LAB 1 – RIDE EXTENSION PRODUCT DESCRIPTION

Joshua Peterson

Old Dominion University

CS410, Fall 2022

Professor J. Brunelle

January 30, 2022

Version 1

Table of Contents

List of Figures

1 Introduction

Robots and autonomous machines are becoming increasingly more common. 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 roughly 15% by the year 2024 (Guerry et al., 2021).

Successful robotics projects rely heavily on software to provide executable instructions for the robot to perform its desired tasks. Building software for robots without the help of specialized tools and frameworks can be cumbersome. That's where Robot Operating System (ROS) comes in. 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 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 provides many useful libraries and tools, becoming a proficient ROS developer can be extremely difficult. 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 together and communicate with one another to perform their desired task. The current ROS environment does little to notify the developer if changes or errors in one node are affecting another. From this, development and debugging can become cumbersome and meticulous. This combined with an unusually steep learning curve leaves new ROS developers lost and overwhelmed. When developers turn to ROS's wiki, much of the information is hard to find and 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 to simplify and speed 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 allows developers to create common ROS components at the click of a button resulting in rapid code development for 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 help facilitate rapid and accurate development along with tools to help speed up the learning process for new developers. This combined with features from Visual Studio code, the Microsoft ROS extension, Rviz, and Rqt creates a dynamic development environment for ROS applications.

2.1 Key Product Features and Capabilities

Visual Studio Code provides easy to use multiplexed terminals, Git support for source control, general debugging capabilities, IntelliSense for general code completion, and a massive extension marketplace. RIDE helps new developers 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 new developers the time and effort it takes to track down documentation or outdated tutorials. 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 to highlight errors or missing pieces and present a button 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 allowing 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 thinks is going on in the world 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 uses Rqt's node graph to provide a graphical representation of the nodal network, along with Rqt's ROS topic monitor and publisher. ROS topics are the named buses over which nodes communicate with each other.

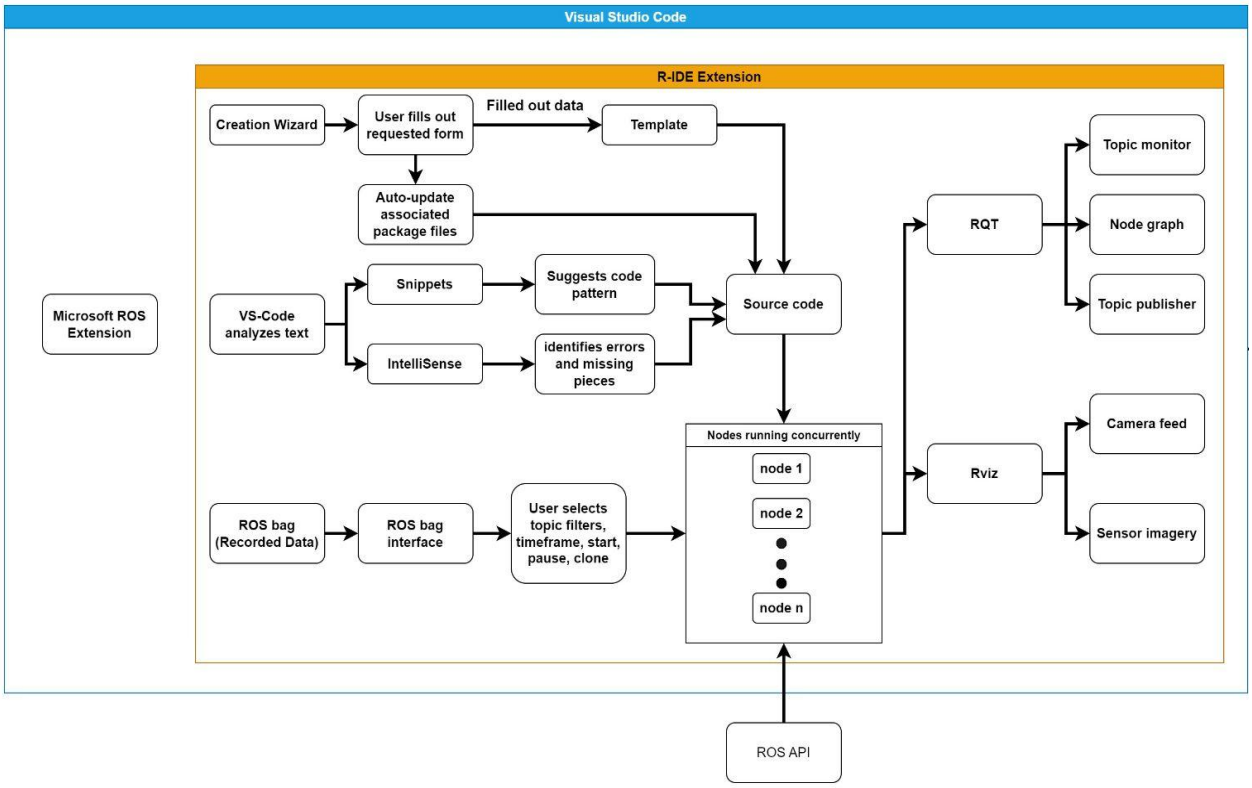
RIDE also provides an interface for developers to easily manage and manipulate ROS bags. ROS bags contain recorded data from various robotic inputs such as sensors and cameras.

ROS bags can be used to play back previous data to aid in the development and debugging process. The ROS bag interface allows users to start and stop recordings, as well as provides a menu to clone ROS bags with or without topic filters.

2.2. Major Components (Hardware/Software)

Figure 1

RIDE Major Functional Component Diagram



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.

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 wizard walks the user through 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 provides assistance in the form of suggested auto-completion 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 bags, record bags, and play recorded data from bags. The data from the 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 and is overseeing various autonomous vehicle projects at Old Dominion University. 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 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, and other universities that utilize ROS.

4 Product Prototype Description

The RIDE prototype includes everything described in the real world product 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 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's possible to simplify and speed up the development lifecycle for ROS projects by providing the user with a simplified GUI and tools to create or manipulate ROS components. The RIDE prototype is receiving 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 any 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; ROS 1 on a Linux environment or

ROS 2 on Linux, Windows, or Mac; and Visual Studio Code. 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 Visual Studio Code, 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 MySQL database to store RIDE's usage analytics.

4.2. Prototype Features and Capabilities

Figure 2

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

As shown in Figure 2, RIDE has several features to assist 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 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, as well as 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 what features are used the most or barely used, and to identify any errors produced while developing with RIDE. The RIDE prototype utilizes mock data for RIDE's usage analytics 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 updated and maintained.

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 in ROS development to create a tool for ROS. 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 while embedding camera feeds and sensor imagery from Rviz.

5 Glossary

Robot Operating System (ROS): ROS is a 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.

ROS Node: A node is a process that performs computation. 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 Bag: A bag is a file format in ROS for storing ROS message data. These bags have an important role in ROS, and a variety of tools have been written to allow you 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 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 it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters. It is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify it if necessary.

ROS Messages: Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays. Nodes can also exchange a request and response message as part of a ROS service call.

ROS Services: 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 Topics: Topics are 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. Nodes that need to perform remote procedure calls, i.e. receive a response to a request, should use services instead. There is also the Parameter Server for maintaining small amounts of state.

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

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

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

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.

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

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

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

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

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.

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/