



A Framework for Quality Assessment of ROS Repositories

2016 IEEE/RSJ International Conference on
Intelligent Robots and Systems

André Santos

Alcino Cunha Nuno Macedo Cláudio Lourenço

INESC TEC & Universidade do Minho, Portugal

October 13th, 2016



Summary

- › Many interesting robotics applications (e.g. health, industry) require high levels of **safety** and **flexibility**.
- › In such scenarios, safety is usually ensured by **software**.
- › This kind of software must be **high-quality software**.
- › Our goal was to get a **panorama of the current quality level** in many popular ROS robots.
- › To achieve this goal, we built a **framework** to automate the collection of several **quality measures**.

Software Quality

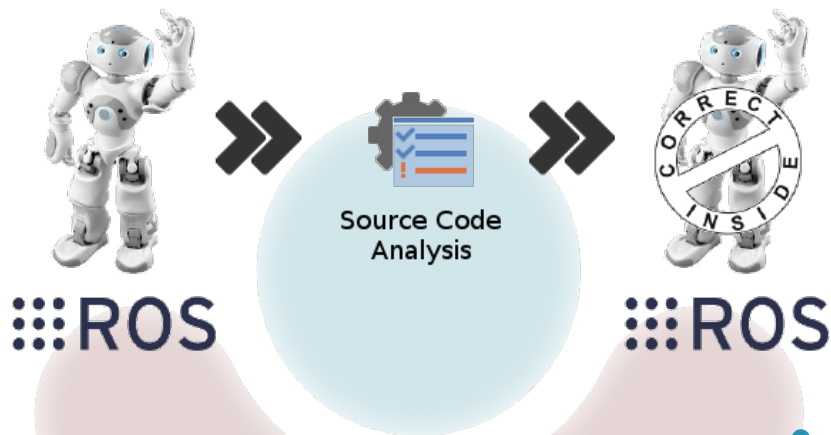
One way to minimise safety issues is to produce **high-quality software**.

To improve safety and quality, adopting **coding standards** – rules and recommendations about how to write the software – is a common practice (e.g. ROS C++ Style Guide, MISRA C++, HIC++).

Another common technique is to analyse **quality metrics** – numeric values about how much a property manifests (e.g. lines of code, number of dependencies, function complexity).

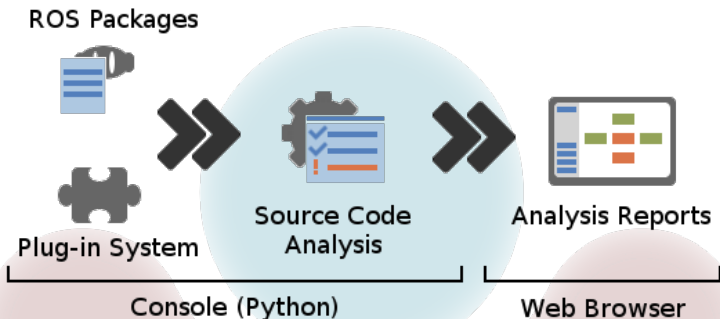
The HAROS Framework: Overview

The **HAROS Framework** (High-Assurance **ROS**) aims at providing an analysis platform for ROS systems, making robots more **reliable**.



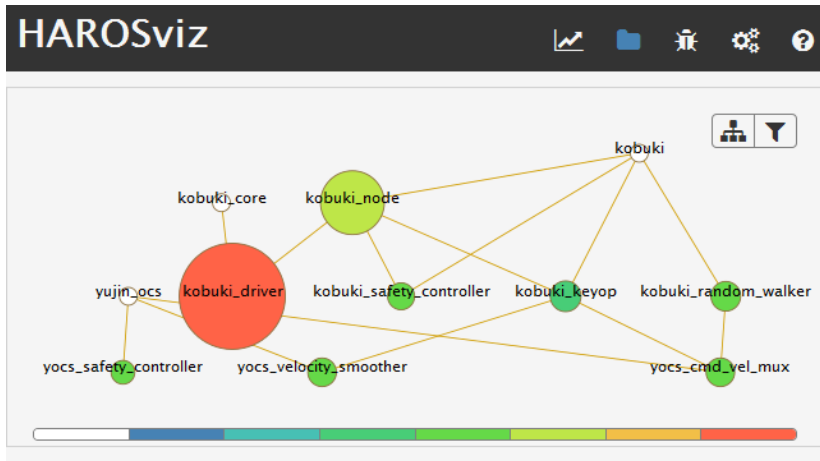
The HAROS Framework: Main Features

- › **Source code fetching** of indexed ROS packages.
- › **Plug-ins** enable integration of third-party analysis tools.
- › **Interactive graphic reports** of the results mirroring the ROS architecture.



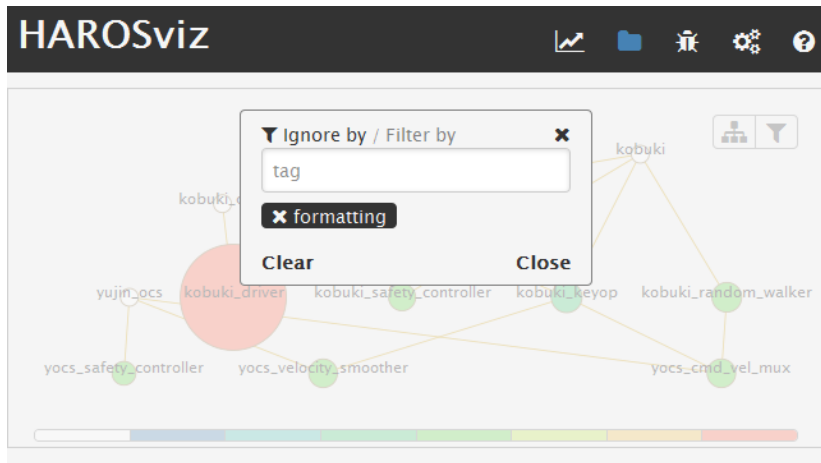
The HAROS Framework: Visualisation

The visualiser builds a diagram of the analysed packages.
Package colours denote the amount of issues.



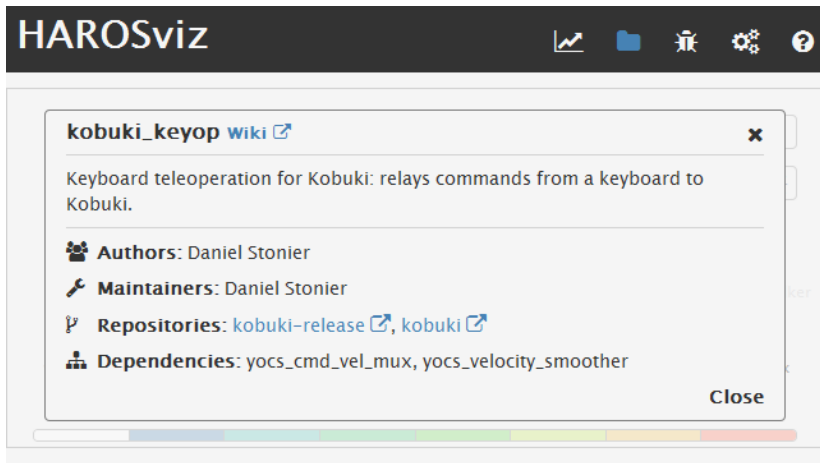
The HAROS Framework: Visualisation

Issues can be filtered or ignored by tags.



The HAROS Framework: Visualisation

Package details are also available.



The screenshot shows the HAROSviz application window. The title bar contains the text 'HAROSviz' and several icons: a line graph, a folder, a trash can, a gear, and a question mark. The main content area displays the details for the 'kobuki_keyop' package. The package name is 'kobuki_keyop' with a 'Wiki' link and a close button. Below the name is a description: 'Keyboard teleoperation for Kobuki: relays commands from a keyboard to Kobuki.' The details are organized into sections with icons: 'Authors' (people icon) listing Daniel Stonier; 'Maintainers' (wrench icon) listing Daniel Stonier; 'Repositories' (bookmarks icon) listing 'kobuki-release' and 'kobuki'; and 'Dependencies' (building icon) listing 'yocs_cmd_vel_mux' and 'yocs_velocity_smoother'. A 'Close' button is located at the bottom right of the details panel. At the bottom of the window, there is a progress bar with segments in white, blue, green, yellow, and red.

HAROSviz

kobuki_keyop [Wiki](#) ✕

Keyboard teleoperation for Kobuki: relays commands from a keyboard to Kobuki.

Authors: Daniel Stonier

Maintainers: Daniel Stonier

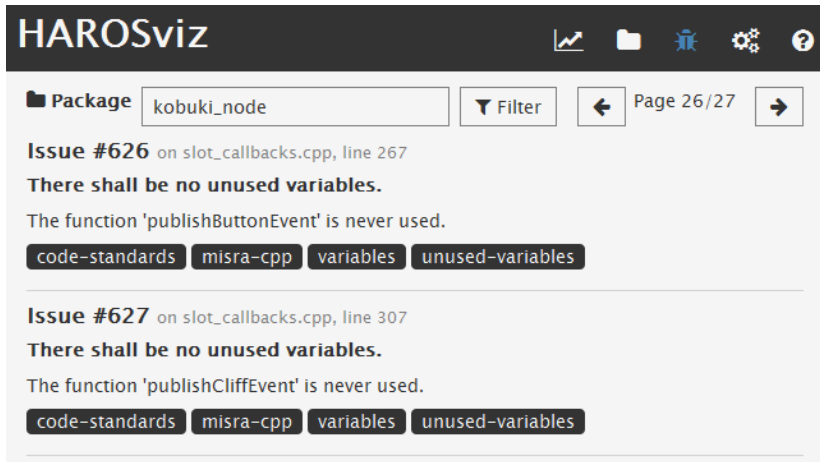
Repositories: [kobuki-release](#), [kobuki](#)

Dependencies: yocs_cmd_vel_mux, yocs_velocity_smoother

Close

The HAROS Framework: Visualisation

Issues can be **inspected** in detail.



The screenshot shows the HAROSviz web interface. At the top, there is a dark header with the title "HAROSviz" and several navigation icons (line graph, folder, bug, settings, help). Below the header, there is a navigation bar with a "Package" dropdown set to "kobuki_node", a "Filter" button, and a "Page 26/27" indicator with left and right arrows. The main content area displays two issues:

Issue #626 on `slot_callbacks.cpp`, line 267
There shall be no unused variables.
The function 'publishButtonEvent' is never used.
code-standards misra-cpp variables unused-variables

Issue #627 on `slot_callbacks.cpp`, line 307
There shall be no unused variables.
The function 'publishCliffEvent' is never used.
code-standards misra-cpp variables unused-variables

The HAROS Framework: Case Study

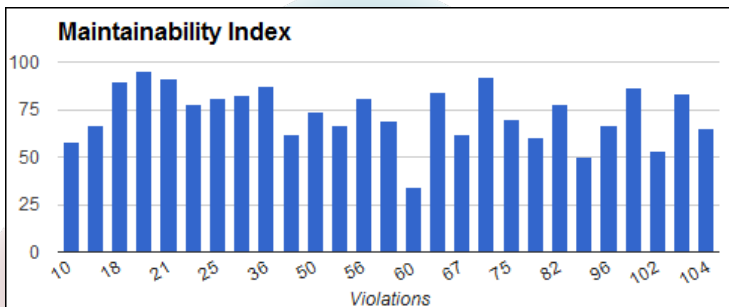
HAROS was applied on 11 ROS robots, using CCCC, Radon, Cpplint and Cppcheck as plug-ins.

- › Analysis sample: 46 repositories – more than 350 000 lines of C++.
- › Assessment of over 100 rules and 15 metrics.
 - › Covering ROS and Google's C++ Style Guide, and a small portion of MISRA C++, HIC++, and JSF AV C++.
 - › Source metrics: lines of code/comments, comment ratio, maintainability, dependencies, cyclomatic complexity, ...
 - › Process metrics (from GitHub): commits, contributors, number of issues.
- › Packages categorised as drivers, libraries, or applications.

The HAROS Framework: Case Study

Some observations:

- › The projects have **thousands of coding rule violations**.
- › There are **few correlations** between metrics – the quality is **inconsistent**.
- › Drivers and applications are **more active** – more developers and commits, but also more issues.

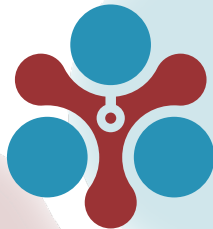


Future Work

- › Integration of stronger analysis techniques, e.g. **formal verification** and **model checking**.
- › **Model extraction** from source code.
- › **Inter-operation** between plug-ins.
- › Integration with the **catkin build system**.
- › **Continuous tracking** of package quality.

Give HAROS a try at
git-afsantos.github.io/haros

Thank you!



HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY