

Lab 1 - Outline
CS 410 - Fall 2022
R-IDE - Team Orange

Joshua Peterson, Dominik Soós,
Dan Koontz, Gavin St. Clair,
Justin Tymkin, James Hart

Table of Contents

1. Introduction
2. Product Description
 - 2.1. Key product features
 - 2.2. Major components
3. Identification of Case Study
 - 3.1. Who is the intended user?
 - 3.2. What is the intended use of R-IDE
4. Product Prototype Description
 - 4.1. Prototype Architecture (Hardware/Software)
 - 4.2. Prototype Features and Capabilities
 - 4.3. Prototype Development Challenges
5. Glossary
6. References

1. Introduction

ROS development contains high barriers of entry for new developers and environments.

- ROS workflows leverage several non-attributional windows that elevate the difficulty of debugging, monitoring, and understanding.
 - Each ROS node has at least one terminal window associated with it
 - There is no distinction when changes in one node affect a related node
 - Simple debugging methodologies become time-consuming due to how ROS displays errors
- Many parts of the documentation aren't up to date and require significant technical knowledge
- Environment setup can be difficult in any project, but especially in one that can use multiple versions and distributed dependencies

An application to simplify and speed up the development lifecycle and learning process.

- Create a GUI such that it:
 - Simplifies common tasks and commands
 - Gives the user access to visualization tools
 - Organizes the developer's interface to facilitate more efficient development
- Quick environment setup:
 - Create a process to quickly build an environment for any ROS project
 - Provide templates for various tasks such as creating nodes, or building launch files

2. R-IDE Product Description

R-IDE will provide an environment to simplify and speed up the development lifecycle and learning process. Part of simplification involves making it easier to debug within ROS, whether that involves somehow injecting a debugger like GDB into ROS or finding a way to tell ROS to pause execution after a given action or publish. This would allow a developer to test a system in a simulated environment in realtime and identify issues.

2.1. Key Product Features and Capabilities

While studying the problem and interviewing the customer base, some major characteristics have been identified that should make up the solution.

- Simplify User Interface

- The user interface will simplify the most common tasks and actions to the point of a button click or menu
- By simplifying the UI, we reduce reliance on outdated tutorials. Developers can create basic nodes and functions that work right away rather than having to delve into documentation that can be 10 years old.
- Quick Environment Setup
 - Create a process to build an environment for ROS projects

R-IDE Features:

- ❖ Graphical User Interface
 - Manipulate, save and share GUI's perspective
- ❖ git
 - Push files to project in source control
- ❖ Open, edit and save files.
- ❖ Debugging – bug tracking features
 - Error detection
 - Ability to save error log
- ❖ Multiple embedded terminal windows
- ❖ Observe and manipulate ROS Topics
- ❖ ROS bag

2.2. Major Components (Hardware/Software)

- ❖ The major functional component diagram for R1D3 consists of a developer environment and a production environment.
- ❖ Inside the local code branch, the ROS developers work with the ROS development environment user interface. This UI integrates the network of ROS nodes from node 1 to node N. It also supports code editing and the ROS API. The developers make use of the UI's features to locally test the code inside or outside a container of the ROS environment.
- ❖ This code is manually transferred to the test server and to the vehicle inside the production environment through the code merge pipeline. The vehicle currently has sensors, actuators, and an onboard vehicle database that send data to an external database. The data in this external database is collected and sent to the test server where the developers are able to debug errors through the code merge pipeline.

3. Identification of Case Study

3.1 Who is the intended user?

- Dr. Belfore and his students
- Other universities
- Robotics hobbyists
- Robotics professionals

3.2 What is the intended use of R-IDE?

- To provide an intuitive yet powerful development environment for ROS based applications
- To provide tools that reduce initial overhead and enable new ROS developers

4. Product Prototype Description

4.1. Prototype Architecture (Hardware/Software)

4.2. Prototype Features and Capabilities

4.3. Prototype Development Challenges

5. Glossary

<https://jpete020.github.io/team-orange/#/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.

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/>