Spring 2018 Research Project Proposal

Laboratory for Embedded Machines and Ubiquitous Robots
University of California, Los Angeles
Quentin Truong

Introduction

Robotics is widely applicable to our everyday lives; however, despite their utility, the design and fabrication of robots is largely inaccessible to the general public due to a lack of easy-to-use design and fabrication tools. The Robot Compiler, RoCo, addresses this need through its intuitive design scheme and ability to move from design to fabrication.

Problem

As RoCo continues to grow as an open source project, the need to manage complexity, maintain functionality, and ensure extensibility will become more and more critical. Without practices in place to address these issues, RoCo's growth will eventually be prohibited by its own complexity.

Currently, RoCo's limited documentation and lack of runnable test scripts creates unnecessary complexity for developers. In particular, the limited documentation creates a steep learning-curve for new developers, hindering RoCo's capacity to operate successfully as an open-source project. Without a quick reference to the software's abstractions and implementations, developers are left without reasonable means to understand how the system actually runs. Moreover, the lack of test scripts increases the complexity of maintaining and extending RoCo. Without testing, developers are unable to ensure that newly added features do not break pre-existing ones.

Solution

There are many basic software development practices applied in nearly all projects to help manage complexity, maintain functionality, and ensure extensibility. Among these practices are documentation and unit testing. For this project, documentation will be hosted on the public GitHub Wiki so that any developer may quickly find and access necessary information. Unit testing will be built using the standard Python unittest framework. The Python unittest framework is chosen over alternatives for its ability to share setup between test cases, aggregate test cases into test suites, and maintain independence of test cases from source code.

Once the documentation and test cases for RoCo have been written, new developers will be able to quickly learn the codebase, integrate new features, and ensure the functionality of the system. The benefits of reducing RoCo's complexity will extend beyond RoCo itself; as RoCo develops, we will learn the requirements and

abstractions necessary for any easy-to-use robotics development and fabrication system.

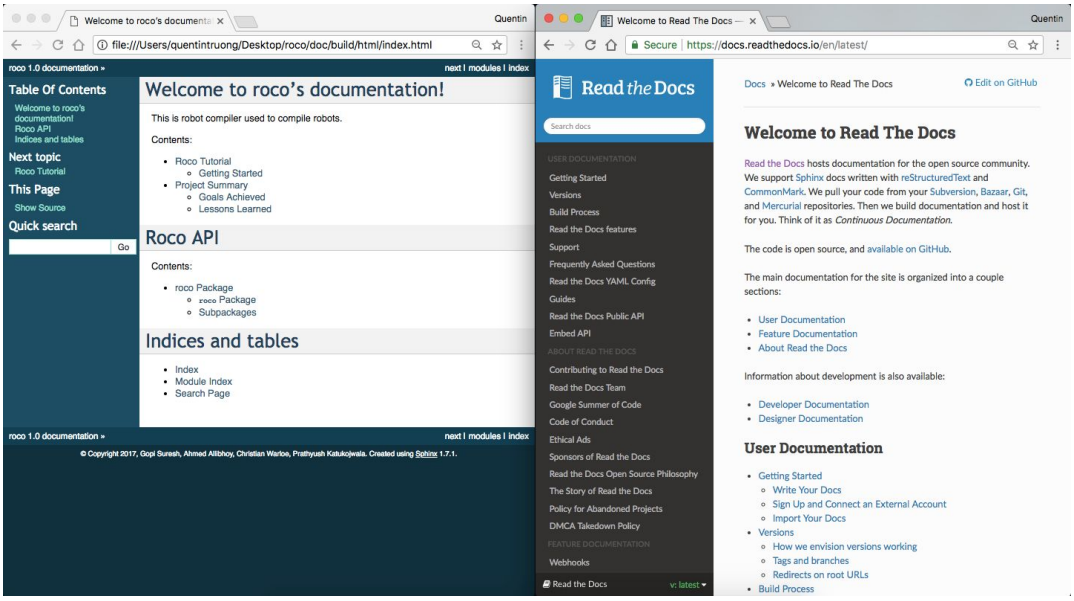

Figure 1: RoCo vs. RTD Documentation: The above panels show the current state of RoCo documentation (left) and an example of the style of documentation we will progress towards (right).

Breakdown of Goals

- Prepare RoCo for future development
    - Document abstraction model
        - Further help developers learn/understand RoCo
        - Ensure future features are built in extensible ways
    - Document classes/functionalities
        - Helps developers to learn/understand RoCo
        - Serve as reference for developers
    - Write unit tests
        - Ensure functionality of RoCo

Breakdown of Tasks

- Create visual overview of abstraction model
    - Logical boundaries of each phase
    - Phases (design vs fabrication)
    - Data flow
- Create visual overview of class relationships
    - Inheritance relationships

- Composition relationships

For each level,

For each class,

- Document classes
  - Purpose
  - Explain key terms/link to them
  - How it affects data
  - Relationship to other classes
- Document functions
  - Purpose
  - Parameters
  - Return values
- Unit test functions
- Track bugs as necessary

Extra Tasks

- GitHub Badges, Continuous Integration Setup, Semantic Versioning, Visual Overview of Data Flow, Pylint

Tools

- reStructuredText format with Sphinx and Read the Docs Theme
  - Sphinx - https://github.com/sphinx-doc/sphinx
  - Sphinx_rtd_theme - https://github.com/rtfd/sphinx_rtd_theme
- Python unittest
- GitHub Issue Tracker
- Google Python Style Guide

Project Timeline

Week: Tasks to be completed by the start of next week

02-25: Decide on tasks, Decide on tools, Write project proposal, Establish timeline, Find relevant research papers, Setup Sphinx, Integrate RTD theme

03-04: Gather abstraction model information

03-11: Turn abstraction model information into graphic

03-18: Gather class relationships information

03-25: Turn class relationships information into graphic

04-01: Document, unit test, cleanup docstrings/imports/formatting for Parameterized and Variable

04-08: Document, unit test, cleanup docstrings/imports/formatting for Composable and Connection

04-15: Document, unit test, cleanup docstrings/imports/formatting for Interface and Port
04-22: Document, unit test, cleanup docstrings/imports/formatting for Component and ContainerComposable
04-29: Document, unit test, cleanup docstrings/imports/formatting for ElectricalComposable and VirtualComposable
05-06: Document, unit test, cleanup docstrings/imports/formatting for CodePort and EdgePort
05-13: Document, unit test, cleanup docstrings/imports/formatting for ElectricalPort and MountPort
05-20: Document, unit test, cleanup docstrings/imports/formatting for SixDOFPort and CodeComponent
05-27: Document, unit test, cleanup docstrings/imports/formatting for ElectricalComponent and MechanicalComponent
06-03: Document, unit test, cleanup docstrings/imports/formatting for NodeMCU (Android.py) and Cutout
06-10: Document, unit test, cleanup docstrings/imports/formatting for Header and NodeMCU (node_mcu.py)
06-17: Document, unit test, cleanup docstrings/imports/formatting for CodeContainer and ElectricalContainer
06-24: Document, unit test, cleanup docstrings/imports/formatting for CodeComposable
07-01: Document, unit test, cleanup docstrings/imports/formatting for additional classes
07-08: Document, unit test, cleanup docstrings/imports/formatting for additional classes
07-15: Document, unit test, cleanup docstrings/imports/formatting for additional classes
07-22: Document, unit test, cleanup docstrings/imports/formatting for additional classes
07-29: Document, unit test, cleanup docstrings/imports/formatting for additional classes
08-05: Document, unit test, cleanup docstrings/imports/formatting for additional classes
08-12: Document, unit test, cleanup docstrings/imports/formatting for additional classes
08-19: Document, unit test, cleanup docstrings/imports/formatting for additional classes