Introduction to the Robot Operating System (ROS): Main Concepts and Application Scenarios

Prakhar Rathi[0000-0002-5196-3953]

[1] Universität Paderborn, Paderborn NRW 33098, Germany
[2] Software Innovation Campus Paderborn - SICP, Paderborn NRW 33102, Germany
prakhrathi@gmail.com

Abstract. This paper delves into the scope of ROS, a meta-operating open-source robot operating system. ROS provides an integrated development environment to program a robot equipped with features to perform highly complex and technically advanced tasks. These tasks are executed with little effort with the assistance of the underlying traditional operating system for memory management and process scheduling. ROS is increasingly becoming a vital factor in the development of large-scale robot software. The aim of this study is to further current knowledge of fundamentals of the ROS and discuss various use-cases of the ROS with prime focus on applications of the robotics industry coupled with augmented reality (AR) and virtual reality (VR). Moreover, this paper briefly describes a few application areas of ROS namely: sending velocity commands based on location information, sign-following robots, and the dobot magician system.

Keywords: ROS, Robotics, Robot Operating System, Robot, Augmented Reality, Virtual Reality, AR, VR.

1 INTRODUCTION

With a rampantly increasing inclination of researchers and developers towards the field of robotics, creating a robot software that is robust and performs all the tasks seamlessly is considerably tough. Depending on the use-cases, robot software can be programmed to execute a wide multitude of tasks which include computer vision, object recognition, following a path based on GPS, movement of body-parts, and so forth. Did you know that in CES 2020, Samsung unveiled a chef bot that could fix you up a plate of salad on your command [1]?

Since a robot is fundamentally a machine, the tasks that seem minor with respect to the human perspective are carried out by the robot as a complex series of actions. These complex series of actions are executed as computer programmed lines of code. Therefore, the expanse of code to be written to program a straightforward robot can be a dismaying imagination. The key problem with this technique is that the extensive range of functionalities that can be put into action by a single robot outruns the scope of expertise of a single researcher, developer, organization, or even an institution [2].

To rectify as many such challenges as may occur while building a robot software, the necessity to decrease redundant efforts and increase the efficiency of the product has been recognized by many researchers from time to time. Several frameworks have been built to aid the development process by allowing reuse of the code for the feature implemented by these frameworks and make the code more comprehensible. Each of these frameworks administers a specific feature, thus a loose coupling between the individual frameworks cannot be gainsaid.

The overall goal of this paper is to introduce the Robot Operating System (ROS) [3] and its application areas in collaboration with VR and AR. Contrary to what the name suggests, it is not a real-time operating system rather, it is a middleware for programming large scale robots. It can be firmly believed that a common platform for integrating various features to use as the lower level of robot development and build on top of this layer would significantly improve the adeptness of the software hence produced.

This paper is structured as follows: An overview of ROS is provided in section 2 followed by the nomenclature of ROS in Section 3. Section 4 explains the philosophical design goals of ROS. Section 5 explains the commonly encountered use-cases while implementing ROS. Section 6 provides an overview of integrating AR/VR with robotics and a few application examples of ROS are discussed in section 7. Thereafter, this paper is concluded in section 8.

2 OVERVIEW OF THE ROS

A typical robot must perform a wide multitude of tasks to be established as an accomplishment (Figure 1). These tasks include complex operations such as computer vision, speech analysis, language processing, and so forth. Dealing with such intricacies remains to be beyond the bounds of a single organization or an institution. For instance, a lab A might be prodigious with GPS and another institution B would exhibit remarkable success with digital signal processing. ROS is well equipped with most of the technical stack required for developing a robot and embedding complex features in it. In addition to it, ROS provides a common platform to practitioners across the globe to use the generic code corresponding to the functionality aspired and integrate it with their own code effectuating the objective. ROS does not replace, instead it works alongside a traditional operating system. ROS is neither a programming language, nor only a library, nor an integrated development environment [5].

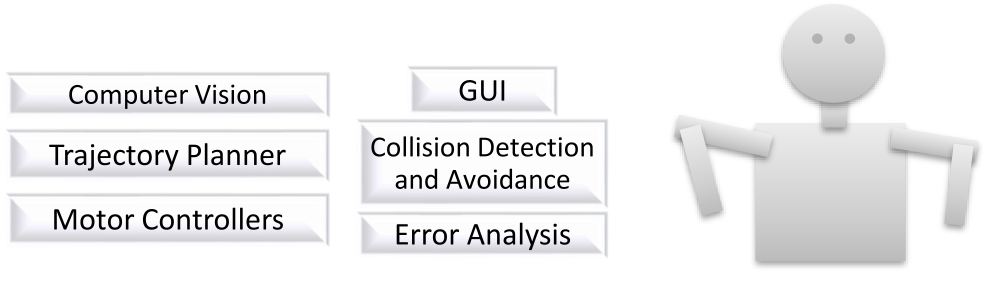


Fig. 1. Some fundamental tasks conducted by a robot [4].

What is ROS? This section explains the platform in detail. The robotics industry is making tremendous progress in every field. Initially being an area of fascination for researchers, robotics has now successfully planted its roots in the commercial sectors as well. As per the report published by the International Federation of Robotics in Feb/2020, approximately 2 million units of robots will be installed in various industries [6] such as surgical enhancements in health-care, co-operative systems in agriculture and vehicle networking, the highly skilled and unmanned armory in the military and many such vital fields of operation [7]. The rest of this section will begin with a short summary of the term robot and the ROS is explained further.

2.1 Robot

A robot is a computer programmed machine that performs various tasks with speed and precision. It can perceive the environment that is its surroundings, take decisions based on the state of the environment and is able to execute the instructions generated [8].

2.2 ROS – A Meta Operating System.

ROS is not a traditional operating system in a manner that is dependent on a kernel-based operating system to support the ROS with fundamental operating system operations such as resource memory allocation, process scheduling, user interface and security, and so forth. ROS is a collection of modules, libraries, and other features that are required for creating an integrated development environment for robot creation and assembly operations. A comprehensive approach to understanding the ROS can be given as follows:

$$\text{ROS} = \text{Plumbing} + \text{Tools} + \text{Capabilities} + \text{Ecosystem [9].} \quad (1)$$

The components comprising of the Equation (1) will be discussed in more detail in the following sub-sections.

Plumbing. ROS was devised as an open-source platform, enabling the users to select the setup of libraries and tools which interacted with the underlying core implementation of the ROS and allow them to move their software stack as per the robot's application requirements. Therefore, the implementation of a robot software is completely unattached to the core ROS except the structure enforcing communication between various individual programs. These individual programs, discussed later in the section 3, are termed as nodes. Thus, ROS is a plumbing providing a message passing interface through publish/subscribe semantics where nodes can either publish or subscribe data for transfer among distributed computing components [10].

Tools. ROS comes equipped with a powerful development toolset providing support for the introspection of the data being communicated among various nodes through the publish/subscribe message passing infrastructure, debugging of errors as they occur, specifying the configuration of the desired product, logging, testing and visualization of the development process. For example, RViz is a tool implementing 3D visualization allowing us to see the environment as perceived by the robot [10].

Capabilities. Capabilities signify that the ROS can perform a multitude of tasks by using pre-existing libraries provided with the ROS to implement most of the robot functionalities. For instance, a high-level perception interface capable of implementing virtual reality concepts [9].

Ecosystem. ROS, being an open-source platform, is studied regularly by a large community. Users across the globe implement and research the ROS and report their respective ideas to enhance the current state of the platform by integrating their improved code for an existing functionality or adding an extra library with proper documentation [3].

3 NOMENCLATURE

This section seeks to address a brief overview of the ROS graph concepts and visualize how they influence the design and implementation of ROS. The ROS infrastructure comprises of the following fundamental concepts:

3.1 Nodes

Nodes can be conceptualized as processes that implement a functionality and perform computations. The nodes are designed to be modular at a fine-grained scale. Each node performs a specific operation which is usually a segment of the algorithm. Nodes can be understood as a software module written in one of the programming languages such as C++ or Java (Figure 2). Therefore, a complete robot system comprises of many nodes interacting with each other through streaming topics, RPC services, and the Parameter Server [2]. For instance, a node captures the image of an object while another node processes image recognition to classify the object, one node will control the arm path planning, one implements the motor wheel functionality, and so forth [3].

The usage of nodes provides various advantages to the development process [2]:

1. *Error Detection and Debugging*. Errors can be detected and generalized to individual nodes depending upon the functionality encountering a crash.
2. *Code Complexity and Code Reuse*. Code complexity is reduced as compared to monolithic systems. A node is designed to execute a specific operation; thus, it can be used wherever the functionality is desired.
3. *Abstraction*. The implementation details are abstract, and the nodes communicate through a minimal API to the rest of the robot structure and other executables.

Camera
Motion Planner
Image Processing

Fig. 2. Nodes are independent components executing specific tasks. The camera node can be implemented using Python whereas, the Motion Planner and the Image Processing can be written using C++ and Java, respectively.

3.2 Publish/Subscribe Communication Infrastructure for Topics

Topics are shared communication channels enabling communication and connection between the nodes. The topics can be perceived as a stream of data with unique names associated to a publish/subscribe semantics of message transfer system, mentioned in section 2.2. The publish/subscribe infrastructure works with anonymity thus, a node is unaware of the interacting node's identity [4]. The nodes which are interested in receiving specific information subscribe to the corresponding topic whereas, the nodes generating the data send it through the publish semantic to the corresponding topic (Figure 3). Multiple publishers and subscribers related to a topic can exist.

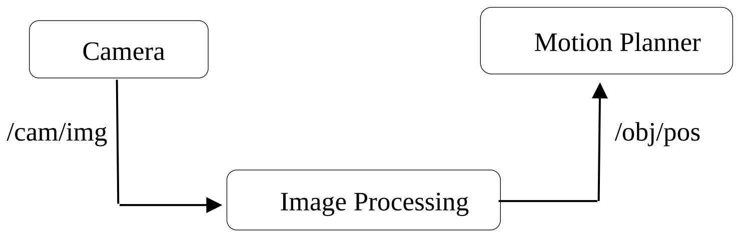


Fig. 3. Nodes interact via Topics. For instance, /cam/img is a topic which is the data from the camera.

3.3 Services

Topics are proposed for the unidirectional streaming of data. For communication between nodes that intend to perform remote procedure calls to receive a response for a request, the many-to-many relationship of one-way publish/subscribe transport is not significant. A robot system consists of many nodes communicating with each other in a distributed environment thus request/reply interactions require a two-way connection. A service is defined by a string name and comprises of a pair of messages: one for the request and one for the response [2]. A client node sends a request message to the service providing node which then replies using the response message (Figure 4). Contrary to the communication infrastructure in topics, only one node can provide a service with a specific name [10].

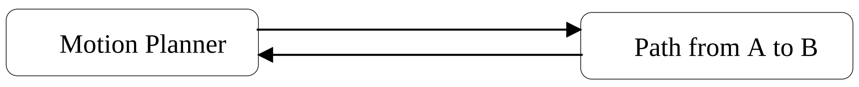
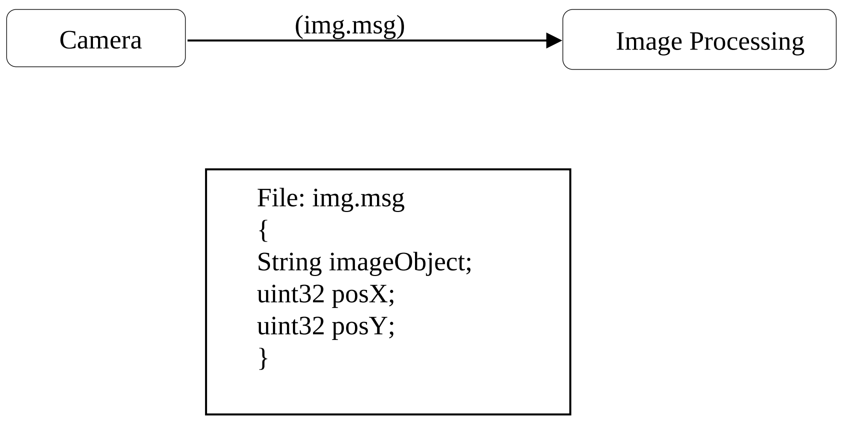


Fig. 4. Motion Planner requests a path from a service which then responds to the request.

3.4 Messages

Inter-node communication is carried out by publishing messages to the relevant topics. Messages are strictly typed, simple data structures supporting the standard primitive data types such as integer, floating-point, boolean, and so forth (Figure 5). Additionally, arrays of primitive types are supported. Messages can comprise of other messages and nested structures analogous to C structs [4].



```
File: img.msg
{
String imageObject;
uint32 posX;
uint32 posY;
}
```

Fig. 5. Messages are sent on defined topics [4].

3.5 Bags

A ROS bag is a file format created by subscribing to different ROS topics and storing the serialized ROS message data as it is received. Tools can be used to further process the data, store, analyze and visualize them [10].

3.6 Master

The ROS Master allows nodes to communicate each by providing naming and registration services thereby allowing lookup to nodes (Figure 6). Master registers the nodes to itself and actively tracks the publishers and the subscribers to relevant topics and the services. A node-to-node communication is setup after all the nodes have registered to the master [10]. However, ROS2 has discontinued the implementation of master and equips each node with the ability to discover other nodes.

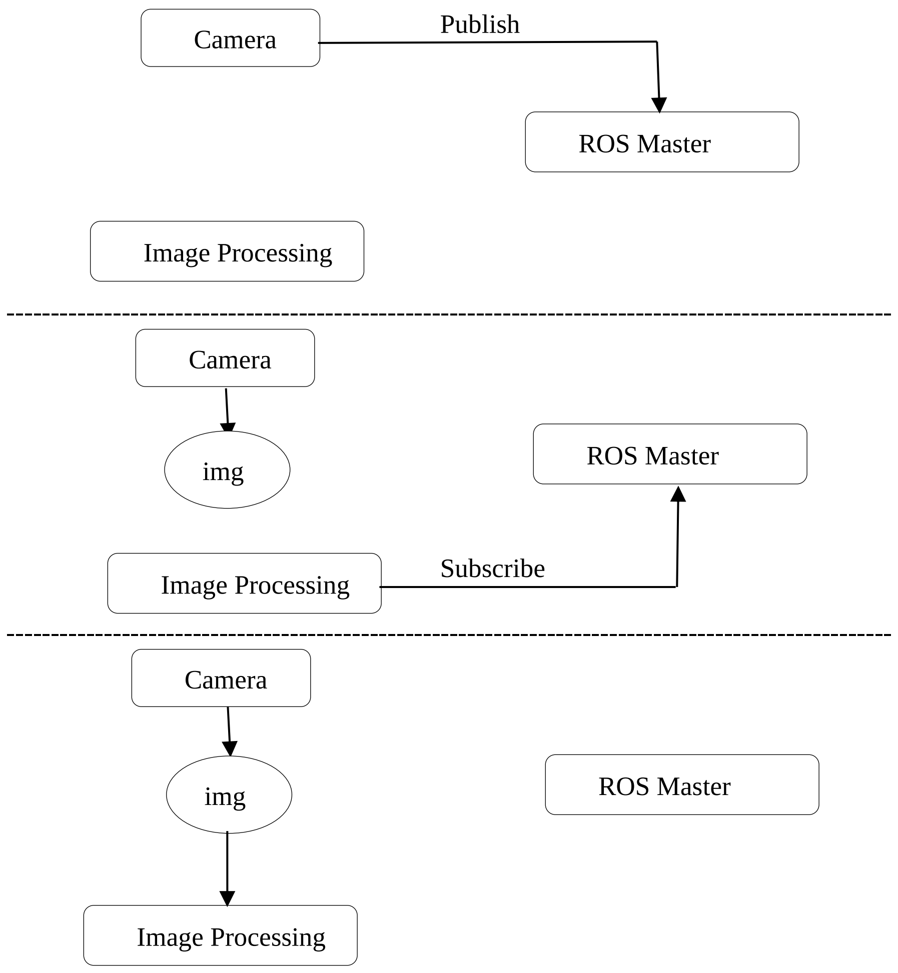


Fig. 6. Master plays an important role in inter-node communication by enabling nodes to lookup for other nodes that publish the required data [4].

3.7 Parameter Server

A parameter server is a shared, multi-variate database accessible with the network APIs [11]. Nodes use this database to access and update parameters at runtime. The design goal of a parameter server overlooks high-performance, therefore, this approach is beneficial with the non-binary data which does not undergo frequent changes such as the weight of the robot, distance between to fixed points, and so forth.

4 PHILOSOPHICAL GOALS

ROS was developed in 2007 while creating the prototypes of dynamic and flexible software libraries intended for robotics development during the STanford AI Robot (STAIR) at Stanford University[1] and the Personal Robots (PR) program at Willow Garage[2]. Further research and periodic improvements resulted in the formation of a generic architecture. There are different robot development platforms such as Google ROBEL, NVIDIA Isaac, Apollo Baidu, among others. However, the design goals of ROS provide the ROS an edge over the others. The design goals of ROS can be summarized as [2]:

- Peer-to-peer messaging.
- Tools-based software design.
- Multi-language support.
- Lightweight.
- Free and Open-Source.

This section sheds light on these philosophies and examines their influence on the design and implementation of ROS by combining the information from the literature work by Quigley, M. et al. (2009) [2], the notes by Mišeikis, J. (2018) [4], the book by O'Kane, J. (2014) [5], and Wikipedia [10].

4.1 Peer-to-Peer Messaging.

A robot system developed using ROS consists of many nodes, presumptively present on different hosts implementing the inter-process communication in a peer-to-peer network topology at runtime. Connections based on a central server are not advantageous as the computers might be connected in a heterogeneous network. Routing all messages through a central server can significantly increase the channel load in addition to posing a threat of data-loss and complete system-failure in case of fault occurrence in the central server.

For example, a typical ROS network configuration can be visualized as a connection between several onboard and offboard machines. The onboard computers are connected via ethernet and bridged with the offboard machines, performing high-computation tasks such as computer-vision or image-processing, through a wireless LAN connection (Figure 7). Implementing a central-server configuration using a computer host either among the onboard machines or the offboard machines will significantly increase the channel load on a relatively slow wireless connection.

[1] http://stair.stanford.edu

[2] http://pr.willowgarage.com

The peer-to-peer topology contradicts the notion of a single server and employs a ROS master on a machine for naming, registration, and lookup mechanism to allow nodes to identify each other and initiate communication at runtime.

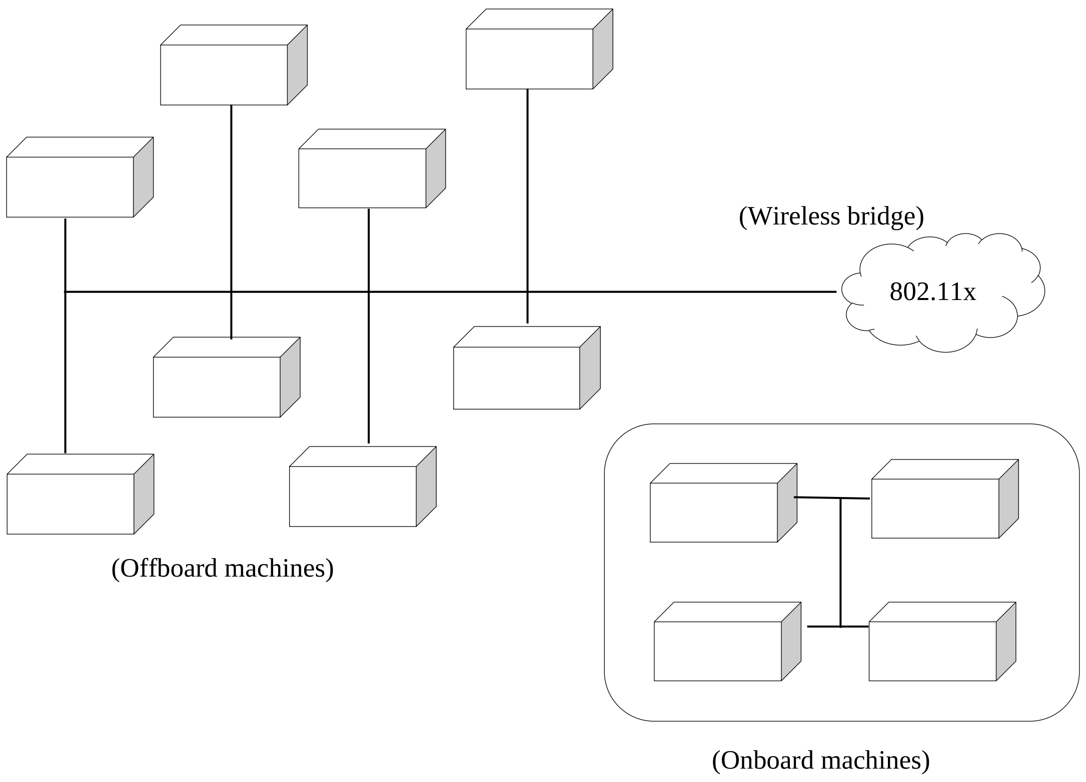


Fig. 7. A typical ROS network configuration [2].

4.2 Tools-Based Software Design

To address the problem of the complexity of robotics, ROS implements a microkernel design approach which includes umpteen finely grained modular tools to develop a robot and implement various ROS features. Instead of implementing a monolithic development environment, many cooperative processes interact with each other.

Each tool is associated with a task, for example, catkin_make builds ROS nodes, rosrun, and roslaunch support running of nodes, rqt_graph is used to view network topology, and rostopic monitors network traffic.

4.3 Multi-Language Support

When coding, different programmers are inclined towards different programming languages above others due to personal and technical tradeoffs between programming time, simplicity of debugging, understanding of the syntax and semantics, runtime efficiency, and many other such reasons. To address these reasons, ROS is designed as a language-neutral platform, thus programmable in many different languages. ROS is currently implemented in four programming languages: C++, Python, Octave, and Lisp along with experimental libraries in Java and Lua.

The ROS specification is employed at the messaging layer. Peer-to-peer connections are negotiated, and the configuration is designed in XML-RPC, which can be programmed using most major languages. Instead of providing C/C++ implementations with stub interfaces to support all languages, native implementations in each target language are preferred. However, in some cases either existing C++ libraries can be effectively re-wrapped such as the Octave client, or messages can be described in the IDL(Interface Definition Language) and message-generation can be enabled by writing classes to support the new language.

IDL is a simple and language-neutral interface facilitating cross-language communication and data-transfer. IDL defines the messages transferred between client and server modules in remote procedure calls in a language-independent format. Nodes can be independently programmed using different languages and access the interface to access each other's data (Figure 8).

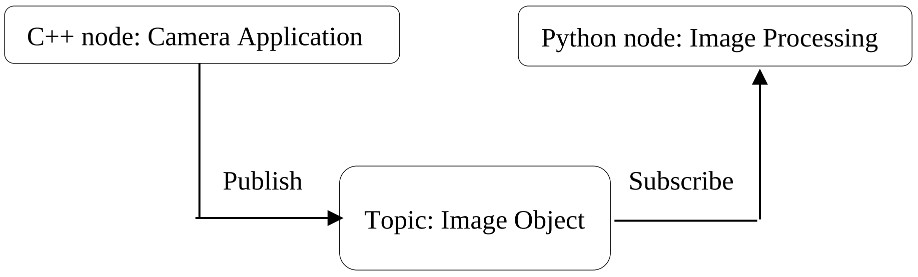


Fig. 8. Nodes interact despite being implemented using different programming languages.

4.4 Lightweight

Robotics software systems often implement drivers or algorithms that can be reused outside of the scope of the project. However, most of these algorithms are entangled with the middleware to such a high degree that extracting their functionality, and thus reusability of the code, becomes an uphill task.

To combat this limitation, developers encourage to compile all the drivers and algorithms in standalone libraries having no dependencies on ROS. The ROS build system integrates independent libraries by performing modular builds within the source code tree. The complexities associated with the functionalities are kept abstract in the libraries and respective small executables expose the nature of the feature implemented to the ROS. This approach enhances code extraction and code reusability, and makes unit testing efficient as the code is factored into standalone libraries.

ROS re-uses code from a variety of open source projects, such as the computer vision library and vision algorithms from OpenCV [12], 3D data processing from Point Cloud Library [13] and motion planning from MoveIt [14], among many others.

4.5 Free and Open-Source

ROS is released under the terms of the BSD license; the complete ROS source code is publicly available for commercial and research use.

5 USE CASES OF ROS

In this section, a short description of various common use-cases that occur while implementing robotic software frameworks, describe the relevant ROS approach and introduce some much-used tools associated with ROS [2].

5.1 ROS Debugging

During the phase of robotics research and development, a large software ecosystem should be up and running for experiments. For example, to perform an image processing experiment, drivers and libraries for the camera(s), object recognizer(s), and other required intermediate nodes must be functional.

ROS potentially minimizes the complexity of debugging in such situations by exploiting its modular structure, which enables node implementations in the development phase to execute in parallel with the pre-existing debugged nodes as the nodes in ROS connect to each other at runtime. Any changes in the infrastructure graph of employed nodes are done by observing the behavior of the process. Starting the process is completely independent thus it can be stopped, re-compiled, and restarted without causing any disturbance to the software system [2].

ROS introduces several tools for debugging a process either using the command-line or in a debugger. For instance, rostopic [15] displays debug information about ROS topics: publishers, subscribers, publishing rates, and ROS message content.

5.2 Logging Data and Playback

The efficiency of an algorithm applied can most ideally be inferred by observing the logged data. Logging and playback are the most important features in the development lifecycle of robotics software systems. ROS's ability to log the data coupled with the feature to play it back enables the engineers to record diagnostic data, aids the researchers to gather datasets, and helps the developers to test the algorithms and perform comparisons among various approaches [2]. Logging and playback can be efficiently executed through the command line on any message streams that are dumped to the disk.

ROS provides this feature with the bag files and implementation tools such as rosbag [16].

5.3 Packaged Subsystems

All the processes can be run from the command line. However, instantiating the same processes repeatedly by typing the commands could get dreary. An optimistic approach would be to compile all the required processes implementing a specific functionality in a single package and devise a single command to run all these processes at once [2]. A ROS package is a directory containing an XML file describing the configuration of the package and the existing dependencies.

The ROS tool roslaunch facilitates the user to apply this approach, roslaunch instantiates all the processes on the cluster which are mentioned in the XML file corresponding to the required functionality [2].

5.4 Integrated Development

The boundless potential of Artificial Intelligence fused with the rampantly growing scope of robotics urges the need for collaboration between researchers and organizations to build and invent large-scale systems. The philosophy of packaging subsystems benefits the idea of collaborative development. A large software system can be visualized as a tree with ROS packages at the leaf nodes.

A utility tool is provided by ROS, rospack, to perform search operations on the dependencies, inspect the code tree, execute queries, etc [2]. The rospack supports simultaneous-development across multiple ROS packages repositories.

5.5 Visualization

The design and development phase of any a robotics software needs to be closely monitored at various states of execution. Using standard output methods such as printf aid the debugging process on a monolithic development environment but are insignificant when observing large-scale software systems in a distributed-development environment [2].

ROS employs a multitude of features which communicate through the publish/subscribe infrastructure and communicate by passing messages. The dynamics of the data stream carrying these messages require a nested and deep observation phenomenon by tapping-into the data structure and channel properties. ROS is equipped with a powerful visualization program known as rviz. The rviz uses a plugin architecture, plugins can be written to display different types of data. Visualization panels are dynamically created to display the sensor data, robot joint states, maps being built, etc [4].

A native ROS port for Python is provided [2]. The rostopic, a powerful tool written in Python, filters messages and converts the message stream into a text stream. Additionally, the rxplot utility provides the functionality of plotting variables in real-time as a time series using Python.

5.6 Transformations

Robotics developers often require tracking geo-spatial relationships for better research and improvement of the software system. For instance, a mobile robot must be studied with respect to a fixed frame of reference.

ROS provides a transformation package called tf to keep track of multiple coordinate frames over time [17]. A coordinate frame can be defined as a set of unit vectors at right angles to each other. The tf constructs a transformations tree dynamically and handles the transformations between coordinate frames, i.e. the robot's movement, relative to a fixed frame of reference, in space and time.

6 AR/VR AND ROS

Robotics is gaining importance in commercial as well as non-commercial areas of expertise. Continuous enhancements in the robot software system have enabled the robots to support high-precision operations such as medical surgeries and military combats. Building a large-scale robot system can be inefficient in terms of cost and time. To achieve the best results, it is necessary to ensure the efficiency of the robot software during the development phase itself. This section introduces the two approaches of simulating a real-world environment and a collaborative approach to combine augmented reality, virtual reality, and robotics.

6.1 Virtual Reality

Virtual Reality (VR) is an efficacious human-machine interaction interface using computer technology to simulate the real-world environment. VR is created by employing the concepts of computer graphics and simulating the senses of vision, touch, hearing, and so forth. This technology places the user inside a virtual experience and allows the human to visualize, manipulate, and interact with the virtual environment [18].

Integrating the ROS systems with virtual reality systems concerns with the bridging of two very different software technologies. The absence of a standard interface to connect ROS with the standard VR paradigms, for example, Unity, deprives the application of ROS from consumer-grade hardware. To overcome this hurdle, a framework has been developed within the ROS system known as the ROS Reality [19].

6.2 Augmented Reality

Augmented Reality (AR) interactively enhances the perception of the real-world by integrating a computer-generated image sequence with virtual objects. AR can be used to bridge the gap between human and robot-systems as AR can overlay a virtual image on a real robot displaying its internal state and allowing the user to manipulate the state directly. ROS provides an AR toolkit named as artoolkit [20] for implementing AR algorithms using visualization information from ROS.

6.3 AR/VR with Robotics

Robots have been supporting various industrial tasks such as manufacturing and packaging since their beginning. With the application of robots in educational, commercial, and industrial areas, users achieve high-performance, durability, and safety when working in hazardous situations.

Modern technologies and scientific advancements such as augmented reality and virtual reality, the advent of smartphones, wireless communications, etc, have provided a fascinating environment for games and media. AR and VR have been passionately researched, improved, and utilized vastly in industrial sectors such as healthcare, education, medical, and so forth.

VR/AR can be combined with robotics to train and monitor robots thereby speeding up the learning process and improve the robot's efficiency. The application of low-latency networks enables users to utilize robots remotely with AR/VR controls.

This section discusses some recent and potential application areas of AR and VR in robotics [21, 22].

Robot Training. Virtual Reality and Augmented Reality can be deployed to train robots for various tasks. Organizations often employ a dual neural network approach to train their robots. The first neural network can be visualized as a visual network, it captures an image of the object and observes the state of the object. The second network, known as the imitation network, tries to understand a demonstration, analyze, and imitate the action being performed.

The motion tracking technology of VR controllers and the motion sensors used in AR and VR gaming can enable robots to visualize, analyze, and learn movement patterns. The concepts of Machine Learning such as reinforcement learning can be applied to empower robots to deliver high-performance in tasks demonstrated in the learning process and other unpredictable scenarios.

One of the prominent examples of this approach is to train a robot to learn programming by demonstration. The demonstration can be created by simulating a virtual environment set up in VR, or by superimposing the virtual environment over the real environment with the help of AR. The Elon Musk[3]-backed OpenAI lab is training robots using a vision network [23].

Human-Robot Interaction. Artificial Intelligence has expanded the areas of expertise related to a robot's functioning beyond the monotonicity of pre-programmed simple tasks. To keep up to the pace of robots becoming increasingly intelligent and exhibiting human-like analytical and logical abilities, it is of utter importance to devise a way of interaction between humans and robots. A group of robotic researchers at the University of Boulder, Colorado[4] used augmented reality to predict a drone's future actions based on its trajectory and response to the existing obstacles among other factors [24].

[3] https://en.wikipedia.org/wiki/Elon_Musk
[4] https://www.colorado.edu/

Object Identification. Artificial Intelligence embedded robots encounter an extensive range of objects and data. A robot employs machine learning algorithms to classify the objects based on existing data used by the robot for learning to differentiate between various objects. However, rendering an environment with a broad range of objects to train the robot can be time-consuming and significantly intricate.

A team of researchers at the University of California, Berkeley[5], has implemented a training technique using a virtually simulated environment wherein they trained the robot to identify the object objects indicated by the user and pick-up the object [25]. Implementing virtual reality curbs the over-head of collecting real-time objects by creating a 3D model of objects for training.

Healthcare. Advanced robotic arms have been used for performing surgeries as well as in the drug manufacturing processes. AR/VR interfaces can be provided for robotic systems to better understand the biological structure and analyze the human body in detail. This approach can allow students to learn and practice virtual surgeries. Furthermore, exoskeleton models based on augmented reality can assist people with disabilities to stand, move, sit, etc.

Military. Robots are making their way into the defense infrastructure by strengthening surveillance measures and security. Drones are an important part of the patrolling operations. Military robots can be equipped with virtual reality to simulate battlefield scenarios and train robots accordingly. Augmented reality assisted exoskeletons have become a crucial component of the military in the last 5 years. These exoskeletons provide manifold increase in durability, strength, stamina, load-carrying capability, and performance of the soldiers. Additionally, with VR equipped drones soldiers can inherently improve the decision-making process to tackle various battlefield situations.

Space Exploration. Implementing telebrotics based robots for remote planets can be tedious due to the large distance limiting real-time communication. Delays between such communications are potentially unavoidable, giving rise to a demand for effective alternatives to reduce delay and enhance communication.

Space researchers are immersively involved in developing technology to implement teleoperation technologies using VR to determine the maximum distance of a robot from the controller which remains unaffected by latency.

Judiciary. Jurors are assigned a critical responsibility of passing judgments based on available spoken, written, and photographic evidence. Virtual reality assisted robots can assist the jurors by enabling them to explore a crime scene in a 3D environment in 360° as many times as required. Thus, VR can help the jurors make more viable and accurate verdicts.

[5] https://www.berkeley.edu/

7 APPLICATION SCENARIOS

The fundamental concepts of ROS illustrated in section 3 and section 4 imply that ROS is an industry-standard platform. An advanced version of ROS is available known as ROS-2. The ROS-2 is being currently tested and supported on Ubuntu Xenial as well as Windows 10. On the other hand, ROS supports Linux operating systems, primarily Ubuntu. The differences between ROS-1 and ROS-2 can be understood in [34]. This section throws light on a few application areas of ROS.

7.1 Sending Velocity Commands

Simulink [26] can control a simulated robot running in a Gazebo [27] robot simulator over ROS network. The model implements a simple closed-loop proportional controller [28] that receives location information from a simulated robot and responds with velocity commands to drive the robot to the desired location (Figure 9). The topic /odom fetches the location and /mobile_base/commands/velocity sends velocity commands.

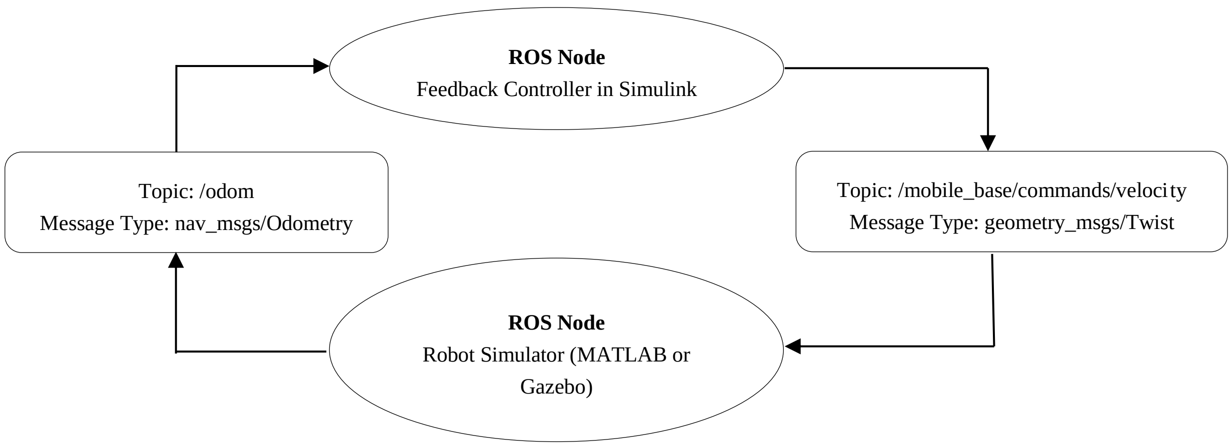

ROS Node
Feedback Controller in Simulink
Topic: /odom
Message Type: nav_msgs/Odometry
Topic: /mobile_base/commands/velocity
Message Type: geometry_msgs/Twist
ROS Node
Robot Simulator (MATLAB or Gazebo)

Fig. 9. Feedback control through continuous interaction between Simulink and a Robot Simulator through topics [28].

7.2 Sign Following Robot

MATLAB [29] scripts that implement a sign-following algorithm can control a simulated robot to move along a path based on signs in the environment [30]. These scripts fetch the location and camera information from the simulated robot and detect the color of the sign. The robot is then given a velocity command based on the color. For instance, the robot shall move on encountering a green sign whereas, it shall halt on a red sign. This approach can also be implemented using Simulink [31].

7.3 ROS for Dobot Magician System

As the programming environment of the project group VARobot[6] focusses on collaborative programming using Dobot Magician System, VR environment, and AR techniques, this section gives a brief overview of the Dobot Magician System. Dobot magician system is a versatile robotic arm integrating programming, electronics, mechanics, and automation [32]. Dobot magician helps with various tasks including practical training, teaching, playback, and so forth. Magician has declared support for the ROS platform, among others such as Arduino microcontrollers.

Dobot magician consists of following body parts: a base, a forearm, a rear arm, an end effector, a power button, and an LED indicator (Figure 10). Dobot magician, in addition to Python scripting, is equipped with a graphical programming environment enabling users to write codes by selecting and piling blocks. Dobot magician implements cross-functionality for STEAM teaching, it can perform the following tasks with high-precision and a user-friendly UI [33]:

- *Pick and place.* The students can simulate a real-time factory environment by adjusting the speed of the arm moving objects from one place to another with the help of the gripper or the vacuum pump kit included with the dobot magician system.
- *Laser Engraving.* The dobot magician enables the users to engrave patterns on various materials such as leather and wood. Dobot magician is equipped with the latest and advanced 405nm 500mW laser tube of blue-violet emission.
- *3D Printing.* Dobot magician can precisely implement dual color 3D printing by exploiting the two extra stepper motor ports.
- *Write and Draw.* The dobot magician can draw smooth lines with high-precision and stability, the user interface empowers the students to customize the image's size and position. In addition to this, dobot magician allows the students to use their personalized texts and images.

[6] https://cs.uni-paderborn.de/dbis/lehre/veranstaltungen/project-group-varobot/information/

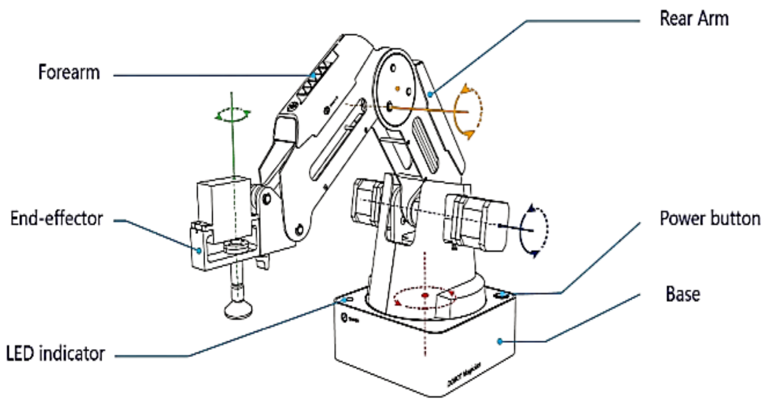


Fig. 10. Dobot Magician [27]

8 CONCLUSION AND FUTURE WORK

This paper explains the various components included in the Robot Operating System (ROS) along with the commonly encountered use-cases. The paper has also described the benefits of integrating augmented reality and virtual reality with robotics to develop highly efficient machines that can perform tasks that are imperceptible and beyond human reach. The increasing intuitiveness towards building intelligent robots and decreasing the complexity of the robotic software development process demand a dynamic prototype to integrate multiple technologies and execute various features.

A lot of research needs to be done to introduce methods and enhance the ROS even further such that it reduces the programming time and efforts required to build large-scale robot software systems with high accuracy. New human-robot-interfaces that can be easily operated and comfortably used by a range of clients from experienced professionals and organizations to users with negligible programming knowledge need to be developed by using a model-driven development approach integrating human-computer interaction (HCI) interfaces.

References

1. Swider, M., 2020. Samsung CES 2020: the best thing at the booth is this salad-making Chef Bot. TechRadar India. https://www.techradar.com/in/news/samsungs-bot-chef-made-me-a-salad-at-ces-2020-and-i-ate-it. Accessed 5 May 2020.
2. Quigley, M. et al., 2009: ROS: An open-source robot operating system. In: Workshops at the IEEE International Conference on Robotics and Automation.
3. Ros.org. n.d. ROS.Org | About ROS. https://www.ros.org/about-ros/. Accessed 6 May 2020.
4. Mišeikis, J., 2018. INF3480 – Introduction to Robot Operating System [Lecture Notes]. https://www.uio.no/studier/emner/matnat/ifi/INF3480/v18/timeplan/lecturenotes/inf3480-part-a---ros---spring-2018---simplified.pdf.
5. O'Kane, J., 2014. A Gentle Introduction to ROS. Columbia, SC: Jason M. O'Kane.

6. IFR International Federation of Robotics. 2020. Top Trends Robotics 2020. https://ifr.org/ifr-press-releases/news/top-trends-robotics-2020. Accessed 5 May 2020.
7. Ohio University, n.d. Top 5 Industries Utilizing Robotics | Ohio University. https://online-masters.ohio.edu/blog/5-industries-utilizing-robotics/. Accessed 10 May 2020.
8. Kumar, S., n.d. Introduction to ROS (Robot Operating System) – Geeksforgeeks. Geeksfor-Geeks. https://www.geeksforgeeks.org/introduction-to-ros-robot-operating-system/. Accessed 9 May 2020.
9. Gerkey, B., 2011. What is ROS Exactly? Middleware, Framework, Operating System? – ROS Answers: Open Source Q&A Forum. Answers.ros.org. https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/#18055. Accessed 7 May 2020.
10. En.wikipedia.org. n.d. Robot Operating System. https://en.wikipedia.org/wiki/Robot_Operating_System#:~:text=Robot%20Operating%20Sy
tem%20(ROS%20or,frameworks%20for%20robot%20software%20development).&text=
Despite%20the%20importance%20of%20reactivity,%2Dtime%20OS%20(RTOS). Accessed 7 May 2020.
11. Wiki.ros.org. n.d. ROS/Introduction – ROS Wiki. http://wiki.ros.org/Parameter%20Server. Accessed 20 May 2020.
12. Wiki.ros.org. n.d. Vision_Opencv – ROS Wiki. http://wiki.ros.org/vision_opencv. Accessed 23 May 2020.
13. Rusu, R.B and Cousins, S., 2011. 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China. Accessed 21 May 2020.
14. Moveit.ros.org. n.d. Moveit Motion Planning Framework. https://moveit.ros.org/. Accessed 23 May 2020.
15. Wiki.ros.org. n.d. Rostopic – ROS Wiki. http://wiki.ros.org/rostopic. Accessed 25 May 2020.
16. Wiki.ros.org. n.d. ROS/Tutorials/Recording and Playing Back Data – ROS Wiki. http://wiki.ros.org/ROS/Tutorials/Recording%20and%20playing%20back%20data. Accessed 28 May 2020.
17. Wiki.ros.org. n.d. Tf – ROS Wiki. http://wiki.ros.org/tf. Accessed 28 May 2020.
18. Jen, Y. H., Taha, Z. and Vui, L. J., 2008. VR-Based Robot Programming and Simulation System for an Industrial Robot. In: International Journal of Industrial Engineering, 15(3), 314-322, 2008. Accessed 24 May 2020.
19. Whitney, D., Rosen, E., Ullman, D., Phillips, E., and Tellex, S., 2018. Ros reality: A virtual reality framework using consumer-grade hardware for ros-enabled robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Accessed 03 June 2020.
20. Wiki.ros.org. n.d. Artoolkit – ROS Wiki. http://wiki.ros.org/artoolkit. Accessed 25 May 2020.
21. ARPOST, 2018. AR and VR Technologies Guide Robots Towards a Smarter Future. ARPOST. https://arpost.co/2018/09/19/ar-and-vr-technologies-guide-robots-towards-a-smarter-future/. Accessed 16 May 2020.
22. Joshi, N., 2019. How AR, VR, and robotics can work together. Allerin. https://www.allerin.com/blog/how-ar-vr-and-robotics-can-work-together. Accessed 16 May 2020.
23. Kharpal, A.,2017. Elon Musk's A.I. project is making humans teach robots through VR in a big breakthrough moment. CNBC, https://www.cnbc.com/2017/05/17/elon-musk-openai-robot.html#:~:text=Elon%20Musk's%20A.I.,in%20a%20big%20breakthrough%20moment&text=OpenAI%20has%20devel-
oped%20a%20way,and%20carry%20out%20the%20task. Accessed 27 May 2020.

24. ATLAS Institute. 2018. Augmented reality enhances robot collaboration. University of Colorado Boulder. https://www.colorado.edu/atlas/2018/03/15/augmented-reality-enhances-robot-collaboration. Accessed 30 May 2020.
25. Houser, K., 2017. An AI Robot Learned How to Pick up Objects After Training Only in the Virtual World. Futurism. https://futurism.com/an-ai-robot-learned-how-to-pick-up-objects-after-training-only-in-the-virtual-world. Accessed 30 May 2020.
26. In.mathworks.com. n.d. Simulink – Simulation And Model-Based Design. https://in.mathworks.com/products/simulink.html. Accessed 14 June 2020.
27. Gazebosim.org. n.d. Gazebo. http://gazebosim.org/. Accessed 14 June 2020.
28. In.mathworks.com. n.d. Feedback control of a ROS-Enabled Robot – MATLAB & Simulink – Mathworks India. https://in.mathworks.com/help/ros/ug/feedback-control-of-a-ros-enabled-robot.html. Accessed 14 June 2020.
29. In.mathworks.com. n.d. MATLAB – Mathworks. https://in.mathworks.com/products/matlab.html. Accessed 19 June 2020.
30. In.mathworks.com. n.d. Sign Following Robot with ROS in MATLAB – MATLAB & Simulink – Mathworks India. https://in.mathworks.com/help/ros/ug/ros-sign-following.html. Accessed 19 June 2020.
31. In.mathworks.com. n.d. Sign Following Robot with ROS in Simulink – MATLAB & Simulink – Mathworks India. https://in.mathworks.com/help/ros/ug/sign-following-robot-using-ros-simulink.html. Accessed 19 June 2020.
32. Robotlab.com. n.d. Dobot Magician Robotic EDU Edition. https://www.robotlab.com/store/dobot-robotic-arm. Accessed 18 May 2020.
33. Dobot Magician User Guide V1.5.1, 2018. Shenzehn, China: ShenZehn Yuejiang Technology Co., p23. https://www.idesignsol.com/files/Dobot-Magician-User-Guide-V1.5.1.pdf. Accessed 03 June 2020.
34. Thomas, D., n.d. Changes Between ROS 1 And ROS 2. Design.ros2.org. http://design.ros2.org/articles/changes.html#:~:text=In%20ROS%201%20the%20developer,haven't%20been%20implemented%20yet. Accessed 6 July 2020.