Enhancing Software Portability with a Testing and Evaluation Platform

Florian Weisshardt, Fraunhofer IPA, florian.weisshardt@ipa.fraunhofer.de, Germany
Jannik Kett, Fraunhofer IPA, jannik.kett@ipa.fraunhofer.de, Germany
Thiago de Freitas Oliveira Araujo, Fraunhofer IPA, thiago.de.freitas.oliveira.araujo@ipa.fraunhofer.de, Germany
Alexander Bubeck, Fraunhofer IPA, alexander.bubeck@ipa.fraunhofer.de, Germany
Alexander Verl, Fraunhofer IPA, alexander.verl@ipa.fraunhofer.de, Germany

Abstract

Recently a variety of service robots is available as standard platforms allowing a worldwide exchange of software for applications in the service sector or industrial environments. Open source software components enhance this sharing process, but require the maintenance of a certain level of quality. This paper presents an approach to a testing and evaluation platform which facilitates the sharing of capabilities over different robot types, environments and application cases.

1 Introduction

In the past years, there have been more and more complex and capable service robots available as standard platforms [1]. Two examples are the Care-O-bot® 3 [2] and the PR2 [3] robots, which are already in use in different organizations spread all over the world [4]. First these robots were mainly targeted at the research community, but nowadays they get closer to real application in the service sector or in industrial environments as well. An example of a robot for industrial environments is the rob@work3 platform [5].

All these robots share their capabilities due to the use of the open source Robot Operating System (ROS) [6] with the goal to minimize development efforts for a single robot by maximizing the reuse of capabilities developed for each of them. Within the ROS infrastructure it is easy to share software components and developed capabilities as shown in [7] like hardware drivers, manipulation, navigation and perception components which are organized in ROS packages.

There are more and more industrial robots being available in ROS with the goal to make use of the capabilities in the ROS framework. In order to accelerate the transfer to the industrial domain the ROS-Industrial Initiative [8] was founded by SwRI.

The development in ROS and ROS-Industrial is realized in a distributed open source community [9] by developers with different background and coding capabilities. Hence, the packages have different activity levels, release cycles and levels of maturity. From an outside point of view it is often hard to distinguish well established and widely used packages from one shot implementations for a single use case.

A requirement for sharing functionalities from one robot to another one, from one environment to another one and from one to another application, as illustrated in Figure 1, is to make sure that these functionalities will perform in a way they are supposed to. The only way to ensure this wide range of hardware and environment independent applications is to perform an extensive testing which is often executed manually and therefore time consuming and related to high costs. One way of decreasing effort and cost is to make use of mostly automated testing of software capabilities on the target robot, environment and application case.

Extensive testing and an objective QA analysis of available ROS components is one of the lessons learned from the early stages of the ROS-Industrial Initiative [8]. The authors state that it is a major issue to develop reliable industrial applications.

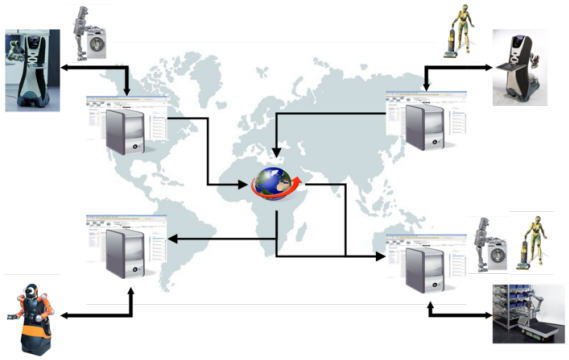


Figure 1: Sharing functionalities between different robots (e.g. Care-O-bot® 3, PR2 and rob@work3), different environments and various application cases.

In this paper, a testing platform for software capabilities is proposed which can be tested with a variety of robot types, environments and application cases in order to enhance the transfer of software capabilities. Figure 2 shows the hierarchical setup of the proposed testing platform which is embedded into the continuous integration

ISBN 978-3-8007-3601-0
© VDE VERLAG GMBH · Berlin · Offenbach

(CI) server Jenkins [10] and offers various types of tests: build, *simulation-in-the-loop*, *hardware-on-the-loop* and scenario tests.

The paper is structured as follows: After discussing related work and the approach in section 2, the testing platform is introduced in section 3. In this section, the setup of the infrastructure and the hierarchical layers of the build and test structure are shown. In section 4, the evaluation of the usage by the Care-O-bot® Research [11] community is shown. The paper closes with a conclusion and outlook in section 5.

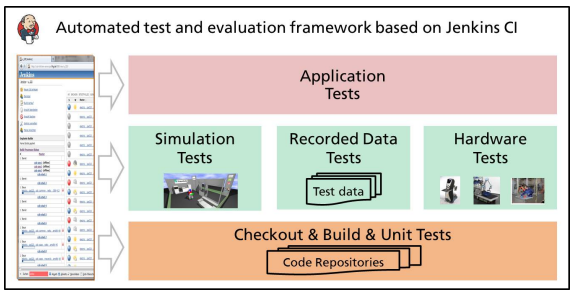


Figure 2: Overview of the testing framework including build, *simulation-in-the-loop*, *hardware-on-the-loop* and scenario tests.

2 Related work and approach

The goal of this work is to obtain a stable and portable software which is usable on several robots, in various environments and in different application cases. Neither type nor architecture nor release version of the installed operation system should influence the performance or reliability of the developed code. Moreover, it should be ensured that different repository versions are compatible to each other.

In general, software development testing and continuous integration are widely used tools. In the domain of robotics and ROS there is related work realized by Willow Garage and now continued at OSRF [12]. They focus on building ROS code for various architectures and operating systems by executing bare unit tests and building binary releases. They use the Jenkins continuous integration server [10] to automate the process.

Based on the work of [12], we propose a testing and evaluation framework which enables automated testing and provides feedback about the operational capability and behavior of software package. Using the proposed testing platform allows benchmarking tests too. This is not only achieved by executing build and unit testing jobs but by adding the capability to execute standardized tests in simulation (*simulation-in-the-loop*) and on the real hardware (*hardware-in-the-loop*) as shown in Figure 2. The results are visualized by a web-based front end in the form of a component catalog which displays the test results including reports, statistics and videos.

The testing platform is based on the following user stories:

Component developer A component developer, responsible for the object perception or the navigation functionality, wants to further develop and enhance his code. After committing the changes of the source code, a build is automatically triggered and tested for several test cases (benchmark tests), on various versions of the operating system, architectures, multiple robots and in a set of environments. Shortly afterwards, the developer receives a feedback: only 60% of the formerly defined test cases are passed. So there is a need to fix the degradation which came into the component. After fixing the code, the developer checks in further changes. The whole testing process starts again until all tests pass again.

Application developer Furthermore, an application developer needs an object perception algorithm for his application. Based on the results of the benchmarking tests being visualized in the component catalog the application developer can choose among various available implementations for an object perception component. This allows to save time while choosing the component which works best for the boundary conditions of the specific application.

3 Testing Platform

Figure 2 shows an overview of the testing platform. First, the code which is organized in software repositories is checked out on the continuous integration server followed by a build process on different architectures. After a successful build, there are various test cases which are triggered using tests in a simulation environment and from previous recorded sensor data. Verifying the correct behavior of single components a compound of multiple components is tested in a scenario setup. The results of all stages are recorded and visualized in a web interface by the continuous integration server.

The infrastructure and the hierarchical layers of the build and test structure is shown in Figure 3. Besides commonly used continuous integration and static code analysis tools the testing framework supports *simulation-in-the-loop* and *hardware-in-the-loop*. Before testing in simulation or on the real hardware there is always a so called *priority build* which does compilation and basic tests on various architectures so that only tests are executed in simulation or on the hardware if the code at least compiles and fulfills some basic requirements. All tests can easily be configured by a specific plugin for Jenkins and therefore enables various stakeholders of the development process (component developer, system integrator and application developer) [13] to use the framework.

ISBN 978-3-8007-3601-0
© VDE VERLAG GMBH · Berlin · Offenbach

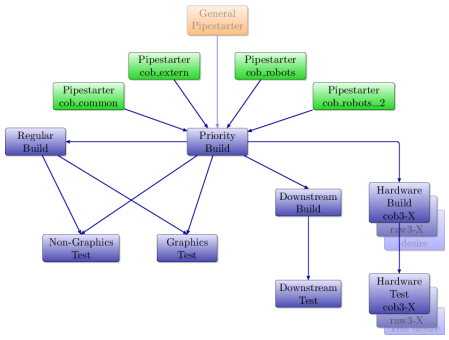


Figure 3: Hierarchy of the build tests.

Every developer can define its own version of the repository which will be tested by creating a set of jobs on Jenkins[1]. As shown in Figure 3, a set of jobs for one repository consists of:

Pipe starter jobs The pipe job is responsible for triggering a new run of the test pipeline. The test pipeline can be triggered either manually by the user or by source code changes in a version control system.

Build jobs Build tests cover basic tests by checkout, dependency checks and installation, compiling, static code analysis. Each build job is executed in a clean chroot environment for various architectures (32bit and 64bit) and OS release version where only the dependencies defined in the ROS packages will be installed. This ensures the same system configuration for all following jobs. In case of a successful build the chroot environment will be saved on the Jenkins server as a tarball, so that further jobs can execute tests on top of the build job. Results of the static code analysis are aggregated on the jobs page.

Hardware-in-the-loop testing Jobs finishing successfully building software components on the build farm directly trigger a test procedure on the robot platforms that are integrated into the continuous integration server. The robots themselves are setup as slaves for Jenkins checking out all repositories which are required for testing on the robot PCs. There are two kind of tests: On the one hand, fully automated tests that require no interaction from a human. On the other hand partly manual tests which need either input from a human observer or need to be supervised by a human because the robot makes use of its actuators and performs movements which could result in damaging the robot or the environment.
During the automated testing all hardware components are checked. Thus, all sensors, e.g. cameras and laser scanners, can be tested for being available on the robot and delivering correct data using so called hz tests [14]. Furthermore, the availability of essential system components can be verified using diagnostics information [15] about the hardware status. The partly manual tests include moving all actuators, e.g. base, arm and gripper as well as testing input and output devices, e.g. speakers, microphones, status LEDs and tactile sensors. The controllers of the actuators are tested relating to their ability to move the components in joint space as well as cartesian space and deliver status information about their joint configuration.
The automated and partly manual tests for the hardware layer are necessary to ensure the correct behavior of all hardware components before being able to perform any other high level tests on the robot hardware.

Tests based on previously recorded data Apart from directly testing on the hardware system, the test framework provides the possibility to perform tests based on previously recorded data, e.g. from real hardware sensors.

Listing 1: Configuration file for testing object detection component 2

```
tolerance: 0.1
PKG: object_detection_2
mode : default

test_bag:
  - name: testCase1
    bag_path: /test/bagfiles/testCase.bag
    yaml_path: /test/yaml/testCase.yaml
    tolerance: 0.1 # meters

  - name: testCase2
    bag_path: /test/bagfiles/testCase2.bag
    yaml_path: /test/yaml/testCase2.yaml
    tolerance: 0.2 # meters
```

Listings 1 shows the definition of tests for an object detection component where the ground truth data is given by another configuration file shown in listing 2. It contains the definition of objects type and position.

Listing 2: Configuration file defining ground truth data for object detection testing

```
objects:
    - label: milk
      tolerance: 0.04 # meters
      position : [−0.09, −0.35, 1.34] # [x,y,z]
      orientation : [0 , 0, 0] # [r,p,y]
    - label: zwieback
      tolerance: 0.05 # meters
      position : [−0.09, −0.35, 1.33] # [x,y,z]
      orientation : [0 , 0, 0] # [r,p,y]
```

Simulation-in-the-loop testing Testing in a simulation environment offer the opportunity to setup a multitude of test cases without the need of avoiding damages to the robot and the environment because simulation can easily restart, e.g. in case of collision. This allows fully automated tests without supervision by an operator. For the simulation tests the Jenkins system will automatically start the jobs on a simulation enabled host and run the simulation prior to executing the test itself.

[1] A running version of the test infrastructure can be found on http://build.care-o-bot.org

ISBN 978-3-8007-3601-0
© VDE VERLAG GMBH · Berlin · Offenbach

Testing the navigation components of a mobile robot can be used as an example for a test in simulation. The test can be defined as shown in Figure 4. The task is to navigate to all goal positions and check if it successfully arrives to the goal by avoiding obstacles.

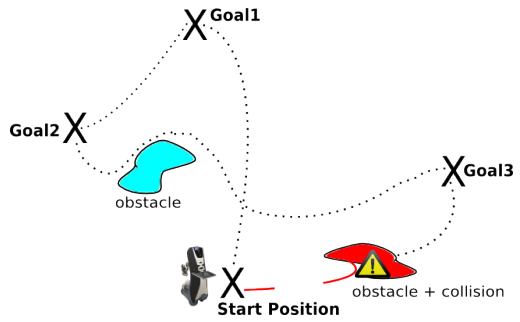


Figure 4: Simple view of the navigation and avoiding obstacles test.

During the execution of the test the benchmarking test will automatically record the data. After the test, the following metrics for the navigation benchmark will be applied and the results will be visualized in the component catalog.

- Time needed [in sec]
- Traveled distance [in m]
- Amount of rotation [in rad]
- Number of collisions [count]

4 Evaluation

The approach is evaluated for the Care-O-bot® Research community [11], which covers over a hundred of developers working on over 50 packages spread over various countries and time zones. There are partners who develop with real hardware systems [4] as well as partners who develop in simulation.
Figure 5 shows an extract of packages from the hardware layer as well as some high-level packages. For each, the number of active developers, forks, commits per month and dependencies is plotted, giving an inside of the development intensities.

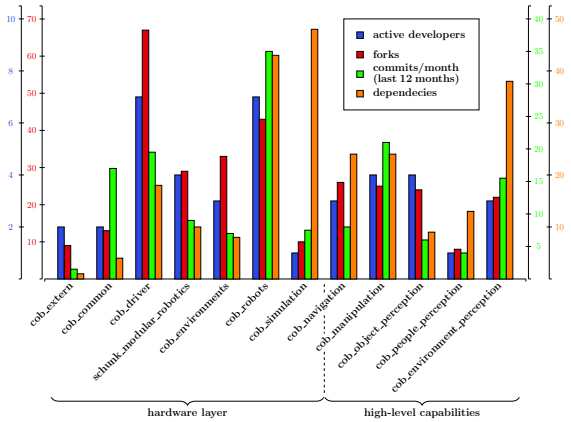


Figure 5: Extract of packages with numbers of active developers, forks, commits per month and dependencies.

On the used Jenkins CI server presently 130 job sets exist, this is equivalent to 1040 jobs. The daily number of builds is up to 50 to 400 jobs (on working days), as shown in Figure 6.

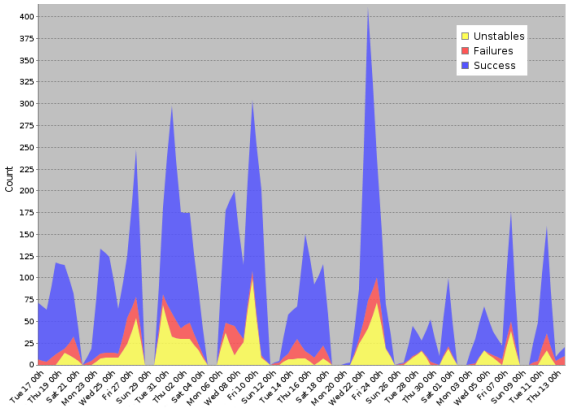
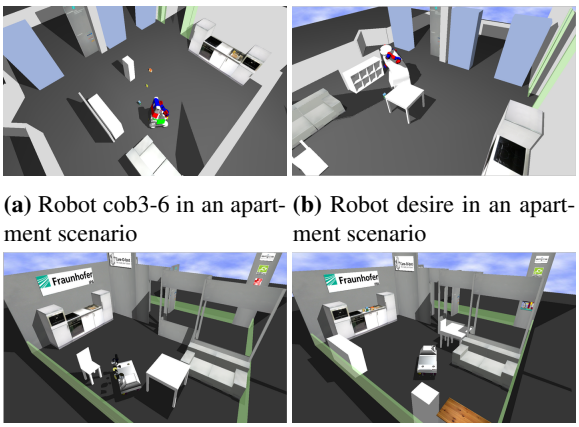


Figure 6: Overview of the builds per day for a two month period.

A detailed evaluation is realized on the basis of the `cob_navigation` package. Figure 7 shows different combinations of an application test for benchmarking a navigation component by testing it with various robot types. During the navigation tests, the following metrics were recorded (duration, distance, amount of rotation and number ob collosions). Figure 8 shows the benchmarking results on the component catalog website. These results can be used e.g. to help selecting the best navigation component for a given robot and environment setup. For example, figure 8a shows robot cob3-6 in an apartment scenario, where the robot navigates to various goals in an apartment scenario.



(a) Robot cob3-6 in an apartment scenario

(b) Robot desire in an apartment scenario

(c) Robot raw-3-1 in a kitchen scenario

(d) Robot raw-3-2 in a kitchen scenario

Figure 7: Navigation tests for different navigation components, robots and scenarios.

Listing 3 contains the definition of the navigation test case including the name for the action interface as well as goal positions.

ISBN 978-3-8007-3601-0
© VDE VERLAG GMBH · Berlin · Offenbach

Listing 3: Configuration file for testing navigation

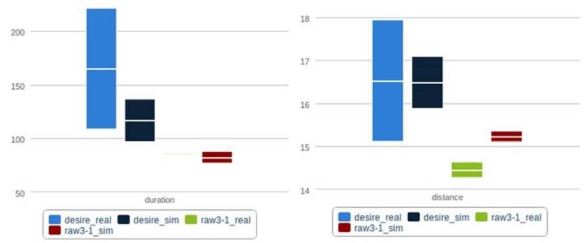
```
action_name: /move_base
xy_goal_tolerance: 0.3     # meters
yaw_goal_tolerance: 0.3    # meters
goals:
  - [2.250,−0.478, 0.222]  # [x,y,theta]
  - [0.250,−0.478, 0.222]  # [x,y,theta]
```

Figure 8 shows the results of the navigation benchmarking tests for two different robots, each tested in simulation and in reality. Looking at the graphs, a user can come up with the following conclusions:

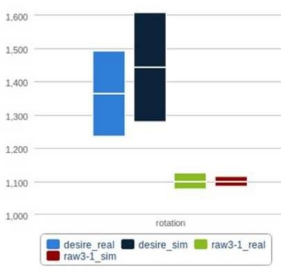
Result: Differences in robot configuration In general, the raw3-1 robot performs better because the average values for duration, distance and rotation are lower in contrast to the desire robot. This is due to a significant difference in the footprint of both robots. The footprint of the desire robot is bigger than for the raw3-1 and thus the desire robot faces more difficulties navigating in narrow environments or passing doors.

Result: Robustness and repeatability Looking at the statistical deviation of the minimum and maximum values, the raw3-1 does not only perform better regarding the mean values but also behaves more robust in terms of repeatability.

Result: Quality of the simulation model Comparing the results for both robots from reality to simulation, it is obvious that there is less difference between reality and simulation for the raw3-1 whereas there is quite a difference for the desire robot. Thus, the simulation model of raw3-1 fits better to the real world behaviour than the simulation model for the desire robot. This result can be used in order to judge the result gained from further simulation tests and express the trustfulness in results gained from simulations.



(a) Duration in [sec] **(b)** Distance in [m]



(c) Rotation in [rad]

Figure 8: Visualization the benchmark results of the component catalog for navigation.

Similar results can be extracted comparing various navigation algorithms for a fixed combination of robot and environment.
The results highlighted above direct the attention to further development as to the navigation component and robot layout with respect to sensors and actuator configurations.

5 Conclusion and outlook

A hierarchical setup of build and testing layers was introduced. It allows an easy way to configure automated testing in order to ensure the correct behavior of software components of a variety of robots, environments and applications. The whole process is automated in a way that the build and test jobs are automatically and individually triggered by every change in the code for each user in order to allow a fast feedback loop. By being able to fix errors early in the development process even before other developers get in touch with the defect, offers a certain level of quality in distributed open source development.
By using the proposed framework, a software component can be tested for a new set of boundary conditions in order to evaluate the usage for a specific target scenario. The framework supports general benchmarking of components by automated testing. The benefits of using the proposed testing platform in a distributed open source community was shown in section 4 by the example of a navigation component.
The component and scenario tests build up the basis for a composition of software components in order to test a whole application in the future. These tests can be carried out by combining simulation based tests with tests using previously recorded sensor data and, if available, the real robot hardware to a *hardware-and-simulation-in-the-loop* setup.
In the end the presented testing framework work will lead to a higher level of Software quality in distributed open source developments and provide the basis for a usage in the industrial domain, e.g. within the ROS-Industrial Initiative.

References

[1] F. Weisshardt, U. Reiser, C. Parlitz, and A. Verl, "Making high-tech service robot platforms available," *ISR/ROBOTIK 2010*, 2010.

[2] U. Reiser, C. Connette, J. Fischer, J. Kubacki, A. Bubeck, F. Weisshardt, T. Jacobs, C. Parlitz, M. Hagele, and A. Verl, "Care-O-bot® 3 – creating a product vision for service robot applications by integrating design and technology," in *The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, USA*. IEEE, 2009, pp. 1992–1998.

[3] S. Cousins, "ROS on the PR2 [ros topics]," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 3, pp. 23–25, 2010.

[4] (2012, Sept.) Care-O-bot distribution. [Online]. Available: http://www.ros.org/wiki/Robots/Care-O-bot/distribution

ISBN 978-3-8007-3601-0
© VDE VERLAG GMBH · Berlin · Offenbach

[5] (2012, Sept.) Rob@work 3. [Online]. Available: http://www.care-o-bot-research.org/robatwork-3

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[7] S. Cousins, B. Gerkey, K. Conley, and W. Garage, "Sharing software with ROS [ros topics]," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 2, pp. 12–14, 2010.

[8] S. Edwards and C. Lewis, "Ros-industrial: applying the robot operating system (ros) to industrial applications," in *IEEE Int. Conference on Robotics and Automation, ECHORD Workshop*, 2012.

[9] (2013, Nov.) ROS metrics. [Online]. Available: http://wiki.ros.org/Metrics

[10] (2013, Nov.) Jenkins CI. [Online]. Available: http://jenkins-ci.org/

[11] (2012, Sept.) Care-O-bot Research. [Online]. Available: http://www.care-o-bot-research.org

[12] (2013, Nov.) Continuous integration at OSRF. [Online]. Available: http://jenkins.ros.org/

[13] A. Bubeck, F. Weisshardt, T. Sing, U. Reiser, M. Hagele, and A. Verl, "Implementing best practices for systems integration and distributed software development in service robotics-the care-o-bot® robot family," in *System Integration (SII), 2012 IEEE/SICE International Symposium on*. IEEE, 2012, pp. 609–614.

[14] (2012, Sept.) ROS hz tests. [Online]. Available: http://www.ros.org/wiki/rostest/Nodes

[15] (2012, Sept.) ROS diagnostics. [Online]. Available: http://www.ros.org/wiki/diagnostics

Acknowledgments

The work leading to these results has received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement no 609206.

ISBN 978-3-8007-3601-0
© VDE VERLAG GMBH · Berlin · Offenbach