motion planner from OMPL, all of which is shown by taking advantage of the MoveIt plugin for RViz that allows, among other things, the planned trajectory to be graphically displayed before executing it on the real or simulated robot.

(**a**) Real scene



(**b**) Trajectory planned in reconstructed scene

Figure 11. Example of trajectory planning.

4.4. Trajectory Re-Planning

The trajectory re-planning module is responsible for detecting collisions with dynamic obstacles and for modifying the plan to avoid them. It requires more advanced features than

the previous module because real-time requirements must also be met. In principle, without these constraints, dynamic obstacles could be handled as static ones, taking advantage of the MoveIt2 functionalities described in Section 4.3. During the execution of the trajectory, initially planned by the previous module, the planning scene is kept up to date by the *Planning Scene Monitor*, and this information is used by the *Collision Detection* to check for collisions between the updated scene and the first subsequent steps of the planned trajectory. If there is an impending collision, the motion planner is invoked again on the new scene to generate a new trajectory from the current state to the initial goal. This type of approach is inapplicable in practice, unless the obstacles are particularly slow, because scene updating, collision detection, and motion planning are time-consuming operations. Despite this huge limitation, this was the only available re-planning strategy in MoveIt [114] and is still the default in MoveIt2 where, however, a new general approach, *Hybrid Planning*, is being introduced [115]. This is an already well-established approach in the ROS community focused on navigation (existing since ROS Navigation Stack for ROS1) that relies on combining heterogeneous motion planners with different speeds and scopes in order to make the robot responsive to dynamic environments. Specifically, the idea is to rely on a slower global motion planner to generate an initial offline motion plan and on a faster local motion planner to replan when this global solution is invalidated by the appearance of a new obstacle in the robot's surroundings. The global planner works following the standard MoveIt pipeline described in Section 4.3. The novelty concerns the local planner. It is invoked when the scene is updated due to the appearance of dynamic obstacles to modify only the global plan, adjusting it on the fly according to real-time scene information. Because of this, the local planner must meet opposite requirements compared to the global one, such as determinism, real-time, and low computation time, while completeness is not required. As a result, the local planner implements different algorithms, such as potential field planner or model predictive control (MPC). If local re-planning fails, the global motion planner can be triggered again to achieve the desired goal. Thus, the hybrid planning approach also requires an *Hybrid Planning Manager* to invoke and coordinate the planners based on a customizable event-based logic, as can be seen from the hybrid planning architecture shown in Figure 12. A noteworthy aspect of this architecture is that the local planner implementation is based on two plugins: the *Trajectory Operator Plugin*, which monitors the global reference trajectory and determines the local problem, and the *Solver Plugin*, which solves it based on the local constraints.

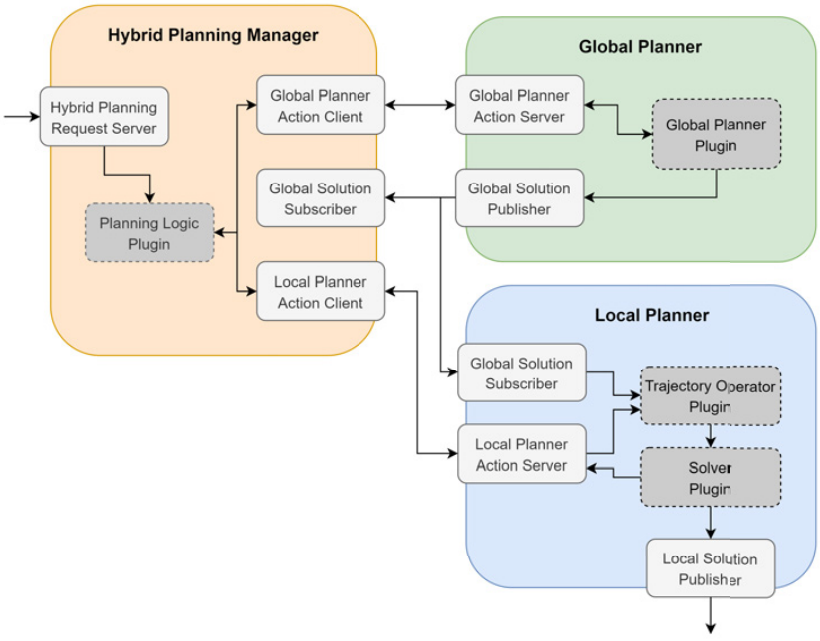


Figure 12. Hybrid planning architecture.

Like everything else in ROS, this architecture is designed to be highly customizable with each component (i.e., *Planning Logic Plugin*, *Global Planner Plugin*, *Trajectory Operator Plugin*, and *Solver Plugin*) that can be completely replaced by a customized version as long as it offers the required interfaces. At present, both the above-described architecture and an example demo using currently available plugins are implemented in the *moveit_hybrid_planning* package belonging to the main version of MoveIt2. However, there are not many applications using this new approach that can still be refined. This is therefore one of the modules with the greatest potential for further development.

4.5. Motion Control

The last module to be investigated is the low-level module that handles the execution of the plan on the real/simulated robot. This module should also act as the driver of the robot, providing not only the interface to send commands to the robot and/or to its end-effector to execute the plan, but also to obtain their status in order to expose it as required by the perception module. However, since the robot state acquisition has been already addressed in the perception module, here, we will focus on control, but the concept is the same. As anticipated in Section 4.1, at least when dealing with real robots, a specific driver is needed for specific hardware because there are no standard interfaces and different manufacturers expose different and multiple ways to send commands to their products. As an example, the already-mentioned OMRON TM-series cobots support an *External Script* mode, through which the cobot can receive motion commands from the outside using another socket TCP server with the dedicated application-level protocol *TMSCT*. In this case, the motion control module must take care of translating the plan computed by the higher-level modules into a sequence of commands recognized by the robot and sending them out. Therefore, it is clear that the part of this module responsible for the communication with the robot must be hardware-specific. What can and should be standardized to use existing ROS functions, as with the perception module, and are the interfaces exposed to other modules? Specifically, the motion control module must offer the action server invoked by MoveIt controllers to request the execution of the planned trajectory. The type of this action must be specified in one of the MoveIt configuration files (i.e., *controllers.yaml*), along with the controllers [102]. One of the most used actions is *FollowJointTrajectory*, a standard ROS action defined in the MoveIt package *control_msgs* in order to require the execution of a trajectory specified in the joint space in terms of either positions, velocities, and accelerations, or positions and effort. Alternatively, other types of controllers are available that deal with single-point control and are divided into three groups according to the command type in output (i.e., *position_controllers*, *velocity_controllers* and *effort_controllers*). For example, in the last group, the controller *joint_position_controller* receives a position as input and sends an effort as output to the hardware interface using a PID controller. The choice of the specific controller must be taken based on the interface available on the specific robot and must be followed by its implementation on the robot driver.

5. Example of Using the Framework to Increase Cobot Autonomy: A Proof of Concept

In this section, the general architecture described in Section 3 is applied to a specific case study in order to show with a practical example how the ROS2 features analyzed in Section 4 can be exploited. The main objective is to enhance the autonomous capabilities of a fixed-base cobot using a single external depth camera to enable it to autonomously perform a predefined task (e.g., pick and place) in a dynamic and changing environment. Taking advantage of the visual and depth information provided by the supplementary camera, an ROS2 application was developed to make the cobot able to adapt the grip to different objects to be moved and to adapt the trajectory to the presence of static obstacles not known in advance. This level of autonomy cannot be achieved just by using standard GUIs provided by manufacturers, because manual reprogramming would be required for each change in environmental or working conditions.

5.1. Hardware and Software Setup

The proposed case study is applied to an OMRON TM5-900 cobot, a 6-DOF (degrees of freedom) collaborative manipulator of the TM-series produced by Techman Robot and distributed by OMRON Corporation. This cobot has only an integrated RGB camera and is a good example of a typical cobot whose degree of autonomy can be improved by using ROS2 to implement the proposed architecture. For the pick-and-place task execution, the cobot has been equipped with a two-finger gripper for small components with an I/O digital interface, specifically the Coact EGP-C from Schunk. The environment perception is entrusted to a depth camera, namely an Intel RealSense Model D435i, a compact device integrating, among other things, an RGB camera, a stereo camera, and an onboard processor. This stereo camera can provide image frames with a maximum resolution of 1280 × 800 pixels at a maximum frame rate of 30 fps and has a field of view (FOV) of 87° × 58°. For this proof of concept, we choose to use a single camera in an eye-to-hand arrangement, mounting it on an external support above a table where the arm is fixed and at a height of about 81 cm to obtain an overhead view of the cobot's work area. Figure 13 shows the hardware setup arranged for the proof of concept and highlights with dashed colored lines the individual components just described. As for the software configuration, the ROS2 application was developed using the ROS2 Foxy distribution on a PC with Ubuntu 20.04 installed, which is the LTS (long-time support) version compatible with the chosen ROS version.

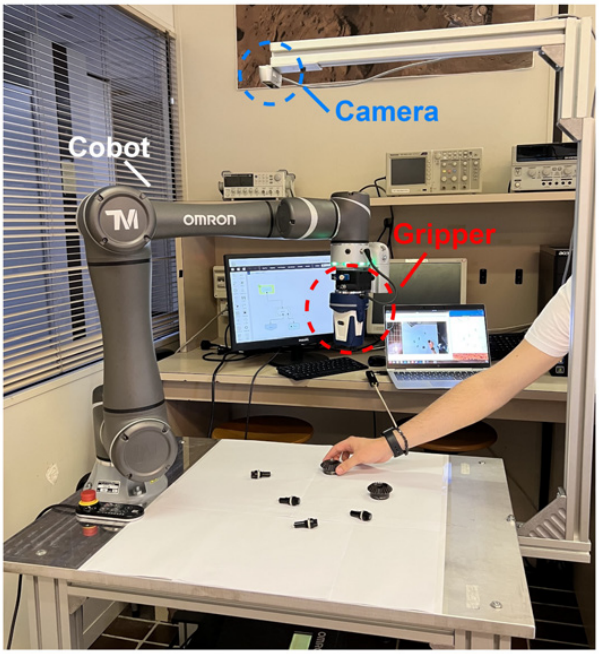


Figure 13. Hardware setup.

5.2. Software Architecture

To implement the proposed case study, the general architecture in Figure 3 was declined into a specific software architecture as shown in Figure 14. As mentioned earlier, only the core modules analyzed in Section 4 were developed, leaving out not only the more advanced tasks of planning and behavior manipulation, whose basic logic is embedded in the planning task, but also re-planning. In fact, for this preliminary proof of concept, only static obstacles were considered because the main objective is not in the planning logic but in showing a practical example of how the overall architecture can be implemented using ROS2 on a real system, thus building the basic infrastructure on which future applications will be developed and offering a proof of its potential.

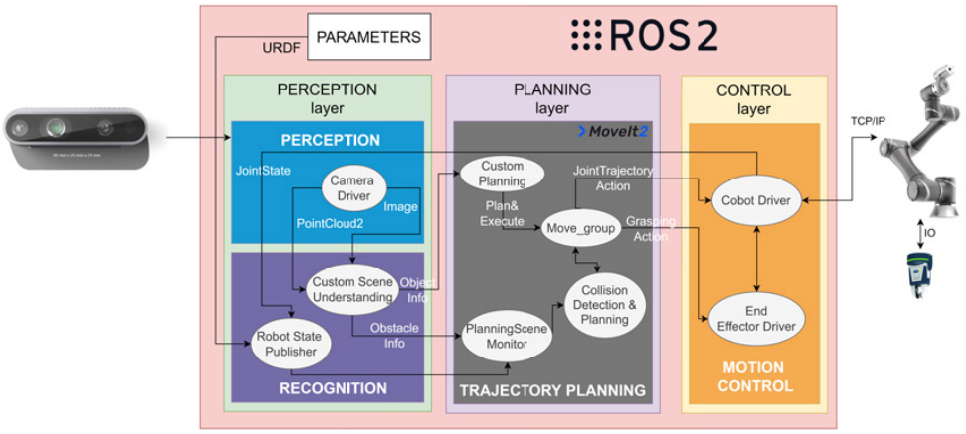


Figure 14. Software architecture.

With this in mind, the developed ROS2 application exhibits the behavior represented by the block diagram from Figure 15 and described in the following. The cobot initially moves to a predefined position (referred to as *Photo_Position*), chosen so as not to obstruct the camera's field of view. Then, the camera scans the scene to find the object to be grasped and the static obstacles to avoid. Once their positions and sizes are calculated, the robot first performs the pick operation, reaching a target pose based on the retrieved object information, followed by the placing operation, which is planned to navigate around obstacles that may obstruct its path.

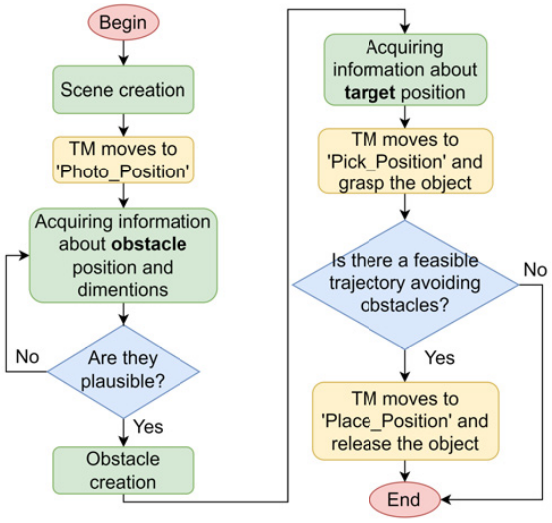


Figure 15. Software flowchart.

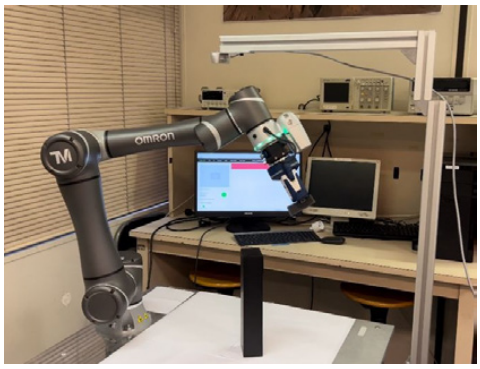
The following subsection will explain the implementation of each layer in the software architecture depicted in Figure 14 in order to achieve the desired behavior.

5.3. Perception Layer Implementation

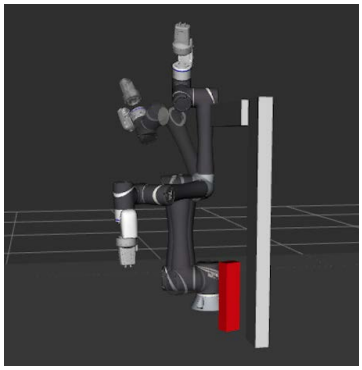
The perception module is the starting point of the implementation. It is composed of two nodes: the *Camera Driver* for the environment perception and the *Cobot Driver*, which, in this layer, is responsible for the cobot state acquisition. The first exploits ROS2 packages provided by camera manufacturers in a GitHub repository to use the Intel RealSense D400 camera series [116], to which the D435i model used here belongs. This driver was used both as an interface for the depth camera, to easily configure it with a list of available parameters, to acquire RGB and depth images, and as an interface for the rest of the application, to publish these data in the standard ROS topics *Image* and *PointCloud2*. The example in

Figure 4 is obtained using this *Camera Driver* with the Intel RealSense D435i camera. The *Cobot Driver* node instead implements the interface towards the TM5-900, and also, in this case, it leverages the dedicated ROS2 driver provided by the cobot manufacturer as a GitHub repository [117]. Among the functionalities offered by this driver, there is the publishing of the standard *JointState* topic, based on joint data acquired from the cobot using the already-mentioned socket TCP server *Ethernet Slave* with the dedicated application-level protocol *TMSTA* once the *Data Table* is appropriately defined from cobot side. Therefore, the implementation of the perception module became quite simple due to the availability of ROS2 Foxy-compatible drivers for both the camera and the robot. Otherwise, it would have been more complex, given the need to develop them from scratch.

Of further interest is the development of the recognition module, at least as far as the environment around the robot is concerned. In fact, the state of the cobot is handled in the standard way already discussed by its driver, which also incorporates the URDF model of the robot. Environment reconstruction, on the other hand, is not handled by the MoveIt plugin in *PointCloud Occupancy Map Updater*, but is performed by manually adding obstacles to the planning scene based on their size, orientation, and center position information that can be extracted from camera images once their shape is detected with OpenCV libraries. Indeed, OpenCV is used to process the RGB image acquired when the robot is in *Photo_Position* firstly to find the obstacle contours, using the already-mentioned *apply_canny* and *findContours* functions. Subsequently, both the pixel dimensions and orientation of its smallest bounding rectangle are calculated with the *minAreaRect* function, while the pixel corresponding to the obstacle centroid is obtained using image moments given by a different OpenCV function (i.e., *moments*). The position of the obstacle's center in the cobot reference frame is calculated from the pixel coordinates of the detected centroid, first determining the spatial position x, y, z on the camera frame of the corresponding pixel in the point cloud, and then using Tf2 to change the reference frame. A more complex task is to find the size and orientation of the obstacle from the bounding rectangle returned by the OpenCV function. Regarding sizes, it is necessary to convert them from pixels to a distance measure (e.g., centimeters) using both the information about the obstacle's distance from the camera (i.e., the z coordinate of its center position in the camera frame) and the size (centimeters) of the single pixel in the image acquired at a predefined distance. Finally, the orientation of the obstacle in the cobot's frame is obtained by the angle of rotation of the bounding rectangle returned by the same *minAreaRect* function (i.e., the angle between the rectangle base and the horizontal axis). In this way, all the information is available to update the initial scenario by means of *PlanningScene*, including only the fixed obstacles (i.e., table and camera support), where the detected static obstacles were added to the scene as *CollisionObject*. Figure 16 shows an example of the updated *PlanningScene* obtained with this manual procedure, and displays on RViz2 in red color the added obstacle, i.e., the black box on the real scene.



(**a**) Real scene



(**b**) Updated PlanningScene

Figure 16. Example of scene reconstruction.

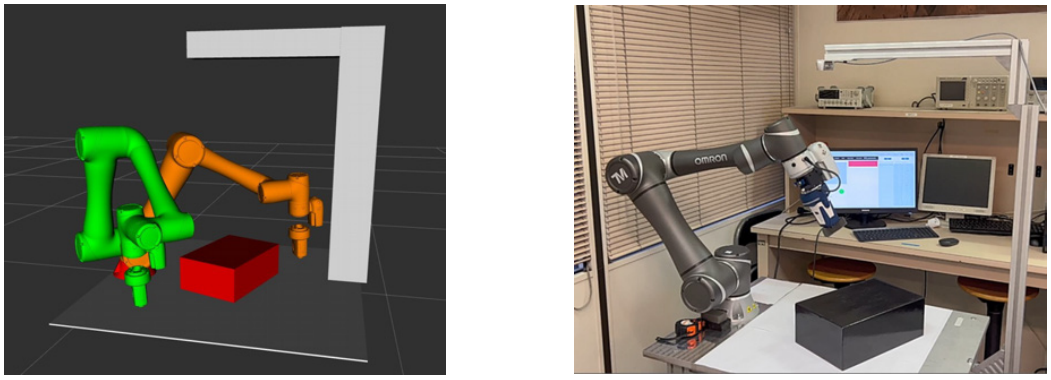
The same approach used to identify the x, y, z position of the obstacles in the cobot reference frame can be adopted also to determine the center of the object to be grasped. This will be used to plan the pick action.

5.4. Planning and Control Layer Implementation

The planning layer relies on the standard planning pipeline provided by MoveIt2 and implements a basic logic, thus not requiring its division into behavior planning, trajectory planning, and manipulation. However, this is sufficient to increase the cobot's level of autonomy, allowing it to adapt its picking and placing operations to variable but static scenarios. In particular, this flexibility concerns two aspects that are handled by this layer. The first is related to the possibility of changing the position and size of the object to be grasped without the need to manually reprogram the cobot thanks to the identification of the target pose for the picking action from the information provided by the perception layer. The second concerns the avoidance of collisions with static but unknown obstacles in the robot's workspace. This is ensured by updating the *PlanningScene* as described in the previous section and by using it for the trajectory planning of the single actions. Both aspects can be easily handled by resorting to the standard functionalities offered by MoveIt2 and described in Section 4.3. In particular, for this preliminary proof of concept, the default MoveIt2 settings were retained, which include FCL for collision detection and RRTConnect from OMPL for motion planning.

Lastly, the control layer is devoted to enabling the execution of the planned pick and place on the real robot. Conceptually, it consists of two nodes, the *Cobot Driver*, which here fulfills the task of sending commands to the cobot, and the *End Effector Driver* to control the gripper. For the simple gripper here adopted, which only provides a digital I/O interface that can be connected to the cobot's control box, the driver relies on that of the robot, which, among other things, provides read-and-write services to the robot's digital channels. The *End Effector Driver* simply requests these services to send commands for opening and closing the gripper fingers in order to grasp and release the object during the pick-and-place actions. Concerning the *Cobot Driver*, as already anticipated, there is an off-the-shelf driver provided by Techman that implements the *TMSCT* protocol to control the cobot from the outside using the *External Script* mode. This driver not only offers other dedicated services to be requested in order to send generic commands to the robot (including a *QueuTag* command to monitor the execution of the other commands) but also directly implements the *FollowJointTrajectory* action supported by MoveIt. Specifically, this action server receives in the action goal the trajectory to be executed and translates the sequence of planned points into a sequence of commands of type *PVTPoint*, each of them specifying the target position, velocity, and duration of the movement that the cobot will execute, and sends them out.

By combining the above-described perception, planning, and control layers, an ROS2 application is developed. It enables the cobot to perform a pick-and-place operation with a higher level of flexibility and autonomy. An example of the achieved results is shown in Figure 17. The two subfigures show how the cobot is able to grasp the detected object (i.e., a small black box in the gripper of the right-hand figure) by reaching the final pose (the green-colored configuration of the left-hand figure). The planning takes into account the pose and size of the object and places it in the desired pose (see the orange-colored configuration). This goal pose is reached while avoiding collisions with external objects in the environment consisting of both the initial scene (i.e., the tabletop and the camera support in light gray) and the newly detected obstacle (the large black box in the figure on the right, corresponding to the red box in the figure on the left).



(**a**) Planning Scene on RViz2 (**b**) Place execution

Figure 17. Example of pick-and-place execution.

6. Conclusions and Future Works

This work analyzes the use of ROS2 as middleware to implement modular and customizable architectures aimed at increasing the autonomy and flexibility of commercially available fixed-base robots, with a particular focus on industrial cobots. In addition to providing an extensive review of the literature regarding ROS2 and its applications, the paper surveys what is currently implemented and available in ROS2 for the development of each module and layer of the proposed high-level architecture, describing their potential and limitations, with particular emphasis on similarities and differences from ROS1 and highlighting both ready-to-use components and those still in development and/or requiring custom implementation. A proof of concept is provided using the ROS2-based framework in a specific case study involving an OMRON TM5-900 cobot equipped with an external depth camera, in order to show how to exploit the analyzed ROS2 features to enable the cobot to perform a pick-and-place task with increased autonomy and flexibility. In the provided case study, the increased flexibility concerns two aspects, i.e., the possibility to modify the position and size of the object to be grasped without resorting to manual reprogramming, and the avoidance of collisions with static unknown obstacles in the robot's workspace. The aim of this experiment is to demonstrate how even a company can provide its industrial cobots with greater flexibility, albeit within certain application limits, only by appropriately combining existing ROS2 features and tools, and how easy it is to add further customized modules to the basic framework to enable a higher level of autonomy.

In conclusion, our analysis confirms that ROS2 is a powerful middleware that can provide companies with all the tools and features needed to implement the architecture described in Section 3 for enhancing the autonomy and flexibility of cobots on the market (Section 4) to make them perform non-standard tasks not supported by their manufacturers without having to start from scratch. Moreover, unlike ROS1, ROS2 was built to be an industrial framework for use in the production environment (Section 2.2). Despite this, as a matter of fact, there are still many remaining challenges for its commercial use. The first is linked to it being open source and constantly evolving, therefore inherently lacking the reliability of commercial software developers, not supported by customer service, not user friendly, and consequently not very appealing to companies. A possible solution to this challenge is represented by ROS-Industrial, which aims to address these issues and dispel these myths. Another challenge is ensuring real-time performance because, as discussed in Section 2.2, DDS guarantees them at the communication level, but they must be properly managed by the developer at system level. Finally, being relatively new, ROS2 is still under development both in terms of porting of the functionalities previously developed for ROS1 and in the development and/or analysis of new features.

Regarding future works, they will further explore the potential of the proposed ROS2-based framework in order to show how to unlock a higher level of flexibility and autonomy by incorporating less standard and more custom components. Specifically, we plan to add a re-planning module, to enable the cobot to also handle dynamic obstacles with the implementation of a proper hybrid planner, and to upgrade the recognition module by also including object classification capabilities required for the recognition of objects to be grasped. Moreover, in order to avoid the inevitable blind spots of a single camera and to achieve complete awareness of the whole robot workspace, the experimental setup could be enhanced by adding two additional depth cameras, thus resorting to a multi-sensor architecture that can be managed by the perception layer. Finally, we also plan to analyze and discuss how to exploit and evaluate ROS2 real-time capabilities within our proposed framework as they become essential for handling dynamic obstacles.

Author Contributions: Conceptualization, M.C.G. and A.B.; methodology, M.C.G. and A.B.; software, F.G.; validation, F.G., M.C.G., and A.B.; formal analysis, F.G., M.C.G., and A.B.; investigation, F.G. and M.C.G.; resources, F.G., M.C.G., and A.B.; data curation, F.G.; writing—original draft preparation, M.C.G. and A.B.; writing—review and editing, A.B., S.L., and M.C.G.; visualization, M.C.G. and A.B.; supervision, A.B. and S.L.; project administration, A.B.; funding acquisition, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by European Project Digital Europe "EDIH4Marche" (European Digital Innovation Hub for Marche), Call DIGITAL-2021-EDIH-01, Grant Agreement No. 101084027 on topic DIGITAL-2021-EDIH-INITIAL-01 (DIGITAL Simple Grants Acton).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available because of the funded project still in progress.

Conflicts of Interest: The authors declare that this study received funding from "European Project EDIH4Marche". The funder had the following involvement with the study: it was not involved in the study design, collection, analysis, interpretation of data, the writing of this article, or the decision to submit it for publication.

References

1. Nikolakis, N.; Maratos, V.; Makris, S. A cyber physical system (CPS) approach for safe human-robot collaboration in a shared workplace. *Robot. Comput.-Integr. Manuf.* **2019**, *56*, 233–243. [CrossRef]
2. Bonci, A.; Pirani, M.; Longhi, S. Robotics 4.0: Performance improvement made easy. In Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 12–15 September 2017; pp. 1–8.
3. Indri, M.; Trapani, S.; Bonci, A.; Pirani, M. Integration of a Production Efficiency Tool with a General Robot Task Modeling Approach. In Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 4–7 September 2018; Volume 1, pp. 1273–1280.
4. Egyed, A.; Grünbacher, P.; Linsbauer, L.; Prähofer, H.; Schaefer, I., Variability in Products and Production. In *Digital Transformation: Core Technologies and Emerging Topics from a Computer Science Perspective*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 65–91.
5. Stadnicka, D.; Bonci, A.; Pirani, M.; Longhi, S. Information Management and Decision Making Supported by an Intelligence System in Kitchen Fronts Control Process. In Proceedings of the Intelligent Systems in Production Engineering and Maintenance—ISPEM 2017, Wrocław, Poland, 28–29 September 2017; Burduk, A., Mazurkiewicz, D., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 249–259.
6. Iosup, A.; Yigitbasi, N.; Epema, D. On the Performance Variability of Production Cloud Services. In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, CA, USA, 23–26 May 2011; pp. 104–113.
7. Bonci, A.; Stadnicka, D.; Longhi, S. The Overall Labour Effectiveness to Improve Competitiveness and Productivity in Human-Centered Manufacturing. In Proceedings of the International Scientific-Technical Conference MANUFACTURING, Poznan, Poland, 16–19 May 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 144–155.

8. Wang, L.; Gao, R.; Váncza, J.; Krüger, J.; Wang, X.; Makris, S.; Chryssolouris, G. Symbiotic human-robot collaborative assembly. *CIRP Ann.* **2019**, *68*, 701–726. [CrossRef]
9. Javaid, M.; Haleem, A.; Singh, R.P.; Rab, S.; Suman, R. Significant applications of Cobots in the field of manufacturing. *Cogn. Robot.* **2022**, *2*, 222–233. [CrossRef]
10. Open Source Robotics Foundation (OSRF). ROS. Available online: https://www.openrobotics.org/ (accessed on 11 August 2023).
11. ROS.org—Open Source Robotics Foundation (OSRF). ROS2. Available online: https://index.ros.org/ (accessed on 11 August 2023).
12. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. ROS: An open-source Robot Operating System. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)—Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3.
13. Willow Garage. Available online: http://www.willowgarage.com/ (accessed on 11 August 2023).
14. Urmson, C.; Anhalt, J.; Bagnell, D.; Baker, C.; Bittner, R.; Clark, M.N.; Dolan, J.; Duggins, D.; Galatali, T.; Geyer, C.; et al. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. In *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–59.
15. Boren, J.; Cousins, S. Exponential Growth of ROS—ROS Topics. *IEEE Robot. Autom. Mag.* **2011**, *18*, 19–20. [CrossRef]
16. Open Computer Vision (OpenCV). Available online: https://opencv.org/ (accessed on 11 August 2023).
17. Bradski, G. The OpenCV Library. *Dr. Dobb'S J. Softw. Tools* **2000**, *120*, 122–125.
18. Point Cloud Library (PCL). Available online: https://pointclouds.org/ (accessed on 11 August 2023).
19. Cousins, S.; Gerkey, B.; Conley, K.; Garage, W. Sharing Software with ROS [ROS Topics]. *IEEE Robot. Autom. Mag.* **2010**, *17*, 12–14. [CrossRef]
20. Korayem, M.; Nekoo, S. The SDRE control of mobile base cooperative manipulators: Collision free path planning and moving obstacle avoidance. *Robot. Auton. Syst.* **2016**, *86*, 86–105. [CrossRef]
21. Bonci, A.; Longhi, S.; Nabissi, G.; Scala, G.A. Execution Time of Optimal Controls in Hard Real Time, a Minimal Execution Time Solution for Nonlinear SDRE. *IEEE Access* **2020**, *8*, 158008–158025. [CrossRef]
22. Open Source Robotics Foundation (OSRF). ROS2 Github Repository. Available online: https://github.com/ros2 (accessed on 11 August 2023).
23. Object Management Group (OMG). Available online: https://www.omg.org/ (accessed on 11 August 2023).
24. eProsima FastRTPS. Available online: https://www.eprosima.com/index.php/products-all/eprosima-fast-rtps (accessed on 11 August 2023).
25. Real-Time Innovations. RTI Connext DDS Professional. Available online: https://www.rti.com/products/connext-dds-professional (accessed on 11 August 2023).
26. OpenDDS Foundation. Available online: https://opendds.org/ (accessed on 11 August 2023).
27. Adlink Vortex OpenSplice. Available online: https://www.adlinktech.com/en/vortex-opensplice-data-distribution-service (accessed on 11 August 2023).
28. Maruyama, Y.; Kato, S.; Azumi, T. Exploring the performance of ROS2. In Proceedings of the 2016 International Conference on Embedded Software (EMSOFT), Pittsburgh, PA, USA, 2–7 October 2016; pp. 1–10.
29. Pardo-Castellote, G. OMG Data-Distribution Service: Architectural overview. In Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, Providence, RI, USA , 19–22 May 2003; pp. 200–206.
30. Schlesselman, J.; Pardo-Castellote, G.; Farabaugh, B. OMG data-distribution service (DDS): Architectural update. In Proceedings of the IEEE MILCOM 2004. Military Communications Conference, Monterey, CA, USA , 31 October–3 November 2004; Volume 2, pp. 961–967.
31. Yang, J.; Sandström, K.; Nolte, T.; Behnam, M. Data Distribution Service for industrial automation. In Proceedings of the 2012 IEEE 17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012), Krakow, Poland, 17–21 September 2012; pp. 1–8.
32. Albonico, M.; Đorđević, M.; Hamer, E.; Malavolta, I. Software engineering research on the Robot Operating System: A systematic mapping study. *J. Syst. Softw.* **2023**, *197*, 1–28. [CrossRef]
33. Gutiérrez, C.S.V.; Juan, L.U.S.; Ugarte, I.Z.; Vilches, V.M. Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications. *arXiv* **2018**, arXiv:1809.02595.
34. Casini, D.; Blass, T.; Lütkebohle, I.; Brandenburg, B.B. Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), Stuttgart, Germany, 9–12 July 2019.
35. Kronauer, T.; Pohlmann, J.; Matthé, M.; Smejkal, T.; Fettweis, G. Latency Analysis of ROS2 Multi-Node Systems. In Proceedings of the 2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Karlsruhe, Germany, 23–25 September 2021; pp. 1–7.
36. Dust, L.J.; Persson, E.; Ekstrom, M.; Mubeen, S.; Seceleanu, C.; Gu, R. Experimental Evaluation of Callback Behaviour in ROS2 Executors. In Proceedings of the 2023 28th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; Volume 1, pp. 1–8.
37. Park, J.; Delgado, R.; Choi, B.W. Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems: An Empirical Study. *IEEE Access* **2020**, *8*, 154637–154651. [CrossRef]

38. Puck, L.; Keller, P.; Schnell, T.; Plasberg, C.; Tanev, A.; Heppner, G.; Roennau, A.; Dillmann, R. Performance Evaluation of Real-Time ROS2 Robotic Control in a Time-Synchronized Distributed Network. In Proceedings of the 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), Lyon, France, 23–27 August 2021; pp. 1670–1676.
39. Thulasiraman, P.; Chen, Z.; Allen, B.; Bingham, B. Evaluation of the Robot Operating System 2 in Lossy Unmanned Networks. In Proceedings of the 2020 IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 24–27 August 2020; pp. 1–8.
40. DiLuoffo, V.; Michalson, W.R.; Sunar, B. Robot Operating System 2: The need for a holistic security approach to robotic architectures. *Int. J. Adv. Robot. Syst.* **2018**, *15*, 1–15. [CrossRef]
41. Aartsen, M.; Banga, K.; Talko, K.; Touw, D.; Wisman, B.; Meïnsma, D.; Björkqvist, M. Analyzing Interoperability and Security Overhead of ROS2 DDS Middleware. In Proceedings of the 2022 30th Mediterranean Conference on Control and Automation (MED), Vouliagmeni, Greece, 28 June–1 July 2022; pp. 976–981.
42. Mayoral-Vilches, V.; White, R.; Caiazza, G.; Arguedas, M. SROS2: Usable Cyber Security Tools for ROS 2. In Proceedings of the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 23–27 October 2022; pp. 11253–11259.
43. Kim, J.; Smereka, J.M.; Cheung, C.; Nepal, S.; Grobler, M. Security and Performance Considerations in ROS 2: A Balancing Act. *arXiv* **2018**, arXiv:1809.09566.
44. Fernandez, J.; Allen, B.; Thulasiraman, P.; Bingham, B. Performance Study of the Robot Operating System 2 with QoS and Cyber Security Settings. In Proceedings of the 2020 IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 24–27 August 2020; pp. 1–6.
45. Erős, E.; Dahl, M.; Bengtsson, K.; Hanna, A.; Falkman, P. A ROS2 based communication architecture for control in collaborative and intelligent automation systems. *Procedia Manuf.* **2019**, *38*, 349–357. [CrossRef]
46. Erős, E.; Dahl, M.; Hanna, A.; Albo, A.; Falkman, P.; Bengtsson, K. Integrated virtual commissioning of a ROS2-based collaborative and intelligent automation system. In Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 10–13 September 2019; pp. 407–413.
47. He, J.; Zhang, J.; Liu, J.; Fu, X. A ROS2-Based Framework for Industrial Automation Systems. In Proceedings of the 2022 2nd International Conference on Computer, Control and Robotics (ICCCR), Shanghai, China, 18–20 March 2022; pp. 98–102.
48. Audonnet, F.P.; Hamilton, A.; Aragon-Camarasa, G. A Systematic Comparison of Simulation Software for Robotic Arm Manipulation using ROS2. In Proceedings of the 2022 22nd International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 27–30 November 2022; pp. 755–762.
49. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. *Sci. Robot.* **2022**, *7*, eabm6074. [CrossRef] [PubMed]
50. Tonola, C.; Beschi, M.; Faroni, M.; Pedrocchi, N. OpenMORE: An open-source tool for sampling-based path replanning in ROS. In Proceedings of the 2023 28th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; Volume 1, pp. 1–8.
51. Tonola, C.; Faroni, M.; Beschi, M.; Pedrocchi, N. Anytime Informed Multi-Path Replanning Strategy for Complex Environments. *IEEE Access* **2023**, *11*, 4105–4116. [CrossRef]
52. Wong, C.C.; Chen, C.J.; Wong, K.Y.; Feng, H.M. Implementation of a Real-Time Object Pick-and-Place System Based on a Changing Strategy for Rapidly-Exploring Random Tree. *Sensors* **2023**, *23*, 4814. [CrossRef]
53. Kang, T.; Yi, J.B.; Song, D.; Yi, S.J. High-Speed Autonomous Robotic Assembly Using In-Hand Manipulation and Re-Grasping. *Appl. Sci.* **2021**, *11*, 37. [CrossRef]
54. Zhou, G.; Luo, J.; Xu, S.; Zhang, S. A Cooperative Shared Control Scheme Based on Intention Recognition for Flexible Assembly Manufacturing. *Front. Neurorobotics* **2022**, *16*, 850211. [CrossRef]
55. Chitta, S.; Jones, E.G.; Ciocarlie, M.; Hsiao, K. Mobile Manipulation in Unstructured Environments: Perception, Planning, and Execution. *IEEE Robot. Autom. Mag.* **2012**, *19*, 58–71. [CrossRef]
56. Bagnell, J.A.; Cavalcanti, F.; Cui, L.; Galluzzo, T.; Hebert, M.; Kazemi, M.; Klingensmith, M.; Libby, J.; Liu, T.Y.; Pollard, N.; et al. An integrated system for autonomous robotics manipulation. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 2955–2962.
57. Diab, M.; Pomarlan, M.; Beßler, D.; Akbari, A.; Rosell, J.; Bateman, J.; Beetz, M. SkillMaN—A skill-based robotic manipulation framework based on perception and reasoning. *Robot. Auton. Syst.* **2020**, *134*, 103653. [CrossRef]
58. Hellmund, A.M.; Wirges, S.; Tas, O.S.; Bandera, C.; Salscheider, N.O. Robot operating system: A modular software framework for automated driving. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 1–4 November 2016; pp. 1564–1570.
59. Alderisi, G.; Iannizzotto, G.; Bello, L.L. Towards IEEE 802.1 Ethernet AVB for Advanced Driver Assistance Systems: A preliminary assessment. In Proceedings of the 2012 IEEE 17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012), Krakow, Poland, 17–21 September 2012; pp. 1–4.
60. Patti, G.; Bello, L.L. Performance Assessment of the IEEE 802.1Q in Automotive Applications. In Proceedings of the 2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE), Turin, Italy, 2–4 July 2019; pp. 1–6.

61. Bonci, A.; De Amicis, R.; Longhi, S.; Lorenzoni, E.; Scala, G.A. A motorcycle enhanced model for active safety devices in intelligent transport systems. In Proceedings of the 2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Auckland, New Zealand, 29–31 August 2016; pp. 1–6.
62. Corno, M.; Panzani, G., Traction Control Systems Design: A Systematic Approach. In *Modelling, Simulation and Control of Two-Wheeled Vehicles*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2014; Chapter 8, pp. 198–220.
63. Bonci, A.; De Amicis, R.; Longhi, S.; Lorenzoni, E.; Scala, G.A. Motorcycle's lateral stability issues: Comparison of methods for dynamic modelling of roll angle. In Proceedings of the 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 13–15 October 2016; pp. 607–612.
64. Dandiwala, A.; Chakraborty, B.; Chakravarty, D.; Sindha, J. Vehicle dynamics and active rollover stability control of an electric narrow three-wheeled vehicle: A review and concern towards improvement. *Veh. Syst. Dyn.* **2023**, *61*, 399–422. [CrossRef]
65. Bonci, A.; Longhi, S.; Scala, G.A. Towards an All-Wheel Drive Motorcycle: Dynamic Modeling and Simulation. *IEEE Access* **2020**, *8*, 112867–112882. [CrossRef]
66. AUTomotive Open System ARchitecture—AUTOSAR. Available online: https://www.autosar.org/ (accessed on 11 August 2023).
67. Henle, J.; Stoffel, M.; Schindewolf, M.; Nägele, A.T.; Sax, E. Architecture platforms for future vehicles: A comparison of ROS2 and Adaptive AUTOSAR. In Proceedings of the 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC), Macau, China, 8–12 October 2022; pp. 3095–3102.
68. Zhang, J.; Keramat, F.; Yu, X.; Hernández, D.M.; Queralta, J.P.; Westerlund, T. Distributed Robotic Systems in the Edge-Cloud Continuum with ROS 2: A Review on Novel Architectures and Technology Readiness. In Proceedings of the 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), Paris, France, 12–15 December 2022; pp. 1–8.
69. Bianchi, L.; Carnevale, D.; Del Frate, F.; Masocco, R.; Mattogno, S.; Romanelli, F.; Tenaglia, A. A novel distributed architecture for unmanned aircraft systems based on Robot Operating System 2. *IET Cyber-Syst. Robot.* **2023**, *5*, e12083. [CrossRef]
70. Testa, A.; Camisa, A.; Notarstefano, G. ChoiRbot: A ROS 2 Toolbox for Cooperative Robotics. *IEEE Robot. Autom. Lett.* **2021**, *6*, 2714–2720. [CrossRef]
71. Brock, O.; Kuffner, J.; Xiao, J., Motion for Manipulation Tasks. In *Springer Handbook of Robotics*; Siciliano, B., Khatib, O., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 615–645.
72. Suomalainen, M.; Karayiannidis, Y.; Kyrki, V. A survey of robot manipulation in contact. *Robot. Auton. Syst.* **2022**, *156*, 104224. [CrossRef]
73. Villani, V.; Pini, F.; Leali, F.; Secchi, C. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics* **2018**, *55*, 248–266. [CrossRef]
74. Taesi, C.; Aggogeri, F.; Pellegrini, N. COBOT Applications—Recent Advances and Challenges. *Robotics* **2023**, *12*, 79. [CrossRef]
75. Liu, H.; Wang, L. Collision-free human-robot collaboration based on context awareness. *Robot. Comput.-Integr. Manuf.* **2021**, *67*, 101997. [CrossRef]
76. Tavares, P.; Sousa, A. Flexible pick and place architecture using ROS framework. In Proceedings of the 2015 10th Iberian Conference on Information Systems and Technologies (CISTI), Aveiro, Portugal, 17–20 June 2015.
77. Song, K.T.; Chang, Y.H.; Chen, J.H. 3D Vision for Object Grasp and Obstacle Avoidance of a Collaborative Robot. In Proceedings of the 2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Hong Kong, China, 8–12 July 2019; pp. 254–258.
78. Megalingam, R.K.; Rohith Raj, R.V.; Akhil, T.; Masetti, A.; Chowdary, G.N.; Naick, V.S. Integration of Vision based Robot Manipulation using ROS for Assistive Applications. In Proceedings of the 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 15–17 July 2020; pp. 163–169.
79. Chiaravalli, D.; Palli, G.; Monica, R.; Aleotti, J.; Rizzini, D.L. Integration of a Multi-Camera Vision System and Admittance Control for Robotic Industrial Depalletizing. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 667–674.
80. Lee, C.C.; Song, K.T. Path Re-Planning Design of a Cobot in a Dynamic Environment Based on Current Obstacle Configuration. *IEEE Robot. Autom. Lett.* **2023**, *8*, 1183–1190. [CrossRef]
81. Ende, T.; Haddadin, S.; Parusel, S.; Wüsthoff, T.; Hassenzahl, M.; Albu-Schäffer, A. A human-centered approach to robot gesture based communication within collaborative working processes. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 3367–3374.
82. Hollmann, R.; Rost, A.; Hägele, M.; Verl, A. A HMM-based approach to learning probability models of programming strategies for industrial robots. In Proceedings of the 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 3–7 May 2010; pp. 2965–2970.
83. Krüger, J.; Lien, T.; Verl, A. Cooperation of human and machines in assembly lines. *CIRP Ann.* **2009**, *58*, 628–646. [CrossRef]
84. Hjorth, S.; Chrysostomou, D. Human–robot collaboration in industrial environments: A literature review on non-destructive disassembly. *Robot. Comput.-Integr. Manuf.* **2022**, *73*, 102208. [CrossRef]
85. Çoban, M.; Gelen, G. Realization of human-robot collaboration in hybrid assembly systems by using wearable technology. In Proceedings of the 2018 6th International Conference on Control Engineering & Information Technology (CEIT), Istanbul, Turkey, 25–27 October 2018; pp. 1–6.
86. Hmedan, B.; Kilgus, D.; Fiorino, H.; Landry, A.; Pellier, D. Adapting Cobot Behavior to Human Task Ordering Variability for Assembly Tasks. *Int. FLAIRS Conf. Proc.* **2022**, *35*, 1–6. [CrossRef]

87. YARP—Yet Another Robot Platform. Software for Humanoid Robots: The YARP. 2023. Available online: https://yarp.it/latest/ (accessed on 11 August 2023).
88. Orocos—Open Robot Control Software. The Orocos Project. 2023. Available online: https://orocos.org/ (accessed on 11 August 2023).
89. Longhi, M.; Taylor, Z.; Popović, M.; Nieto, J.; Marrocco, G.; Siegwart, R. RFID-Based Localization for Greenhouses Monitoring Using MAVs. In Proceedings of the 2018 8th IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (APWC), Cartagena, Colombia, 10–14 September 2018; pp. 905–908.
90. Longhi, M.; Marrocco, G. Ubiquitous Flying Sensor Antennas: Radiofrequency Identification Meets Micro Drones. *IEEE J Radio Freq. Identif.* **2017**, *1*, 4, 291–299. [CrossRef]
91. MOOS—Mission Oriented Operating Suite. mit.edu. 2023. Available online: https://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php (accessed on 11 August 2023).
92. Serrano, D. Middleware and Software Frameworks in Robotics—Applicability to Small Unmanned Vehicles. In Proceedings of the NATO-OTAN ST Organization Cerdanyola del Vallès, Spain, 4–5 January 2015; pp. 1–8.
93. Karpas, E.; Magazzeni, D. Automated Planning for Robotics. *Annu. Rev. Control. Robot. Auton. Syst.* **2020**, *3*, 417–439. [CrossRef]
94. Pereira, J.L.; Queirós, M.; C. da Costa, N.M.; Marcelino, S.; Meireles, J.; Fonseca, J.C.; Moreira, A.H.J.; Borges, J.L. TMRobot Series Toolbox: Interfacing Collaborative Robots with MATLAB. In Proceedings of the 3rd International Conference on Innovative Intelligent Industrial Production and Logistics—IN4PL. INSTICC, SciTePress, Valletta, Malta, 24–26 October 2022; pp. 46–55.
95. Nabissi, G.; Longhi, S.; Bonci, A. ROS-Based Condition Monitoring Architecture Enabling Automatic Faults Detection in Industrial Collaborative Robots. *Appl. Sci.* **2023**, *13*, 143. [CrossRef]
96. Bonci, A.; Longhi, S.; Nabissi, G. Fault Diagnosis in a belt-drive system under non-stationary conditions. An industrial case study. In Proceedings of the 2021 IEEE Workshop on Electrical Machines Design, Control and Diagnosis (WEMDCD), Modena, Italy, 8–9 April 2021; pp. 260–265.
97. Kermenov, R.; Nabissi, G.; Longhi, S.; Bonci, A. Anomaly Detection and Concept Drift Adaptation for Dynamic Systems: A General Method with Practical Implementation Using an Industrial Collaborative Robot. *Sensors* **2023**, *23*, 3260. [CrossRef] [PubMed]
98. Elfes, A.; Steindl, R.; Talbot, F.; Kendoul, F.; Sikka, P.; Lowe, T.; Kottege, N.; Bjelonic, M.; Dungavell, R.; Bandyopadhyay, T.; et al. The Multilegged Autonomous eXplorer (MAX). In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 1050–1057.
99. Hernandez-Mendez, S.; Maldonado-Mendez, C.; Marin-Hernandez, A.; Rios-Figueroa, H.V.; Vazquez-Leal, H.; Palacios-Hernandez, E.R. Design and implementation of a robotic arm using ROS and MoveIt! In Proceedings of the 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), Ixtapa, Mexico, 9–11 November 2017; pp. 1–6.
100. Gazebo Simulator. Available online: https://classic.gazebosim.org/tutorials?tut=ros2_overview&cat=connect_ros (accessed on 8 August 2023).
101. Wang, Y.; Liu, L.; Zhang, X.; Shi, W. HydraOne: An Indoor Experimental Research and Education Platform for CAVs. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), Renton, WA, USA, 9 July 2019.
102. Koubaa, A. (Ed.). *Robot Operating System (ROS): The Complete Reference (Volume 1)*; Studies in Computational Intelligence; Springer: Berlin/Heidelberg, Germany, 2016; Volume 625.
103. Wang, X.; Yang, C.; Ju, Z.; Ma, H.; Fu, M. Robot manipulator self-identification for surrounding obstacle detection. *Multimed. Tools Appl.* **2017**, *76*, 6495–6520. [CrossRef]
104. Hornung, A.; Wurm, K.; Bennewitz, M.; Stachniss, C.; Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Auton. Robot.* **2013**, *34*, 189–206. [CrossRef]
105. Robotic, P. MoveIt 2 Documentation—Planning Scene Monitor. 2023. Available online: https://moveit.picknik.ai/humble/doc/concepts/planning_scene_monitor.html (accessed on 11 August 2023).
106. Available online: https://github.com/leggedrobotics/darknet_ros.git (accessed on 11 August 2023).
107. Canny, J. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *PAMI-8*, 679–698. [CrossRef]
108. Şucan, I.A.; Moll, M.; Kavraki, L.E. The Open Motion Planning Library. *IEEE Robot. Autom. Mag.* **2012**, *19*, 72–82. [CrossRef]
109. Meijer, J.; Lei, Q.; Wisse, M. Performance study of single-query motion planning for grasp execution using various manipulators. In Proceedings of the 2017 18th International Conference on Advanced Robotics (ICAR), Hong Kong, China, 10–12 July 2017; pp. 450–457.
110. Zucker, M.; Ratliff, N.; Dragan, A.D.; Pivtoraiko, M.; Klingensmith, M.; Dellin, C.M.; Bagnell, J.A.; Srinivasa, S.S. CHOMP: Covariant Hamiltonian optimization for motion planning. *Int. J. Robot. Res.* **2013**, *32*, 1164–1193. [CrossRef]
111. Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; Schaal, S. STOMP: Stochastic trajectory optimization for motion planning. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 4569–4574.
112. Pan, J.; Chitta, S.; Manocha, D. FCL: A general purpose library for collision and proximity queries. In Proceedings of the 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 14–18 May 2012; pp. 3859–3866.
113. SDK, B.P. bullet3 . 2023. Available online: https://github.com/bulletphysics/bullet3.git (accessed on 11 August 2023).
114. Völz, A.; Graichen, K. A Predictive Path-Following Controller for Continuous Replanning With Dynamic Roadmaps. *IEEE Robot. Autom. Lett.* **2019**, *4*, 3963–3970. [CrossRef]

115. Robotics, P. MoveIt2-Hybrid Planning. Available online: https://moveit.picknik.ai/main/doc/concepts/hybrid_planning/hybrid_planning.html (accessed on 11 August 2023).
116. IntelRealSense. Realsense-Ros. 2023. Available online: https://github.com/IntelRealSense/realsense-ros.git (accessed on 11 August 2023).
117. TechmanRobotInc. tmr_ros2. 2023. Available online: https://github.com/TechmanRobotInc/tmr_ros2.git (accessed on 11 August 2023).