Lab 2 – R-IDE Prototype Specification

Dominik Soós

Old Dominion University, Computer Science Department

CS411W - Professional Workforce Dev. II

Professor J. Brunelle

April 25th, 2023

Version 2

Table of Contents

List of Figures

List of Tables

1. Introduction

Robot Operating System is a widely used open-source framework that is popular among new and experienced developers for building robotics applications. It has gained recognition for its ability to simplify the process of creating complex and sophisticated robot behavior by enabling developers to focus on the core functionalities of their applications. ROS has a growing research community of developers working with the framework. As the demand for robotics in various industries, such as car manufacturing, logistics, and healthcare, continues to grow, the market for ROS and other robotics frameworks is expected to rise significantly.

Despite its numerous benefits, ROS presents high barriers of entry for new developers and environments. Setting up and configuring a ROS environment can be time-consuming and require specialized knowledge. This process includes installing the ROS middleware in a Linux environment, creating a new catkin workspace, and setting up environment variable, which can be difficult for experienced developers, especially difficult for beginners who are not familiar with the ROS architecture and conventions. Additionally, there is a steep learning curve, as ROS requires specialized knowledge and an extensive list of terminologies. New users can often find themselves feeling overwhelmed to get started due to the system that consists of large number of libraries, tools, and conventions that they must understand first before they can start developing useful software.

Debugging can be a significant challenge when using ROS. The framework is a complex system with many interconnected components, implying that identifying and fixing errors can be a daunting task. ROS applications need to run on multiple terminal windows, requiring developers to switch frequently between them to manage and debug their code. Debugging and managing the changes between nodes can also be challenging as well as organizing the ROS

interface. Further complicating the learning process for new developers, the documentation is outdated and confusing.

To address these challenges faced by new and experienced developers when working with ROS, Robotics Integration & Development Environment (R-IDE) was developed. R-IDE is a Visual Studio Code extension that simplifies the development of ROS applications, with a focus on streamlining the development lifecycle and the learning process for ROS developers.

1.1 Purpose

The purpose of R-IDE is to simplify the development of ROS applications and make it more accessible for new developers and experienced ones. With a streamlined developer environment, R-IDE aims to address the challenges of having multiple terminal windows, managing multiple nodes simultaneously, inefficient debugging and file organization, and navigating outdated documentation.

By consolidating common tasks and commands, R-IDE offers a simplified experience that allows users to focus on creating the project without being overwhelmed by the complexities of the ROS architecture. R-IDE provides access to visualization tools and organizes the developer's ROS interface, making it easier to navigate and work with. R-IDE's setup process and template system are designed to simplify the development process even further. The setup process allows developers to quickly create ROS projects, while the template system provides templates for creating nodes in both C++ and Python, building launch files, and handling other common tasks in ROS development. This feature saves developers time and effort, enabling them to focus on building their applications rather than spending time setting up their environment.

Another key feature of R-IDE is its debugging capabilities for ROS nodes and ROS bag recording. By providing these capabilities, R-IDE makes it easier for developers to troubleshoot and optimize their ROS applications. Overall, the R-IDE extension aims to reduce the steep learning curve associated with working with ROS, making it more accessible and user-friendly for new developers.

R-IDE aims to simplify the development lifecycle and learning process for ROS developers, including Dr. Lee Belfore, who is an Associate Professor of Electrical & Computer Engineering at Old Dominion University. The extension provides a user manual for the setup process, which enables Dr. Belfore and his engineering students to create ROS projects more quickly than they could before the introduction of R-IDE. R-IDE's template system takes the development process a step further by providing code templates for creating publisher and subscriber nodes in both C++ and Python. With R-IDE, code typing is not required to set up the ROS environment and start executing a sample node. It facilitates the building of launch files, enabling ROS developers to visualize ROS development using the ROS Topic Monitor and Node Graph features that come with R-IDE.

1.2 Scope

R-IDE focuses on three main areas of improvement: package management, a graphical user interface (GUI) for command-line functions, and code templates for rapid ROS package development.

Automating package management is one of the main features of R-IDE, helping users efficiently create, manage, and understand ROS packages. This feature simplifies and streamlines the process, enabling the developer to focus on creating the project without being overwhelmed by the complexities of package management and file updating.

The GUI for command-line functions is another area that R-IDE focuses on. The GUI offers a user-friendly approach to accessing command-line tools, minimizing the learning curve for new ROS developers. The R-IDE GUI enables users to perform the necessary tasks with ease and efficiency, reducing the time and effort required to work with the ROS architecture.

R-IDE also provides code templates for rapid ROS package development. This feature is designed to save developers time and effort by providing pre-made templates for creating nodes in both C++ and Python, building launch files, and handling other common tasks in ROS development. The templates provided by R-IDE simplify the development process, enabling developers to get started with ROS quickly and easily.

Finally, to ensure that the R-IDE extension remains up-to-date and reliable, the development team adheres to the guidelines and updates of the dependent software while also actively seeking feedback from stakeholders to constantly improve the product. With a focus on streamlining the development lifecycle and the learning process for ROS developers, R-IDE is a powerful tool for simplifying the development of ROS applications.

[THIS SPACE INTENTIONALLY LEFT BLANK]

1.3 Definitions, Acronyms, and Abbreviations

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.

1.4 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/

Peterson, J., Soós, D., Koontz, D., Tymkin, J., & St. Clair, G., (2023). Lab 1 – R-IDE Extension Product Description. In J. Brunelle (Ed.), CS 411W: Professional Workforce Development. Old Dominion University

1.5 Overview

This lab report provides an overview of the R-IDE software configuration, external interfaces, purpose, and scope. It also outlines the key features of R-IDE, including its ability to simplify common tasks and commands, provides visualizations tools, and organizes the developer's interface. With R-IDE, ROS developers can expect a more streamlined and efficient development process.

2. General Description

R-IDE, comprehensive VSCode extension, is thoughtfully designed to cater to the development and testing needs of ROS applications. This fully functional extension equips ROS developers with a wide range of tools and features that simplify and expedite the processes of building and debugging. With R-IDE, developers can seamlessly create new functional nodes, services, and messages as a proof of concept, while also gaining access to previously recorded ROS Bags from Dr. Belfore for playback, making it an invaluable asset for ROS developers.

2.1 Prototype Architecture Description

R-IDE is a VS-Code extension that coexists with Microsoft's ROS extension. Figure 1 illustrates where R-IDE is theoretically located along with the major functional components. R-IDE showcases a creation wizard that prompts the user to complete a form, generating code templates for standard ROS components such as publisher and subscriber nodes for both C++ and Python. Publisher and subscribers are key components that facilitate communication between different nodes in the ROS system. A publisher is a node that produces and broadcasts data (also known as messages) on a specific topic. This data can be sensor readings, control commands, or any other relevant information. Publishers ensure that the data is available to any other nodes that might be interested in receiving it. A subscriber, on the other hand, is a node that

listens for and receives data on a particular topic. Subscribers consume the data published by the publishers and use it for further processing, analysis, or control actions. They subscribe to specific ROS topics and automatically receive updates whenever new data is published on those topics.

The ROS Bag Manager interface empowers users to select, clone, play/pause, stop and replay ROS Bags with ease. The output data is subsequently displayed through the ROS Topic Monitor embedded within the extension. Users can manipulate their chose ROS Bag, with options to clone the entire or trimmed version of the ROS Bag or play the bag. During playback, users have to ability to pause, stop and replay the selected ROS Bag. The incoming messages for from the ROS Bag can be displayed in the ROS Topic Monitor in order to simplify the debugging process for developers. R-IDE does not oversee code deployment to robots. The event data is being collected and stored in a MongoDB database using Mongoose to facilitate usage analytics. This process enables the developers of the R-IDE extension to track user interaction

and handle potential errors.

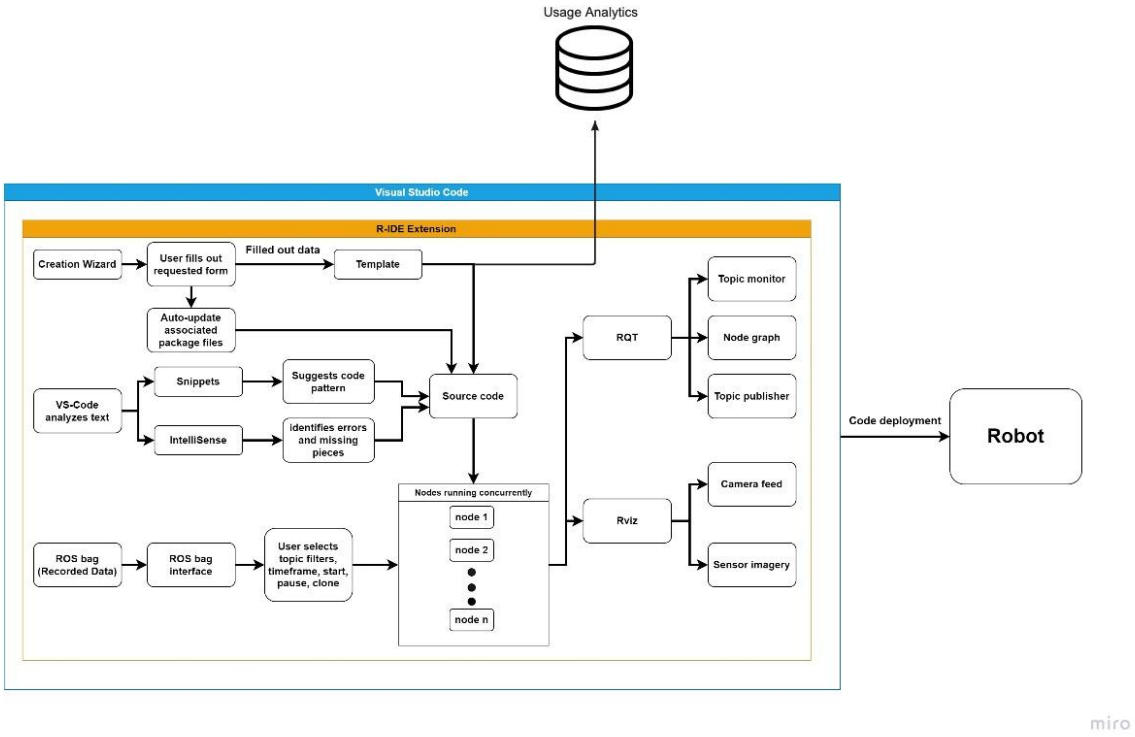

Figure 1 Major Functional Component Diagram

2.2 Prototype Functional Description

R-IDE has several features that make it a useful tool for ROS developers. R-IDE is equipped with several features that enhance the user experience when developing ROS applications. These features include the Creation Wizard for common files, autoupdate package files, autocomplete common code patterns, ROS Bag features, visualization tools, ROS Topic features and data management. Table 1 illustrates that all of these features are going to be available in the R-IDE prototype as well as the real world product.

The debugging capabilities within R-IDE allows the developer to conveniently debug and terminate ROS nodes. This feature simplifies the debugging process, enabling developers to focus on their application logic and to identify issues more efficiently.

R-IDE's advanced debugging capabilities greatly simplify the debugging process for developers by allowing them to efficiently debug and terminate ROS nodes without the need for cumbersome manual interventions. This functionality streamlines the troubleshooting process, enabling developers to focus on refining their application logic and more effectively identifying potential issues. By enhancing the debugging experience, R-IDE significantly reduces the time spent on troubleshooting, which in turn results in faster development and deployment of ROS applications. Moreover, the debugging capabilities in R-IDE also facilitate better collaboration among development teams, as they can easily share and resolve identified issues.

The Creation Wizard in R-IDE is a powerful and versatile tool that expedites the process of generating publisher and subscriber nodes in both C++ and Python programming languages. By automatically producing message and service files, the wizard saves developers valuable time and minimizes the potential for errors during the initial development stages. This feature not only accelerates the creation of ROS components but also ensures that they are consistent with ROS standards, resulting in more reliable and robust applications. The user-friendly interface of the Creation Wizard also contributes to a more intuitive and efficient development experience for both novice and experienced ROS developers.

R-IDE seamlessly integrates auto-update functionality for CMakeLists.txt and package.xml files, ensuring these critical files remain current with the existing ROS workspace. This feature prevents potential conflicts and errors that can arise from outdated or incompatible files, thereby enhancing the overall stability of the ROS application. Additionally, the autocomplete feature for common code patterns in C++ and Python employs IntelliSense technology to offer suggestions based on ROS standards. By enhancing the overall code quality, reducing the likelihood of syntax errors, and streamlining code writing, R-IDE helps developers

create well-structured and efficient ROS applications that are easier to maintain and extend in the future.

The ROS Bag interface in R-IDE empowers users to effortlessly play, clone, and manipulate ROS Bags within the ROS workspace, enabling a more streamlined and efficient approach to working with recorded data. During playback, users have the option to stop or resume the ROS Bag, allowing them to analyze specific segments of data in detail. When cloning a selected ROS Bag, users can trim the ROS Bag according to its duration, effectively reducing storage requirements and enabling focused analysis of relevant data segments. This feature simplifies data management and enhances the user's ability to work with recorded ROS data efficiently, making it an indispensable asset for developers working with complex robotics systems.

R-IDE incorporates advanced visualization tools such as RViz and Node Graph, which assist developers in comprehending and evaluating behavior within the ROS Topic Monitor. RViz enables users to visualize what the robot perceives in real-time, providing valuable insight into potential error sources and facilitating a better understanding of the robot's actions in the environment. This feature is particularly useful when working with sensor data or testing complex robotic behaviors. Node Graph allows developers to visualize the relationships and communication pathways between nodes in a graphical manner, aiding in the identification of bottlenecks, data flow issues, and other potential problems within the ROS network. These visualization tools provide developers with a deeper understanding of their applications, leading to more robust and efficient robotic systems.

The ROS Topic Monitor offers a mechanism to subscribe to a designated ROS Topic that is currently running and display the messages of the selected ROS Topic's coming through.

While running, R-IDE actively listens to the current ROS Topics and dynamically updates the ROS Topic Monitor. The ROS Topic Publisher makes it seamless to enter message data and publish it to the appropriate ROS Topics, ensuring the correct message format is being published, streamlining the communication within the ROS ecosystem. Usage Analytics is collecting data about the Creation Wizard and uploading the data via Mongoose.

	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

Table 1 Real World Product Features Table

2.3 External Interfaces

R-IDE will use specific software, and user interfaces for users to interact with all components of the application. These external interfaces include Virtual Studio Code, ROS 1, and ROS 2. Virtual Studio Code is a popular open-source code editor developed by Microsoft, while ROS 1 and ROS 2 are frameworks used to build ROS applications. These interfaces are essential for R-IDE to function correctly and efficiently.

2.3.1 Hardware Interfaces

There are no hardware interfaces for the R-IDE extension.

2.3.2 Software Interfaces

R-IDE requires specific software interfaces to function correctly. These interfaces include VScode API, MongoDB via Mongoose, ROS Bridge, ROSLib, RViz, and ROS. The VScode API extension allows R-IDE to interact with the VS-code editor and its APIs. MongoDB via Mongoose is used by R-IDE to monitor usage analytics and collect information about the error messages occurring while creating ROS nodes. The ROS Bridge enables communication between ROS and non-ROS systems, while ROSLib provides an interface for ROS programs to communicate with. RViz is a 3D visualization tool that displays sensor data and other information from ROS topics. Finally, ROS is the core framework used to build ROS applications.

2.3.3 User Interfaces

R-IDE provides a Graphical User Interface (GUI) that allows users to interact with the application efficiently. The Svelte Framework is used to develop the GUI.

2.3.4 Communications Protocols and Interfaces

R-IDE communicates with the database to insert event data and handle errors. However, the communication protocols and interfaces used for these purposes are not specified.

3. Specific Requirements

Separate Document