Lab 2 - RIDE Product Specification

Joshua Peterson

Old Dominion University

CS411W: Professional Workforce Development

Prof. Janet Brunelle

April 11, 2023

Version 2

Table of Contents

List of Figures

List of Tables

****Note: Lab 2 Section 3 is delivered as a separate document from Lab 2 Sections 1&2. Section 3 is a team-prepared/submitted document, Sections 1&2 is an individually-prepared /submitted document.**

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 crucial 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. Robot Operating System (ROS) establishes the code necessary to coordinate robot actions such as movement, perception, and manipulation. It provides a robust framework for developing and integrating software modules, tools, and algorithms, enabling developers to create complex robotic systems with ease. With ROS, developers can prototype, test, and deploy robotics software, making it a vital tool in the field of robotics. ROS is currently being utilized by a wide range of major 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. While ROS nodes can be designed to run autonomously, they often need to communicate with each other to achieve complex tasks. The issue with the current ROS environment is that it does not provide enough information to developers about how changes or errors in one node can impact the operation of the others, which can make it difficult to troubleshoot and debug complex robotic systems. 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 official documentation, much of the information is hard to find and is spread out across outdated web pages and documents. To put it simply, ROS development contains high barriers of entry for new developers and environments.

RIDE is specifically designed to work as a Visual Studio Code extension, taking advantage of the stable UI environment already in place. By utilizing Visual Studio Code's graphical user interface and inherited functionality, RIDE provides a seamless and intuitive experience for ROS developers, simplifying and speeding up the development lifecycle and learning process. 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.

1.1 Purpose

The goal of the RIDE extension is to simplify the ROS development lifecycle and increase productivity for ROS developers. The RIDE extension is intended to equip developers with tools that not only facilitate the creation of ROS applications but also to help throughout the entire development lifecycle including maintaining, debugging, and updating complex applications. By providing a comprehensive development environment that integrates seamlessly with Visual Studio Code's graphical user interface, RIDE empowers developers to focus on creating high-quality software while minimizing the overhead associated with development tasks. Whether creating common ROS components, managing ROS bags, or visualizing complex system behavior, RIDE's intuitive user interface and advanced functionality enables developers to maximize their productivity.

1.2 Scope

The RIDE prototype is intended to function as a real-world product and provide a reliable and efficient development environment for Dr. Belfore, a professor in the electrical and computer engineering department and the head of the autonomous vehicles research group at Old Dominion University, and his students. The RIDE extension features a creation wizard that streamlines and automates the creation of common ROS components. RIDE also automates package management and makes updates to configuration files as needed. Additionally, the RIDE extension will utilize Visual Studio Code’s extension API to offer a graphical user interface for ROS bag management, and visualization tools for camera and sensor imagery to aid in debugging and development. The RIDE extension will also include snippets that enable rapid code generation of common ROS components.

The RIDE team has created a risk-mitigated environment by integrating RIDE as an extension within VSCode. Moreover, the RIDE team is in constant communication with the customer, Dr. Belfore and his students, to ensure that the project requirements are met within the expected timelines and to the satisfaction of all stakeholders.

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

Peterson, J. (2023). LAB 1 – RIDE EXTENSION PRODUCT DESCRIPTION [Unpublished manuscript]. Old Dominion University.

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/

1.5 Overview

This product specification provides a thorough overview of RIDE's architecture, detailed software and hardware requirements, and a comprehensive list of features that the extension offers. The subsequent sections of this document provide the major functional components that are incorporated into the RIDE extension, as well as detailed breakdown of the specific hardware and software requirements on a separate document.

2. General Description

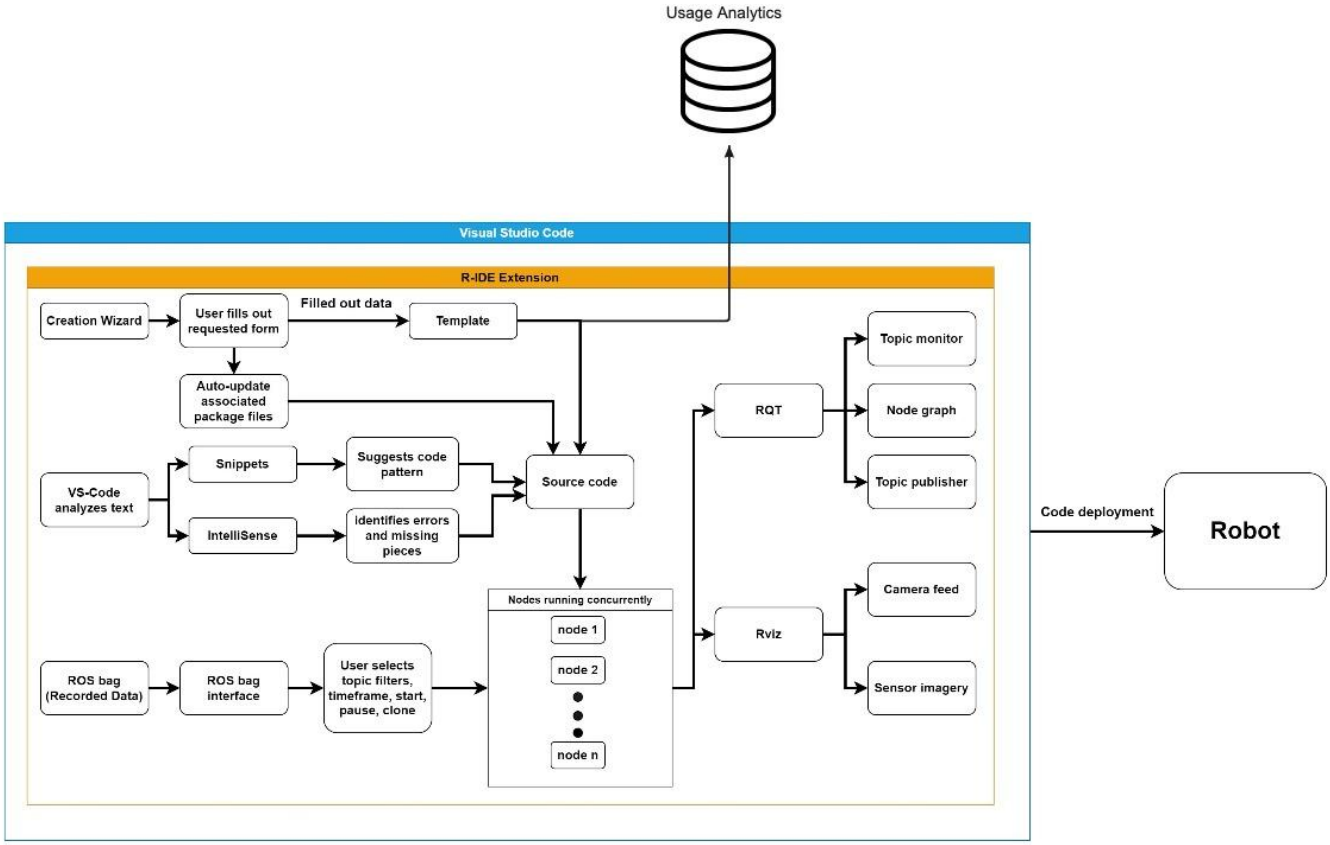
RIDE is a full extension for VSCode. The RIDE extension is a proof of concept and provides features to the current ROS environment to facilitate rapid and accurate development of robotics applications. The extension will also provide tools to speed up the learning process for new developers. The RIDE extension, in conjunction with features from Visual Studio Code, the Microsoft ROS extension, Rviz, and Rqt, will create a dynamic development environment that enables development, editing, updating, and debugging ROS applications.

2.1 Prototype Architecture Description

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 developing a ROS application with RIDE 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 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 to existing topics with the message publisher.

2.2 Prototype Functional Description

Table 1 lists the 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

2.3 External Interfaces

The RIDE extension shall utilize specific hardware, software, and user interfaces for developers to interact with. The system will utilize and depend on popular software development tools such as VSCode and ROS 1/ROS 2.

2.3.1 Hardware Interfaces

The RIDE extension requires hardware such as a desktop or laptop that meets the inherited requirements of ROS and VSCode.

2.3.2 Software Interfaces

Development of the RIDE extension requires the use of several software API's. To ensure seamless integration with VSCode, the RIDE extension utilizes VSCode's Extension API, which enables the extension to interact with the VSCode editor and provide developers with a familiar development environment. RIDE also utilizes mongoDB via mongoose to store usage analytics. ROS is a popular robotics framework that RIDE utilizes through ROSLib, a JavaScript library for ROS. RIDE leverages ROS bridge to act as a proxy between ROSLIB and ROS, enabling seamless integration between ROS1 and ROS2. In addition, RIDE leverages Rvis to enable developers to visualize camera and sensor imagery.

2.3.3 User Interfaces

RIDE aims to deliver a user-friendly and intuitive development environment for robotics applications. To achieve this, RIDE leverages the capabilities of Svelte, a lightweight yet robust front-end framework for designing and implementing graphical user interfaces (GUIs) within VSCode's webviews.

2.3.4 Communications Protocols and Interfaces

RIDE communicates with the database using MongoDB, which provides a RESTful API for data insertion and retrieval.

3. Specific Requirements

Separate Document