LAB 1 – RIDE EXTENSION PRODUCT DESCRIPTION

Joshua Peterson

Old Dominion University

CS411W: Professional Workforce Development

Prof. Janet Brunelle

March 13, 2023

Final Version

Table of Contents

List of Figures

List of Tables

1 Introduction

Robots and autonomous machines are becoming increasingly more common, appearing not only in factories and manufacturing facilities, but also in homes, hospitals, and public spaces. They are adopting many routine tasks–whether it be for convenience, cost effectiveness, or because the task is too dangerous for humans to perform. Robots are utilized across various industries– including but not limited to: military, health care, agriculture, and autonomous vehicles. Many of the robotic markets are expected to grow at an impressive rate over the next few years. For example, the mobile robot market is expected to grow at a rate of 24% per year as the market value was estimated at $19 billion in 2018, and is expected to reach $54 billion by 2023 (Cardona et al., 2020). The industrial robotics market is also projected to increase sharply–roughly 15% by the year 2024 (Guerry et al., 2021).

To ensure the success of a robotics project, it is crucial to have reliable software that provides clear and actionable instructions for the robot to follow. However, developing software for robots without the support of specialized tools and frameworks can be a challenging and time-consuming task. This is where Robot Operating System (ROS) comes in, offering a comprehensive and accessible platform for building and deploying robotics software. ROS is an open-source set of libraries and tools that help developers build robot applications. ROS is currently being utilized by various companies such as the National Aeronautics and Space Administration (NASA), Sony, and Microsoft (Wessling, 2022; Ackerman, 2021; Fujita, 2018). ROS has an estimated market value of $270 million and is projected to reach $380 million by 2028 (Global Robot Operating System Market Research Report, 2022).

Although ROS offers numerous libraries and tools, mastering ROS development can be a challenging and time-consuming process. The default ROS system layout contains an unnecessary amount of terminal windows for even a small portion of a project. Each terminal window is associated with a node which is running concurrently with other nodes in the system. A node is a process that performs computation. Nodes are typically combined and communicate with one another to perform a desired task. The current ROS environment does little to notify the developer if changes or errors in one node are affecting the others' operation. Because of this, development and debugging can become cumbersome and meticulous. Moreover, the combination of these challenges with an unusually steep learning curve can leave new ROS developers feeling lost and overwhelmed. When developers turn to ROS's wiki, much of the information is hard to find and is spread out across various outdated documents. To put it simply, ROS development contains high barriers of entry for new developers and environments.

RIDE is an extension within Visual Studio Code for simplifying and speeding up the development lifecycle and learning process for ROS developers. RIDE utilizes Visual Studio Code's graphical user interface (GUI) and inherited functionality for general organization and other useful software development features. RIDE's user interface streamlines the creation of common ROS components, enabling developers to rapidly develop ROS applications and reducing the initial overhead for new ROS developers.

2 RIDE Product Description

The RIDE extension provides features to the current ROS environment to facilitate rapid and accurate development. The extension also provides tools to speed up the learning process for new developers. The RIDE extension combined with features from Visual Studio code, the

Microsoft ROS extension, Rviz, and Rqt creates a dynamic development environment for creation and management of ROS applications.

2.1 Key Product Features and Capabilities

Visual Studio Code offers a range of powerful features, including easy-to-use multiplexed terminals, Git support for source control, general debugging capabilities, IntelliSense for general code completion, and a massive extension marketplace. RIDE guides new developers to create common ROS components with the help of custom wizards and templates. These wizards gather information from the user through a menu and automatically generate code from pre-built templates. This saves developers the time and effort required to search for documentation or tutorials, which may be outdated. RIDE also utilizes Visual Studio Code's snippets to analyze code in real-time and suggest code completion based on the pattern that was analyzed. Simultaneously, RIDE uses IntelliSense to analyze code in order to highlight errors or missing pieces as well as to present an easy way to automatically fix the identified problems.

RIDE contains a view with an embedded version of Rviz. Rviz is a free, open-source 3D visualization environment that allows developers to view what robots are seeing and doing in real-time or with recorded data. Developing and debugging can be difficult without being able to visually see what the robot perceives is going on in the area around it. Rviz allows the developer to see through the robots eyes–whether they are cameras or laser scanners. In addition, RIDE integrates some of Rqt's core functionality. Rqt is a software framework that implements useful tools for ROS development. RIDE leverages Rqt's node graph to display the nodal network graphically, and also utilizes Rqt's ROS topic monitor and publisher to monitor and/or send messages between nodes via ROS topics, which are the named buses that the nodes communicate through.

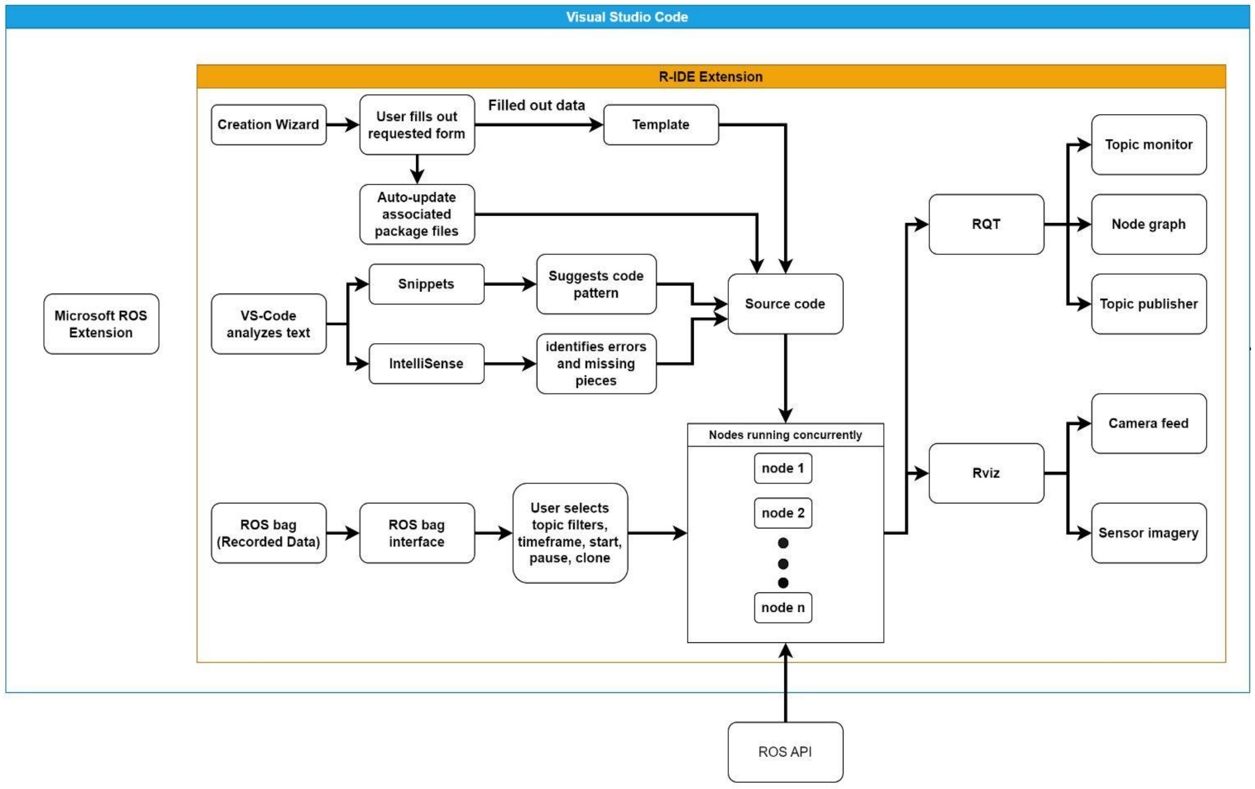
In addition to Rviz and Rqt integration, RIDE provides a user-friendly interface for managing and manipulating ROS bags, which contain recorded data from robotic inputs such as sensors and cameras. This feature can be particularly helpful during development and debugging as it allows users to play back previous data. The ROS bag interface enables users to start and stop recordings, clone ROS bags–with or without topic filters, and even trim them to a specified time frame, providing developers with greater control over their data.

2.2. Major Components (Hardware/Software)

As shown in Figure 1, RIDE exists within Visual Studio Code alongside various other ROS tools and plugins. Because ROS 1 and ROS 2 API's are significantly different, RIDE was created with modularity in mind and is available as an extension pack to support both ROS 1 and ROS 2 separately. The RIDE extension pack also contains the Microsoft ROS extension to provide additional features for ROS developers.

Figure 1

RIDE Major Functional Component Diagram



The flow of development begins after launching Visual Studio Code and opening the main RIDE extension GUI. From RIDE's main GUI, the user can start coding quickly by opening the Creation Wizard. The Creation Wizard presents a form, gathering information about the component the user wishes to create. After the user fills out the requested data, RIDE automatically generates the desired file with code from RIDE's custom templates. If the user decides to code beyond the template, snippets and IntelliSense provide assistance in the form of making auto-complete suggestions and identifying errors or missing pieces. When the user executes their source code, they are able to test and debug their new ROS application with recorded data from ROS bags. The ROS bag interface allows the user to filter topics, clone ROS bags, record ROS bags, and to play recorded data from ROS bags. The data from the ROS bag travels through the nodal network while the user's defined calculations or tasks are being performed. The output data can be monitored in multiple ways. The user can see data in the form of camera feeds and sensor imagery through Rviz, as well as Rqt's node graph to get a graphical representation of the network of nodes. The user can also view the list of current ROS topics in the topic monitor and publish new topics with the topic publisher.

3 Identification of Case Study

RIDE's primary intended users are Dr. Belfore and his students. Dr. Belfore is a professor in the Electrical and Computer Engineering Department at Old Dominion University and is overseeing numerous autonomous vehicle projects. He and his students utilize ROS in the development of the autonomous vehicles software. Dr. Belfore has the challenging task of teaching ROS application development to his students with a limited window of time. RIDE's purpose is not only to organize the current development environment, but it is also to help the students overcome the steep learning curve of ROS application development and enable productivity as quickly as possible. Although the primary users for RIDE are Dr. Belfore and his students, the RIDE extension is generic enough to be utilized by any ROS developer–including professionals who work in the robotics industry, hobbyists, along with faculty and students from other universities that utilize ROS.

4 Product Prototype Description

The purpose of the RIDE prototype is to test and prove the feasibility of a full-scale implementation of the RIDE Extension. The RIDE prototype includes everything described in the real-world product (see Figure 1) with the exception of a completely embedded version of

Rviz. Rviz is a large application that requires heavy processing and memory. So rather than embedding Rviz completely, the RIDE prototype only includes a couple of visualization features from Rviz–such as input camera feeds and sensor imagery to aid in development and debugging. The RIDE prototype is not only a proof of concept but will also be a completed extension for Visual Studio Code. The RIDE prototype proves that it is possible to simplify and speed up the development lifecycle for ROS projects by providing the user with a simplified GUI and tools that can create or manipulate ROS components. The RIDE prototype receives weekly feedback through dialogue with Dr. Belfore and his students. Visual Studio Code also provides a forum for ratings, reviews, and suggestions for individual extensions through the extension marketplace. This forum is closely monitored by the RIDE development team for any additional feedback.

4.1. Prototype Architecture (Hardware/Software)

The RIDE prototype hardware requirements consist of inherited hardware requirements for Visual Studio Code, ROS, and Rviz. To begin developing ROS projects with RIDE, a user will need the following software dependencies:

1. ROS 1 or ROS 2 along with their inherited software dependencies
2. Visual Studio Code
3. Microsoft ROS Extension

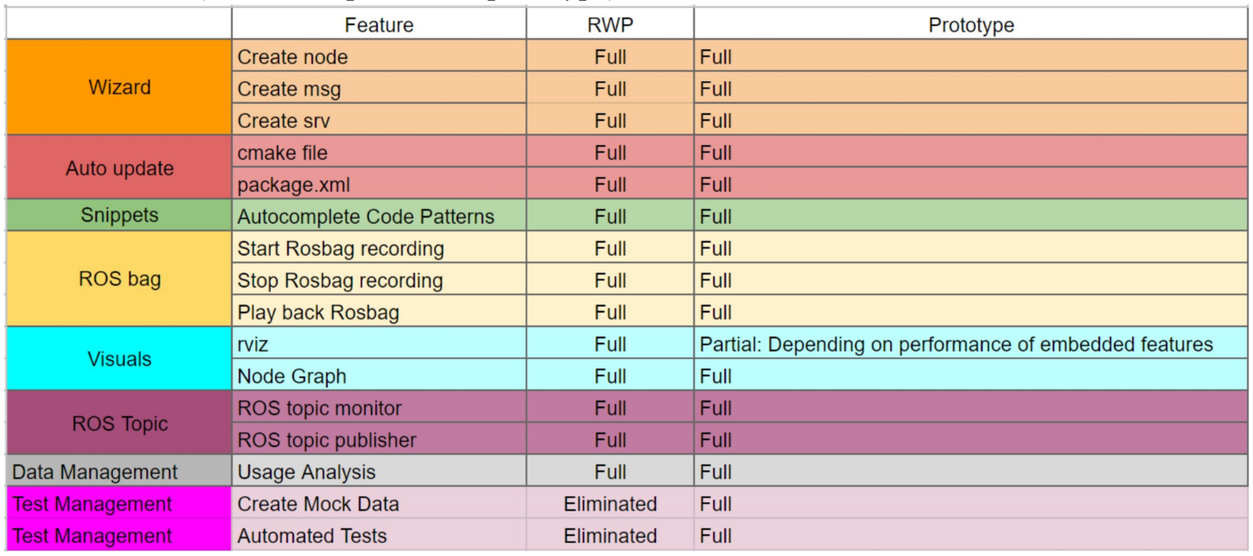
The RIDE prototype major functional component diagram (MFCD) is identical to the real world product's MFCD . As shown in Figure 1, RIDE exists within Visual Studio Code alongside the Microsoft ROS Extension. From within RIDE, users will have access to ROS's application programming interface (API) and be able to begin development. The RIDE prototype will also be connected to a MongoDB database in order to store RIDE's usage analytics.

4.2. Prototype Features and Capabilities

Table 1 lists the numerous features offered by RIDE that aid in ROS development. The RIDE prototype contains a creation wizard that collects information from the user to automatically generate common ROS components such as ROS nodes, ROS messages, and ROS services. As the components are generated by the wizard, necessary changes are automatically applied to important configuration files within the project (cmake and package.xml). RIDE utilizes Snippets to analyze code and suggest general code completion and autocompletion for common ROS components. The RIDE prototype also consists of a ROS bag GUI that allows the user to record, playback, and manipulate ROS bags. Users are able to clone ROS bags with or without filtering topics, and also to have the ability to trim ROS bags if a user only wants a portion of the recorded data. The RIDE prototype has a visualization GUI that contains Rviz for camera and sensor imagery from the various inputs, and it also has a node graph for a graphical representation of the nodal network.The RIDE prototype also includes a GUI for monitoring current ROS topics and publishing to ROS topics. RIDE utilizes usage analytics to monitor and log usage for RIDE's wizards and templates. The logs produced by RIDE's usage analytics are used to recognize the features that are used the most or the least, and to identify any errors produced while developing with RIDE. The RIDE prototype utilizes mock data for RIDE's usage analytics in order to ensure the database is being populated accurately. The RIDE prototype also contains automated tests to speed up and automate regression testing while the prototype is being updated and maintained.

Table 1

Features Table (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

Development for RIDE faces several challenges. The biggest challenge is RIDE's development team overcoming ROS's steep learning curve and becoming proficient enough with ROS to create a tool that simplifies ROS development. Along with this learning curve, ROS documentation is difficult to locate, making development even more challenging. There is also a slight disconnect working across various departments at ODU. There are many opportunities for miscommunication and misunderstanding. It is absolutely critical that there is constant communication and open dialogue between the RIDE development team and the Electrical and Computer Engineering Department (ECE) faculty and students. Another obstacle is RIDE's compatibility with both ROS 1 and ROS 2. RIDE needs to be modular and dynamic to ensure seamless compatibility. RIDE's development team also anticipates difficulty when embedding camera feeds and sensor imagery from Rviz.

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: 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.

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/