Lab 1 – R-IDE Product Description

Daniel Koontz

Old Dominion University

CS411W, Spring 2023

Profesor J. Brunelle

March 13, 2023

Final Version

Table of Contents

List of Figures

List of Tables

1. Introduction

Automation and robotics are increasingly becoming the new norm in many facets of life. For example, semi-autonomous vehicles are on roads and highways, robotic arms appear in factories and warehouses, and vacuum cleaners can clean without human intervention. Robots and autonomous machines are an important part of the future of society.

The robotic and autonomous machines industry is expected to grow substantially in the coming years. The mobile robotics market is expected to grow at 24% per year from $19 billion in 2018 to $23 billion by 2021 and further to $54 billion in 2023 (Cardona et al., 2020).

Robot Operating System (ROS) is an open-source set of tools and software libraries to help build robotics applications. ROS has an estimated market value of $270 million as of 2020 (*Market Reports*, 2022), with projected growth to $280 million in 2028 and $460 million in 2032 (*Future Market Insights*, 2022). ROS has been utilized in industries such as autonomous vehicles, aerospace, healthcare, and agriculture. ROS has been utilized by organizations such as NASA (“Lunar Rover”) (Wessling, 2022), Microsoft (Ackerman, 2021), and Sony (Fujita, 2018). ROS is made up of topics and messages and it facilitates communication between components. Topics represent the type of data and the relevance of that data. This data is passed as messages which contain the relevant data. These topics and messages can be recorded into a ROSbag file that can be played back or logged for future use. Messages can also be sent to visualization tools and can allow developers to work with an entirely simulated robot. ROS supports common components such as cameras, LIDAR, and motor controllers. These components are identified within ROS as nodes and can be visually displayed as a node graph. Most applications require many nodes.

While ROS has seen extensive use, development with ROS remains a challenge for many engineers and developers. In particular, the user experience for ROS can be a significant hurdle for newcomers. Most ROS projects involve many ROS nodes and each node is controlled through a separate terminal, screens can easily become cluttered, and it can be difficult to differentiate one process from another. Similarly, it can be difficult to identify when nodes interact with each other. Debugging for ROS applications is difficult, and often the root of the error is hidden in hundreds of lines of logs. When users search for documentation to solve these issues, they come upon articles that can be over ten years old with barren descriptions. Many pages are missing or contain broken links to other documentation and source code. All of these problems contribute to the difficulties in building and maintaining a ROS project.

R-IDE (ROS Integrated Development Environment) is an extension pack on Visual Studio Code designed to improve the user experience when developing ROS applications. Hosted on the Visual Studio Code extension marketplace, the R-IDE Graphical User Interface (GUI) expands the practical capabilities of ROS. The interface creates efficient pathways for common tasks and commands and provides visualization tools for users inside their workspace. The GUI naturally helps by having a clean and well-organized workspace. By taking advantage of R-IDE, developers can quickly build simple projects and have access to various template files to implement. R-IDE users can easily navigate source code and data to build, fix, and test ROS projects.

2. Product Description

R-IDE is an extension pack for VScode designed to simplify and speed up the development lifecycle and learning process for building ROS applications. It utilizes wizards, templates, IntelliSense, and snippets to help developers build products faster. In addition, R-IDE

includes visualization features, a ROS topic monitor, ROS debugging capabilities, and a ROSBag interface to help test and validate their robotic systems. R-IDE can be used in real-time or with a previously recorded ROSBag.

2.1. Key Product and Features

The key features of R-IDE are its simplified user interface and quick setup, and, along with these, are the added visualization features embedded directly into the integrated development environment (IDE). The primary tools involve code manipulation and completion. R-IDE uses wizards to create new files and update the relevant compilation and package files. R-IDE also takes advantage of tools natively in VSCode to allow users to autocomplete common code patterns. Most common tasks and actions are simplified to a button click or menu interface. This simplified user interface reduces the need for overreliance on outdated documentation. R-IDE's debugging capabilities improve the readability of the error log and enable users to detect errors within their code.

Visualization tools allow developers to manage and conceptualize the reality of what the components of their project currently accomplish. Developers can publish messages to topics in order to test how their system behaves on a set of known data points and in order to build better robots. Users can use the topic monitor to visualize all of the different data points their system sees at any given time. Users can manipulate ROSBags using R-IDE's ROSBag interface to identify and isolate problems that become visible while testing ROS applications.

Other tools are included in VSCode or as VSCode extensions, such as git integration, file manipulation, and terminal multiplexors. By encouraging developers to use a powerful and modern IDE, users gain access to a multitude of tools now common in any software development workspace.

2.2. Major Components (Hardware / Software)

R-IDE consists mainly of the developer environment within the VSCode IDE. It is comprised of modular extensions packaged together into a single extension pack. R-IDE takes advantage of the tools that exist in VSCode, such as Microsoft's ROS extension to provide syntax highlighting and commands to control ROS software inside the IDE.

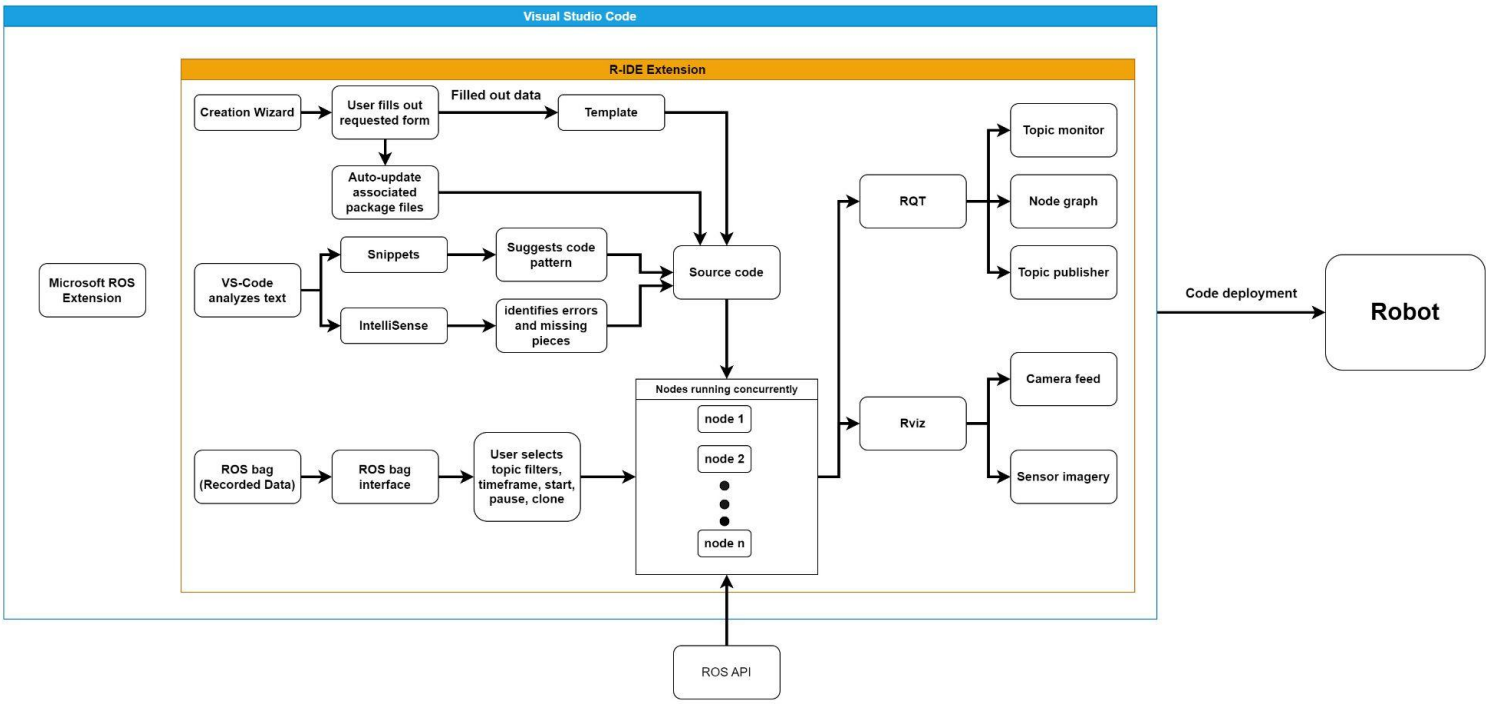
While working with code, developers have access to tools that improve development and encourage clean, readable, testable code. Users are able to access creation wizards in order to build components of their project –such as creating ROS nodes, topics, and messages. In addition, these wizards keep the necessary compilation and package files updated as the user makes changes to their source code. While the user is typing, VSCode allows R-IDE to make suggestions for building common code patterns and implement them for the user.

R-IDE allows users to interact with ROSBags through an additional view inside VSCode. Users can record, playback, clone, and filter ROSBags entirely with the interface. This gives users powerful control over ROSbags without navigating away from the source code and data stored within the IDE.

The user has visualization tools inspired by the existing GUI tools for ROS. Users can view and publish ROS topics while the ROS master is running and at the same time can also view what nodes are active and connected in the graph view. R-IDE does not handle code deployment, but there are other VSCode extensions that may assist in code deployment. The major components of R-IDE according to developers are shown in Figure 1.

Figure 1.

Major Functional Component Diagram



3. Identification of Case Study

The main users of R-IDE are ROS developers. ROS developers include people from universities and from the robotics industry. Hobbyists can also gain access to R-IDE and take advantage of its many benefits. R-IDE provides an intuitive interface for ROS development and the tools to enable new and existing developers.

The purpose of R-IDE is to improve the user experience when developing and interacting with ROS. A case study is being performed at Old Dominion University with members of the Electrical and Computer Engineering department who are currently learning and building with ROS under the direction of Dr. Lee Belfore. Students and professors use R-IDE within VSCode

as their primary development environment. Users are encouraged to use the templates and wizards in R-IDE to build initial ROS nodes, services, and messages. The developers and maintainers of R-IDE can access usage data so that fixes can be made as quickly as possible. Thus these developers can ascertain how often different services are used and make an estimation of the approximate time to complete a wizard. Data is collected by providing surveys to the case study members and analyzing the time it takes for members to build simple applications.

As developers migrate to use R-IDE, feedback improves and expands the capabilities of R-IDE. New services and synergies that benefit the user can be identified that improve upon the development process.

4. R-IDE Product Prototype Description

The purpose of R-IDE is to enable developers to quickly and efficiently build ROS programs. The reason for the creation of this prototype is to build a working solution for the Intelligent Ground Vehicle Competition (IGVC) team to use to design and test their programs at the competition. R-IDE is designed to be a fully functional VSCode extension at the end of its prototype phase. The working prototype has proof-of-concept functionalities that range from partial implementation for some visualization tools to a fully working final version for file-handling tools. R-IDE is able to: assist in creating and updating new ROS nodes, services, and messages; record, play, and edit ROS bags; and manage the files for compilation. The development team for R-IDE receives feedback directly from Dr. Belfore's team at weekly meetings and also prompts users for feedback through the extension.

4.1. Prototype Architecture (Hardware/Software)

The prototype architecture is identical to the real-world product architecture. R-IDE tracks how users interact with the different features so as to improve and track the capabilities that are most relevant and can be best improved. TypeScript is the primary language to build the User Interface inside the Svelte frontend framework, while much of the templates are made up of C++ and Python since ROS is primarily written in these languages. The major functional components are the same between the prototype and the working product, therefore Figure 1 illustrates the major components of R-IDE and the prototype.

Because R-IDE is hosted on the VSCode Extension Marketplace, the user must utilize VSCode as an IDE. The user is required to have a version of ROS relevant to their project along with their own relevant languages for their project. Depending on the version of ROS, the user may be required to have ROS running in a Linux environment (ROS 1) or they may be able to also run inside a Windows or Mac environment (ROS 2).

R-IDE inherits its hardware requirements from both VSCode and ROS. RViz and other visualization features are embedded inside the extension, but the requirements for these tools can be dependent on the project the user is working on. The remote hosting capabilities of VSCode allow users to write code on a personal computer while working a remote computer that may better suit the needs of their project, potentially the computer onboard the robot itself.

4.2. Prototype Features and Capabilities

The features of R-IDE are retained in the prototype. However, some visualization tools have only partial implementation as shown in Table 1. This includes the full functionality of the wizards, auto-update, snippets, ROS bags, ROS topics, and the data management tool. The prototype prioritizes the tools that grant the greatest boost in both speed and ease of development.

Table 1.

R-IDE Real World Product vs Prototype Features

	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

While almost all features have functionality in the prototype, R-IDE only has partial functionality for some visualization tools. This is because can be particularly expensive to implement inside VSCode and there are tools that exist externally to VSCode. Removing functionality from this feature allows R-IDE developers to focus on more innovative tools.

4.3. Prototype Development Challenges

R-IDE is a complex extension that has faced many challenges during its development. The primary categories relate to the way in which R-IDE interacts with files and editors. It does so by: 1. allowing compatibility between different versions of ROS and the languages required, 2. easing the learning curve and leaning less on outdated documentation, and 3. embedding the visualization tools into VSCode.

The earliest challenge to appear in development is tracking and making changes in files –especially as they are edited live. To mitigate package issues, VSCode offers listener capabilities within its API, allowing extensions to monitor files for changes. Afterward, R-IDE can take advantage of regular expressions to identify where in package files to update in order to match.

Maintaining compatibility between different versions of ROS and the different languages they require is a tedious, but not insurmountable task. The best method is to make parallel changes incrementally –never letting one version or language get too far ahead.

Another potential challenge is mitigating the outdated tutorials and documentation. A simplified UI, along with clear, consistent, and living documentation will improve the user's understanding and knowledgebase of ROS and R-IDE. In complicated or difficult situations, R-IDE can give the user the ability to override automation features.

Visualization tools are a potentially high-powered but costly feature to add to R-IDE. The current prototype only seeks to implement these features partially but the features can be disabled if they are too costly or ineffective on the user end. Many of these tools already exist externally as a fallback for the user.

5. 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. It tracks 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 and 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 a 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: a 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 a 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): a set of software libraries that help 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 teaches 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 a 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.

6. 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/