LAB 1 - R-IDE PRODUCT DESCRIPTION

Justin Tymkin

Old Dominion University

CS411

Professor J. Brunelle

March 13, 2023

Final Version

Table of Contents

Figures

Tables

1. Introduction

Robot Operating System (ROS) is a flexible and powerful open-source framework that is widely used in robotics and development. ROS has gained significant attention in recent years due to its modular and scalable architecture that enables researchers and developers to build complex robotics systems easily. ROS has been increasingly used in various domains, including space robotics, autonomous vehicles, and industrial automation.

Fujita (2018) discussed the use ROS in the development of the Aibo robot, which was used as a companion robot for astronauts on the International Space Station (ISS), in a presentation at ROSCon 2018. The Lunar Rover is another example of a robot that uses ROS for its software system, as reported by the Open Robotics website (2021). According to the Global Robot Operating System Market Research Report (2022), the robotics industry has grown significantly over the past decade, with ROS being the software of choice. The report estimates that the industry is currently worth 249.2 million and is expected to reach a value of nearly double that amount in the next decade.

Although ROS works in a multitude of developer environments, often these environments are overwhelming and frustrating for new users. The ROS system uses a node-based architecture for communication between different components. Each node is a process that communicates with other nodes using messages. Learning how to create, configure, and manage ROS nodes is challenging for new developers. Programming packages for robotics using ROS on a Linux system require multiple terminal windows, leading to a messy User Interface/User Experience (UI/UX). As a result of the complex architecture of ROS and the messy UI/UX, debugging nodes in ROS will also be a challenge for new developers.

A solution to the challenges faced by new developers using ROS is the development of an integrated development environment (IDE) that simplifies the process of creating and debugging nodes. A single workspace should consolidate the necessity of having multiple terminal windows for individual nodes, which the IDE should provide an interface for. Other tools to assist in helping new developers overcome the steep learning curve of ROS are a tool to streamline the process of creating a node, a tool to hasten code completion, and a tool to visualize information about the nodes. Empowering new ROS developers and helping to overcome the steep learning curve associated with ROS, offering the ability to create and debug nodes in a single environment is crucial. By reducing the learning curve required to navigate ROS, developers will devote more time to coding enhancements to the robot. This increased focus on coding will improve the effectiveness of new developers in the project.

2. R-IDE Product Description

ROS-Integrated Development Environment (R-IDE) is an extension that facilitates the development of robotics and automation projects using ROS. It is designed as an IDE for ROS users, with a user-friendly interface that is accessible through the VSCode marketplace. By integrating with VSCode, R-IDE offers an intuitive and streamlined user experience for new ROS users developing robotics programs. Features included in R-IDE: wizards to allow users to quickly create components within a ROS package, the ability to record and play back data in a ROS bag, visual tools such as Rviz and node graphs for users to visualize and understand the behavior of nodes created, and a topic monitor to track and analyze data as it flows between different nodes in a project.

2.1 Key Product Features

Providing an extension within VSCode for ROS development provides a well-organized and user-friendly environment, resulting in an enhanced UI/UX. VSCode offers a range of options such as easy navigation between R-IDE, the text editor, and the debugging buttons on the side of the window. Additionally, it allows for opening new windows, which enables ROS users to work on multiple nodes or debug multiple packages simultaneously.

One unique feature of R-IDE is the creation wizard that offers the ability to create ROS nodes, messages (msg), or services (srv) without the need to navigate through multiple terminal windows. It eliminates the hassle of keeping track of various nodes communication by having everything in one place. The wizard guides the user through the creation steps and sends generic templates to the text editor, making it easier to begin programming.

Another useful feature of R-IDE is the ability to record and play back a ROS bag. This feature allows users to store ROS msg data and replay it to understand any mistakes they may have made. The ROS bag records the raw data collected by the robot using the users program, providing flexibility in learning what is happening with the robot.

R-IDE also offers support for ROS developers through its integrated tools such as Rviz and RQT. Rviz is a 3d visual tool that replays what the user has programmed the robot to see. RQT, on the other hand, offers a range of tools for ROS users, including the node graph, which shows how the current node communicates other nodes. Additionally, RQT includes the topic monitor tool, which allows users to select a ROS topic they want to look at, showing its types, bandwidth, and Hz of all topics. Lastly, the message publisher tool tracks the message type to publisher and the frequency for publishers in Hz.

2. 2 Major Components

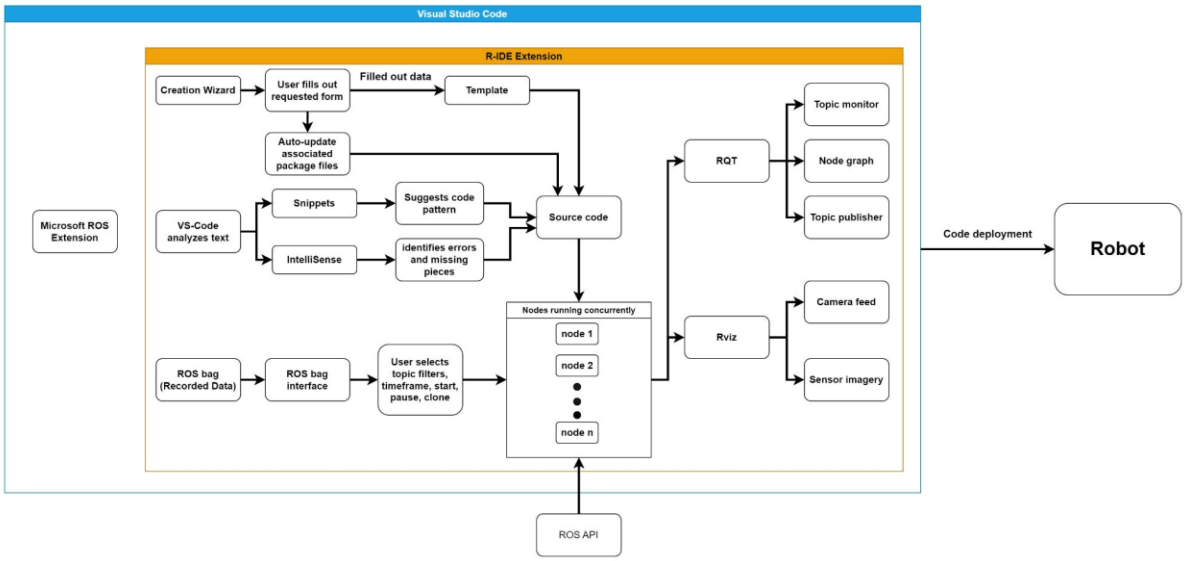
R-IDE consist of several key components for ROS users, as shown in the Major Functional Component Diagram (MFCD) in Figure 1. These components consist of the creation wizard, VSCode text analyzer, ROS bag support, and visual tools in Rviz and RQT. The creation wizard offers an easy-to-use interface for creating nodes, messages, or services within a ROS package. The user can select one of the three options, then the wizard will walk the user through the process of creating it, for example: what language should this be written in, the name of this file, and where should this file be saved.

After the user has completed creating the new node, message, or service, the file will auto fill with generic templated code provided by R-IDE. The user can then begin developing, while the user is developing VSCode text analyzer will assist in commonly used words or phrases, using the Intellisense built inside VSCode, to speed up the process. R-IDE will also have pre-determined ROS snippets to assist in streamlining the developing process. After creating a ROS package with the R-IDE wizard, the user can utilize the debugger in VSCode to ensure everything works. With the templated code, the VSCode Intellisense, and the ROS snippets, the debugging process is quicker and still offers the ability to set breakpoints, step through code, or inspect variables.

If the user has successfully created a ROS package, the ROS bag and visual tools will help to determine if the package is working. The ROS bag records data collected from the package running, then the user can play it back. Rviz will create a 3D visualization of what the robot currently sees if the package is running. If the nodes are communicating, the node graph will show the user what subscriber nodes are communicating with publisher nodes, and what topics are being used to communicate between them.

Figure 1

R-IDE Major Functional Component Diagram



3. Identification of Case Study

R-IDE is developed by students from the Computer Science (CS) department of Old Dominion University (ODU), for students from the Electrical and Computer Engineering (ECE) department of ODU. ODU competes in the Intelligent Vehicle Competition (IGVC) annually, led by Dr. Belfore of the ECE department. The ECE team competes in the autonomous vehicle competition with the goal of having the Monarch 1, the autonomous vehicle at ODU, drive and navigate through a course by itself. Dr. Belfore and the CS team believe that R-IDE will be a dynamic tool for the ECE team in competition as it will help to lower the learning curve of ROS and streamline the development process of ROS for the Monarch 1 before the IGVC.

4. Product Prototype Description

R-IDE is a full extension for VSCode, as proof of concept the extension creates functional nodes, services, and messages. Additionally, it can play back previously recorded ROS bags from Dr. Belfore. By integrating R-IDE as an extension within VSCode, the CS team can develop in a risk-mitigated environment. To ensure the project expectations are met, the CS team meets weekly with Dr. Belfore, and has open dialogue with the ECE team working on the Monarch 1.

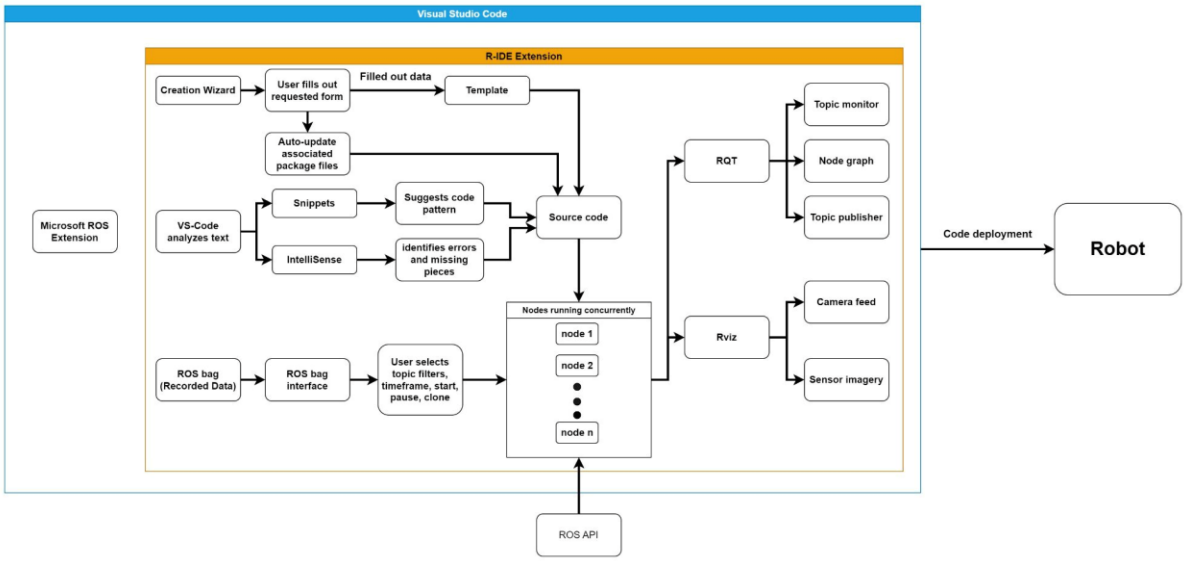
4.1 Prototype Architecture (Hardware/Software)

To use R-IDE, the user must have a computer that meets the hardware requirements to run VSCode, ROS, and Rviz. The software requirements include access to VSCode and the ability to run either ROS 1 or ROS 2. ROS 1 operates within a Linux environment, whereas ROS 2 can run on Linux, Mac, or Windows system.

The R-IDE extension is designed to operate within the VSCode environment and works in tandem with the Microsoft extension for ROS. Figure 2 below shows the components of R-IDE, the creation wizard guides the user in filling out a form that generates a code templated for common ROS components. Customized code snippets and Intellisense are suggested and applied by VSCode to assist in code development. The ROS bag interface included with R-IDE is responsible for manipulating and managing ROS bags. Various visualization tools embedded within VSCode display output data. However, it should be noted that code deployment to the robot is outside the scope of R-IDE and is handled separately.

Figure 2

R-IDE Major Functional Component Diagram



4.2 Prototype Features and Capabilities

Table 1 presents an overview of all the functionalities and features available in R-IDE. The creation wizard enables the user to create nodes, messages, and services with the help of customized snippets designed specifically for ROS users. Upon completion of the wizard, R-IDE automatically updates the package .xml and cmake file and creates a generic code template for the user-defined node, service, or message.

The ROS bag interface in R-IDE facilitates functionalities such as starting, stopping, and recording of ROS bag files. This feature enables users to review previously collected data at a later time. The visual tools, including Rviz, node graph, topic monitor, and message publisher, provide additional insights into data collection and node communication. R-IDE collects usage analysis data to determine the popularity of specific features, which will help identify areas for

improvement or optimization. Additionally, R-IDE employs mock data and automated testing, only for the developers of R-IDE, to test new features and ensure functionality.

Table 1

R-IDE Real World Product vs. Prototype

	Feature	RWP	Prototype
Wizard	Create node	Full	Full
	Create msg	Full	Full
	Create srv	Full	Full
Auto update	cmake file	Full	Full
	package.xml	Full	Full
Snippets	Autocomplete Code Patterns	Full	Full
ROS bag	Start Rosbag recording	Full	Full
	Stop Rosbag recording	Full	Full
	Play back Rosbag	Full	Full
Visuals	rviz	Full	Partial: Depending on performance of embedded features
	Node Graph	Full	Full
ROS Topic	ROS topic monitor	Full	Full
	ROS topic publisher	Full	Full
Data Management	Usage Analysis	Full	Full
Test Management	Create Mock Data	Eliminated	Full
Test Management	Automated Tests	Eliminated	Full

4.3 Prototype Development Challenges

Developing R-IDE for the ECE team requires weekly meetings to ensure that the CS team satisfies the requirements of Dr. Belfore and the ECE team, while establishing realistic expectations for R-IDE features and capabilities. As R-IDE is intended to function within VSCode, it must adhere to VSCode extension standards to be released in the marketplace. The creation of R-IDE mandates a wealth of ROS expertise, which presents a steep learning curve for the CS team to meet the requirements set by the ECE team. ROS development involves multiple terminal windows, and R-IDE in VSCode should highlight syntax errors, and proper cursor and line placements across all active windows. The integration of existing tools into VSCode and ensuring their seamless functionality is another challenge for the CS team. The current version of

R-IDE is developed for ROS 1, but the developers for ROS have stated they will no longer update ROS 1 as they work on the newer version ROS 2. R-IDE will function in collaboration with ROS 1 in VSCode, but there are compatibility issues with ROS 2 and R-IDE. Finally, given the age of ROS, locating documentation for ROS is a daunting task.

Glossary

Autonomous Machine: A machine capable of sensing its environment, carrying out computations to make decisions, and performing actions in the real world.

Command Palette: refers to the input box in VSCode where users can enter commands.

Filtering: Create new ROS bag files from older ROS bag files, filtering the new bag based on requested ROS topics from the old one.

Git: Git is a distributed version control system: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Graph View: a graphical representation of the communications between different nodes in a ROS system. It provides a visualization of the ROS topics being published and subscribed to by each node. Graphical View is a useful tool for understanding and debugging complex ROS systems.

GUI (Graphical User Interface): type of user interface that has elements such as buttons, menus to allow the user to interact with the application.

IDE (Integrated Development Environment): application software that provides tools and features for software development. They include features like debugger, source code editor, code completion and even version control in some cases.

IntelliSense: IntelliSense is a general term for various code editing features including: code completion, parameter info, quick info, and member lists.

Monarch 1: two-passenger autonomous vehicle design based on Polaris GEM e2, which is an electric vehicle that is intended to participate in the IGVC's "Self-Drive Challenge".

MongoDB: free non-relational database that stores data in JSON format. The data is stored in collections of documents that can have nested structures for flexible data storage.

Mongoose: library for Node.js that provides a higher level of abstraction on top of MongoDB that simplifies the process of interacting with MongoDB.

RQT: A QT-based framework for GUI development for ROS. It contains tools that support ROS topics, bags, node graphs, and many other tools for visualization and manipulation of ROS nodes.

ROS Bag: a file format in ROS for storing ROS message data. An important role in ROS, and a variety of tools have been written to allow the user to store, process, analyze, and visualize them. Bags are the primary mechanism in ROS for data logging, which means that they have a variety of offline uses.

ROS Master: The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. The Master also provides the Parameter Server and is most commonly run using the roscore command, which loads the ROS Master along with other essential components.

ROS Messages: A message is a simple data structure, comprising typed fields. Nodes communicate with each other by publishing messages to topics. Standard primitive types

(integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays. As part of a ROS service call, a message can be exchanged between nodes in the form of request and response.

ROS Node: executable programs to perform tasks within the robotic system. Nodes are combined together and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on.

ROS Package: software in ROS is organized by packages, a collection of ROS nodes, a ROS library, a dataset, or anything else that requires a useful module for basic functionality.

ROS Parameter Server: A parameter server is a shared, multivariate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. As the server is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters. The server is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify it if necessary.

ROS Services: provide a way for nodes to communicate with each other in a synchronous, request-response fashion. The request / reply is done via a Service, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call.

ROS Subscriber: Node that receives data from a topic published by another node. Nodes use ROS topics to communicate, and a subscriber subscribes to a topic to receive messages that are published on that topic.

ROS Topics: Named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. Topics are intended for unidirectional, streaming communication.

Robot Operating System (ROS): set of software libraries that helps to build robot applications. Ranging from drivers to algorithms and powerful developer tools, ROS is the preferred tool for robotics projects.

Rviz: (Ros Visualization) A 3D visualizer for displaying sensor data and state information from ROS.

Snippets: A coding tool to automatically generate repeating code when developing.

Template: An editable text or code snippet that can be filled with given values from another tool like a wizard.

Topic Monitor: Displays debug information about ROS topics including publishers, subscribers, publishing rate, and ROS messages.

Tutorial: A method of transferring knowledge that teach via example and supplies the information to complete a given task.

Trimming the ROS bag time range: Reducing the duration in time of a requested ROS bag.

VSCode Extension: A tool designed to add additional features and capabilities to VSCode. It can create new dialogues, add functions, or change the appearance of VSCode.

WebView: A type of web browser that can be embedded within applications allowing it to display web content.

Wizard: A user interface that presents dialog to lead a user through a sequence of steps. Often used to configure a service for the first time or to simplify a complex or unfamiliar process.

References

Ackerman, E. (2021, June 24). Microsoft announces experimental release of ROS for Windows 10. Retrieved November 3, 2022, from https://spectrum.ieee.org/microsoft-announces-experimental-release-of-ros-for-windows-10

Belfore, L. (jul 8, 2022). *Software Design Report for the ODU Monarch I* (Vol. 1.3, p. 3, Tech.).

Cardona, M., & Manzanares, J. (2020). *COVID-19 Pandemic Impact on Mobile Robotics Market* (pp. 1-4) (A. Palma, Ed.). IEEE. doi:10.1109/ANDESCON50619.2020.9272052

Fujita, T. (2018). Tomoya Fujita R&D center Sony Corporation - roscon.ros.org. Retrieved November 3, 2022, from https://roscon.ros.org/2018/presentations/ROSCon2018_Aibo.pdf

Gerkey, B. (2014, September 1). Ros running on ISS. Retrieved November 3, 2022, from https://www.ros.org/news/2014/09/ros-running-on-iss.html

Global Robot Operating System Market Research Report 2022(status and outlook). (2022, June 29). Retrieved November 3, 2022, from https://www.marketreportsworld.com/global-robot-operating-system-market-21185690

Guerry, M., Müller, C., Kraus, W., & Bieller, S. (2021, October 28). IFR International Federation of Robotics. Retrieved November 3, 2022, from https://ifr.org/downloads/press2018/2021_10_28_WR_PK_Presentation_long_version.pdf

Lunar Rover. (2021). Retrieved November 3, 2022, from https://www.openrobotics.org/customer-stories/lunar-rover

Robot Operating System Market. (2022, August). Retrieved November 3, 2022, from https://www.futuremarketinsights.com/reports/robot-operating-system-market

Robot operating system. (n.d.). Retrieved November 3, 2022, from https://www.ros.org/

Wessling, B. (2022, February 11). Open robotics developing space ROS with Blue Origin, NASA. Retrieved November 3, 2022, from https://www.therobotreport.com/open-robotics-developing-space-ros/