A Methodology for Testing Mobile Autonomous Robots

Jannik Laval, Luc Fabresse and Noury Bouraqadi
Mines-Telecom Institute, Mines Douai, France
e-mail: firstName.lastName@mines-douai.fr
web: http://car.mines-douai.fr

***Abstract*— Mobile autonomous robots are progressively entering the mass market. Thus, manufacturers have to perform quality assurance tests on series of robots. Therefore, tests should be repeatable and as much automated as possible. Tests are also performed for purpose of repairing robots. This calls for reusing tests already defined for quality assurance. In this paper we introduce a methodology to support the definition of repeatable, reusable, semi-automated tests. Our methodology describes the process of conducting tests in a way that maximizes safety for human operators, while avoiding to damage tested robots.**

I. Introduction

Mobile autonomous robots are finding their way to the mass market. Thus, robot manufacturers have to switch from the prototype building mode to industrial production line. Multiple instances of the same robot have to be built fast enough to meet the customers demand. These robots are supposed to be exact copies of the original prototype, and thus exhibit the very same behavior. Hence, the quality assurance (QA) team should test that produced robots meet actually with their specifications.

Since tests are likely to be conducted on a large number of robots, they should be *repeatable*. Running a given test several times should consist in making the tested robots perform the same actions under the same circumstances, in the same environment. Automation can help ensuring repeatability, while speeding up the test process.

Testing robots is also important for repair. Technicians have to test a robot to diagnose the actual defect and figure out the source of dysfunction. Repair should rely on tests to identify which robot behavior or parts do not comply with the specification.

Comparing a robot to its specification is a task shared between QA and robot repair. Since defining tests is tedious and time consuming, we advocate that tests should be *reused* between QA and maintenance teams. This sharing can be done in both ways. (i) Tests defined by QA should be reusable for repair. (ii) Technicians fixing robots may identify a frequent defect that is not detected by QA tests. Thus, they'll have to define new tests to detect such flaws. These tests should be inserted back in the QA test suite.

Test reuse can be pushed a step further, by focusing on behaviors rather than on robots. Indeed, behaviors matter more that the way they are implemented, at least from the customer point of view. Thus, tests that evaluate the way robots conform to a behavior, should be reusable and apply to different robots, even if they are built out of different parts.

In this paper, we argue that robots' tests should be *repeatable* and *reusable*. Last but not least, tests should be conducted in a way to maximize *safety*. Risks to harm human operators, or to damage the robot or other equipments should be minimal. The main contribution of this article is a test methodology to meet these requirements. This methodology is the result of our experience and lessons we learned while developing a service robotics application involving actual robots.

We have identified three test dimensions that are: the robot activity (sensing vs. acting), the environment nature (static vs. dynamic) and testers knowledge of the environment (little knowledge vs. good knowledge). These dimensions allow grouping tests in eight different sets. Starting from these test sets, and based on our requirements we re-organized and ordered tests ending up into a hierarchy of five test levels. Thus, our methodology guides roboticists to define *repeatable* and *reusable* tests covering various facets of a robot. The *automation* concern is also addressed during the test definition step. Last, our methodology describes the process of conducting tests while maximizing *safety* for operators, the robot and other equipments.

Other sections of this paper are organized as follows. Section II discusses requirements that should be met with a process to test robots. Then, Section III presents thoroughly our methodology. In Section IV, we illustrate our methodology by reporting our experiment testing two ROS-based robots as part of a development of a service robotics application. Next, Section V describes existing work related to robot test. Last, Section VI concludes the paper and sketches future work.

II. Requirements for Robot Test Process

To define requirements for testing robots, we took our inspiration from software testing. Chung et al. [5] already proposed to build tests adapted from software engineering. Testing software is a well-known process that has multiple advantages: it validates the behavior of a piece of code, it allows developers to refactor, maintain and improve the source code backed with tests as a validation process. Software tests also allow validating changes in the environment, such as a new Operating System. Applied to the robotics domain, we have identified and adapted 4 requirements:

- **Safety**: Safety is a key point during the tests execution. Indeed, since tests are run on robots that potentially have defects, it is important to ensure that they will

not damage themselves, someone or something in the environment. This means that tests should be executed in a precise order to first ensure the validity of basic functionalities before high-level ones that are likely more dangerous. Note that tests should also validate the fact that robots will operate safely.

- **Reuse**: Tests have to be reusable for heterogeneous robots built out of different components but which are supposed to exhibit the same behavior. For example, different robots may be equipped with equivalent though different sensors or actuators. Still, their behavior should be the same. This means that tests have to focus more on robots behaviors rather than their internal structure.
- **Repeatability**: Tests should allow testers repeat the same scenarios, and make robots face the exact same situations. In this regard, the dynamicity of the environment where mobile robots are supposed to face is challenging. Nevertheless, repeatability of tests is necessary to be able to validate the behavior of each robot at the end of a product line, or to ease the diagnostic when repairing a robot. One way to deal with repeatability is automation. Tests should be automated or at least semi-automated to ensure that robots will face the exact same situation on every test run. Another benefit of automation is higher productivity that results from minimizing operations performed by humans.

III. Our Methodology

A. Test Dimensions

Our methodology organizes tests into levels to maximize safety. These levels are derived from the following dimensions that we have identified in robot tests:

- Sensing only vs. sensing and acting: A robot can either sense its environment, or it can sense the environment while acting. Since acting relies on feedback obtained through sensors, tests involving actuators requires fully functional sensors.
- Testers Knowledge of the environment: To test a robot, tester should know the environment (e.g. dimensions of the arena as well as locations and sizes of obstacles) with some levels of accuracy. Based on this knowledge, testers can predict values of sensed data that reflect a correct behavior of the robot. Still, to test a robot more thoroughly, testers need to experiment with different environments, including ones for which they have little knowledge or imprecise one.
- Dynamicity of the environment for test: The environment could be static or dynamic. A static environment does not change while the robot performs its mission. A dynamic environment is one that evolves over time. Typically, it contains moving entities. Changes in sensed data can be either caused by the robot itself or by other entities.

Given an environment, one should run sensor tests first to ensure the sensors are working properly. Then, tests involving motion and actuation can be conducted, since sensing can be considered as reliable.

To help understanding the behavior of the robot, tests should be first done in an environment totally controlled by developers. It's easier to detect faulty behaviors and trigger security halts. Once a robot passes those tests, testers can proceed with experiments in less controlled environments.

Last, regarding the dynamicity dimension, safety requires that tests in static environments should be run before those related to dynamic environments. Robot should first be able to function and move in a stable world, before being able to cope with situations with mobile entities.

B. Ordering Tests for Safety

By combining the three previously defined dimensions, we can obtain eight different kinds of tests. Nevertheless, we consider some combinations of these dimensions as useless. This is the case of tests which check sensing capabilities in an unknown environment that can be either static or dynamic. Indeed, since the environment is not controlled by developers, it makes it difficult for them to define meaningful tests, since they can not predict valid values for sensed data.

We ordered the remaining sets of tests in a way that maximizes safety. Figure 1 shows the resulting tests order organized in five safety levels. Upper levels are prerequisites for lower ones. This means that a test should not be performed on a robot until all tests in the levels above pass. This order is based on the following assertions. For the first dimension, we consider that sensing alone is safer than when acting is involved. For the second dimension, we consider that moving in a known environment is safer that moving in an unknown one. For the third dimension, we consider that a static environment is safer that a dynamic one.

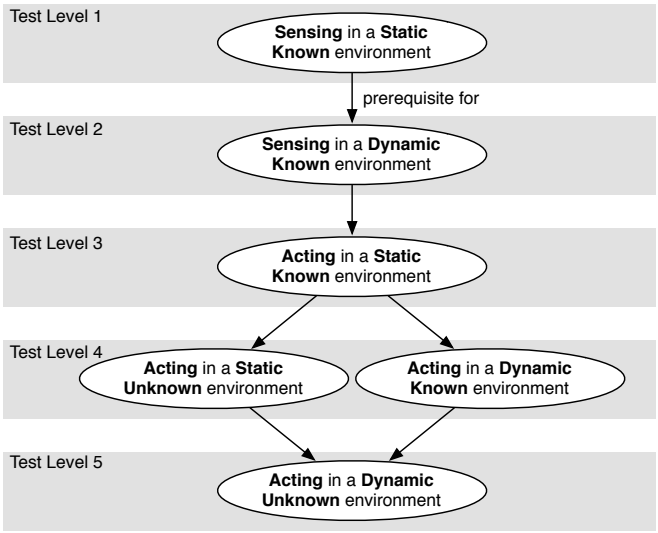

Fig. 1. Five Safety Test Levels

We now present each test level:

1) Testing robot *sensing* capabilities in a *static known* environment. This test suite ensures that data produced by sensors matches the expected accuracy level.
2) Testing robot *sensing* capabilities in a *dynamic known* environment. Such an environment is controlled by

developers, even if it is dynamic. For example a room with some moving entities following a known path or with the light changing according to some known patterns. Compared to sensing in a static environment, a dynamic one introduces the time factor. In a dynamic environment, tests are more about the frequency of collected data than their accuracy. For example, tests in this level may evaluate if a robot can collect data fast enough to detect potential obstacles.

3) Testing robot *action* in a *static known* environment. Here the goal is to ensure that the robot actuators work properly, and that the robot performs basic actions. Still these tests can make use of sensors, since the robot is likely to behave based on perception. Besides, testers should ensure that they have some solutions to stop the robot in case of emergency (e.g. an emergency stop button).
4) This level contains two complementary kinds of tests related to robot *action* in both *static unknown* and *dynamic known* environments.
   - Testing robot *action* in *static unknown* environment. The goal of these tests is to increase confidence in the tested robot, by making it faces many different situations. Thus, testers have more chance to detect unexpected or unwanted behaviors in situations uncovered by tests in a static known environment. An example of such tests, is putting a vacuum cleaner robot in a room full of obstacles and check that it does not get stuck in between.
   - Testing robot *action* in a *dynamic known* environment. At this stage, testers can predict sensed data. Since the environment is supposed to be known, they can also predict when events occur and test if the robot performs the right actions. Tests do not only ensure that the robot does the right action, but they also verify that the behavior is fast enough. For example, knowing trajectories of all entities of a given environment, one can test obstacle avoidance in a dynamic environment.
5) Testing robot *action* in a *dynamic unknown* environment. This level represents the final tests. It aims at confronting tested robots to as many different situations as possible, especially ones unforeseen by testers. Tests are considered as passed if the robot can cope fast enough with encountered events. An example of a such test is making a mobile robot search for a person based on face recognition, in a place many people passing by (e.g. a train station).

C. Defining Reusable and Repeatable Tests

The definition of a test consists of:

- Test level. This information allows developers to identify prerequisite tests that should pass before attempting to run the current test. This maximizes safety, but also eases identifying issues. For example, consider a robot that collides with obstacles. Collisions can have different causes, including a malfunctioning sensor. Thus, tests for robot motion should not run until all tests for sensors pass.
- Initial conditions. They refer to the state of the environment (e.g. light intensity, obstacle positions). They also include the state of the robot (e.g. its pose in the environment). Initial conditions should be set prior to running the test. They ensure the repeatability of the test. Ideally, the test setup should be automated. At least, human action should be limited, which have the nice side effect of speeding up the test process. Note that for the sake of reusability, initial conditions should be as little as possible bound to the robot internals.
- Expected outcome. Each test should evaluate some facets of a robot, such as sensed data or some behavior. When dealing with numerical data, the expected outcome should be expressed as an interval covering the range of valid values. This enables reusability since different robots with different components can produce different values. It also allows repeatability, because of the noise, but also because slight differences when setting up initial conditions (e.g. initial pose of the tested robot). In some tests, the expected outcome can be more qualitative. An example is "robot should follow smooth trajectories".
- Task to perform. This item defines the task to be performed by the robot, and which outcome will be compared to the requirements. The task can be as simple as reading a sensor value, or a more complex task such as fetching a drink from a fridge.
- Evaluation. It is performed once the robot had finished the task to do. The evaluation refers to the comparison of the actual outcome to the expected one. Ideally, this comparison is performed automatically. However, human operator intervention might be required in situations such as measuring the travelled distance. Human evaluation is even difficult if not impossible to replace for qualitative tests, which are typically ones run in unknown environments (levels 4 and 5).
- Minimum runs count. Tests usually should be run more than once. This is true when evaluating that some value falls inside some range. But, this is also the case when it comes to qualitative tests.

D. How to use Tests?

Test levels we have introduced in Section III-B can be used for QA and repair. But, they can also be used for incrementally developing a control software for a robot, in the following iterative process derived from Test-Driven Development [1] (TDD):

1: Select a test level: Respecting the order given in Section III-B, the tester has to select a test level such as all its prerequisites pass. Initially, the first level will be chosen.

2: Define tests for the selected level, and implement the corresponding features. This is done in a cycle, addressing one feature at a time, as following:

2.1 Define a test for a specific feature: The test should be defined as described in Section III-C and focus

on a single feature of the robot. Tested features can be of different granularity ranging from simple sensing to complex behaviors.

2.2 Develop the feature to make the test pass: Here developers write the code that is supposed to implement the targeted feature. Developers should run the test and adapt the code they have written until the test passes.

2.3 Enhance the design: At this step, developers should review their code and enhance it from the software design point of view (e.g. use of design patterns [6]). This step is important from future software evolution and maintenance. At the end of this step, developers should ensure that the test still passes, meaning that the design was enhanced while ensuring the correct functioning of the developed feature.

2.4 Go back to step 2.1 if the current test level is not covered yet. Otherwise go to step 3.

3: Run all the tests of the current level: This step ensures that there is no regression, i.e. features that do not work anymore because of the implementation of other features.

4: Go back to step 1 if there are some test levels left.

Note that tests that evaluate the robot in unknown environments (Level 5 and part of level 4) often do not result into new features. Instead, they allow identifying situations that developers didn't though of, where the robot fails. These situations should be converted into tests that should be added to the appropriate level.

This TDD approach allows one to safely and reliably develop the software for some robotic mission. Besides, it has a valuable sub-product, namely the tests. Indeed, tests can be reused afterwards for repair and for QA in a production line.

IV. Experiments

The context of our experiment is the CAIRE project[1] which targets service robotics applications such as new services to shopping mall customers. We bought two identical robots, weighting 30 kg each. They are 2 wheels differential drive robots, which maximum speed is 1 m/s. They have multiple sensors: 1 laser, 16 infrared sensors, 9 ultra sonic sensors, 2 webcams, and 2 wheel rotary encoders for odometry. We wrapped the proprietary firmware into a ROS node to communicate with these robots and benefit from the ROS ecosystem. We developed this node using PhaROS[2], a ROS client implemented with the Pharo open-source Smalltalk environment[3].

In this section, we describe some tests defined for these two robots using our methodology.

[1] http://car.mines-douai.fr/category/project/caire/
[2] http://car.mines-douai.fr/category/pharos/
[3] http://www.pharo-project.org

A. Sensing in a Static Known Environment (Level 1)

We expose here tests for the laser range sensor, a Sick S300[4] with a 270 degrees scan angle. The distance measuring range is 30 meters. We put the robot in a known environment: a box of one square meter. (illlustrated by Figure 2) This environment is static, since there is no moving entity inside the box.

Fig. 2. Testing the Robot Sensors in a Static Known Environment

We execute five tests inside the box, one with the robot located at the center of the box, and the others with the robot at each corner. These tests are semi-automatic, since human intervention is required for placing the robot at the appropriate positions in the box. Expected distances measured by the laser should range between 20cm (half of the robot width) and 60cm (half of the box width + 10cm) when the robot is in the middle of the box. The maximum values can go up to 80cm when the robot is at corners. Distance measurement and comparison with expected values is done automatically by a test software.

Results show that distances for angles 0 to 1 and for angles 269 to 270 are too small (less that 18 centimeters) for both robots. After analysis, we found out that lasers were badly mounted. The border of the robot body is too close to the laser. This problem is easy to fix. So, we could proceed using both robots.

B. Sensing in a Dynamic Known Environment (Level 2)

The example we provide here evaluate how fast is the onboard laser. Our goal is to ensure that the scan frequency is high enough for detecting obstacles that are less than 20cm far of the robot, before collisions.

The average human walking speed is approximatively 1.49m/s (5.0km/h). The maximum speed of our robots is 1m/s. Suppose that we have a robot at maximum speed and a walking human moving towards each other. If they are at 20cm one from the other, the collision will occur in approximatively 80ms. So, to ensure safety, we expect that

[4] http://www.sick.com/

the robot detects the danger in less than 80ms. Thus, the expected laser scan frequency should be at least 12.5Hz.

Data collected by the laser is published in a ROS topic named /scan[5]. Our test uses the rostopic hz /scan[6] command provided by ROS to automatically measure the frequency of publications. Thus, our test takes into account delays introduced by processing raw data from the laser and transforming into the required format for publication in a ROS topic.

It turns out that the scans frequency is approximatively 15Hz. This means that the robot detects obstacle position changes every 67ms approximatively. We can conclude that the laser scans frequency is high enough to allow the robot to detect potential collisions.

C. Acting in a Static Known Environment (Level 3)

Based on previous tests, we are confident about the sensing capabilities of our robots. Besides, we verified that the emergency stop button is working properly. In this section, we report tests we performed to check whether the robot can move accurately at different speeds (0.25 m/s, 0.5ms, 1 m/s), based only on odometry, in a known static environment.

1) Moving in a Corridor: The environment we used in this series of tests is a section of a corridor which both ends were closed using boards. The corridor section is 5m long and 1.30m wide. The robot which is 40cm wide was put at one end of the section, 50cm far from the closing board and 45cm far from the walls. This setup requires human operations.

We run different tests where the robot moving at different speeds was expected to travel 4m in a straight line. A 3% margin of error was considered as acceptable. We measured the traveled distance ourselves (human operation). But, we relied on the laser to automatically check that the distances to both walls remain higher than 30cm from walls.

We repeated the experience 10 times for each robot at different speeds. We observed that one robot can actually travel forward at least 4m with an average error 7.5cm i.e 1.87% which is acceptable. For the second robot, test failed 5 times when the robot went too close to the wall. We identified that the problem comes from a wheel partially blocked by a broken spring. Because this robot failed to pass this test, we did not run higher-level tests on it.

2) Following a S-Shaped Path: Knowing that one of our robot can move forward accurately, we checked more complex trajectories. We set up a test where the robot had to follow a s-shape path. We marked on the floor 4 points of the path coordinates: (0,0), (1,-0.5), (3,0.5), and (4,0). This path corresponds to a cumulative rotation of 180 degrees. The robot was expected to pass by every one of these waypoints. We considered 3% as an acceptable margin error for travelled distance, 10% for rotation.

The test requires human intervention for putting the robot at point (0,0). Human operators were also required to evaluate that the robot goes through the way points, and stops at an acceptable position. Measurement of the final position revealed that the robot meets our expectations. The traveled distance margin error is approximatively 1.9%, and the rotation margin error is approximatively 9.2%.

D. Acting in a Dynamic Known Environment (Level 4)

We illustrate tests for this level with the example of emergency stop in case an obstacle comes too close to a moving robot. The experiment consists in making a robot move forward following a straight line in a corridor. The corridor is initially empty, but when the robot reaches a particular position marked in the floor, an operator puts a cardboard panel at 30cm from the robot. The robot is expected to stop immediately without touching the board.

We repeated the experiment several times with the only robot that passes the tests from the previous level. It turns out that the robot passes this test for different speeds ranging from 0.25m/s up to 1m/s.

The experiments allow us to discover issues on robots that are not easy to highlight without tests. Moreover, after repairing the robot failing on level 3, we will reuse our tests and validate its correct reparation.

V. Related Work

Koo Chung et al. [5] propose a guideline to adapt the ISO standard for software testing to the components for robotics. The paper takes the main items of ISO 9126 and adapt them to each component of the robot. Like in ISO, the approach is based on scenario. Compared to this work, our approach enables to test not only isolated components, but also the whole robot. Besides, Chung et al. did not consider the safety issue, neither test reuse. They however support some level of repeatability by defining expectations for each test.

Biggs [3] proposal also focuses only on components. He proposes a repeatable regression testing method for software components that interact with hardware. In a first session, the approach backups data from the hardware on a particular port. Then, the port is emulated and receives previously saved data independently of the hardware. This technique is clearly repeatable after the first session where data is collected. It is also safe, since the hardware is not involved when running tests. The limitation is that it focuses only on individual components, and more specifically software ones.

Bensalem et al. [2] propose a safe-by-construction architecture, based on a formal description. However, this approach is limited to the functional level. It thus should be complemented by tests of higher levels, such as the ones we proposed.

Many approaches use simulation to test the software before plugging it into the hardware. For example, SITAF [9] is a framework to test robot components by simulating environment. It generates test cases based on a specification given by the developer. This test generation combined with simulations allows repeatability of tests. It also discards the need of test reuse, since they are generated. Besides, working

[5]We eventually connected this topic to the GMapping SLAM algorithm from OpenSlam

[6]see rostopic documentation http://www.ros.org/wiki/rostopic#rostopic_hz

at the simulation level allows detecting flaws without taking any risk either for humans or for the robot. However, although some simulations can be realistic to some levels, tests with actual robots as we discussed in our methodology, are still needed.

Among work relying on simulations, some introduce hardware-in-the-Loop. For example, Chen et al. [4] propose to insert an extra step for hybrid tests between simulation and tests in real world. The same idea was taken a step further by Petters et al. [8] who present multi-level testing strategies for teams of autonomous robots. The authors propose three levels of tests: component tests (automated), online test (with the judgement of a human) and offline test (using logs). Each of these levels can be run with simulated robots, or with actual robots. Although the availability of different test levels allows better safety, the authors provide no guideline to help engineers selecting the appropriate level. Among work relying on ordering tests, Son et al. [10] propose 3 levels of tests: unit testing, state testing and API testing. The 3 levels allow to evaluate performance of components. The presented system allows one to validate the safety of the tested components by measuring their durability for example.

Lim et al. [7] also propose a test method for robot components. The method is based on OPRoS [11]. The authors propose an IDE that generates automatically test cases. In this work, there are also 3 levels for tests: unit testing, integration testing and system testing. This work provides safety, test reusability and automation, but it is particularly applied to the software part of the system. Applying their method to a whole robot would conduce to the same problems we discussed in this paper.

VI. Summary and Future Work

Developing mobile autonomous robots is a challenging task. Making reliable robots for the mass market is even more challenging. This is why we argue that the robotic industry needs a rationalized process, especially at the end of product lines for quality assurance. In this regard, we believe that tests should play an important role in the process of making robots.

Testing robots should be performed in a way that is safe for human operators, and that preserves robots and other expensive equipments. Besides, tests, especially in the context of product lines, should be repeatable and as much automatic as possible. The methodology we have introduced in this paper targets this goal.

Our methodology builds upon three dimensions for robotic tests. The first dimension is about operations performed by the tested robot: sensing only vs. sensing and acting. The second dimension is about testers knowledge of the environment were tests are performed: they can have a precise knowledge of the environment vs. little knowledge of it. The last dimension is about the environment: it can be static vs. dynamic with other moving entities for example.

The above three dimensions allow us to group tests in different sets. Based on an analysis of these sets, we have identified five relevant test levels. We ordered them to maximize safety of humans while limiting risks to damage tested robots. Indeed, testers must ensure that a robot passes lower levels, before proceeding with tests from higher levels.

We reported our experiments with two similar ROS-based robots. These experiments were conducting while developing a demo application that will be showcased in an industrial event next october 2013. Our methodology allowed us to develop a semi-automatic repeatable test suite. This suite already help us to identify different defects of the robots and fix them. Since we can re-run it at will, we can reuse it in the future to check our robots' correct functioning.

Regarding future work, we plan to focus on repeatability and automation of tests. Currently, some tests require human intervention for the setup or for checking the result conformance to requirements. Ideally, test environments should be instrumented, similarly to an automated product line. Another interesting perspective derives from our experience defining tests. We will investigate how software tools can help human testers to develop tests and enforce their execution order.

Acknowledgments

This work is supported by Nord-Pas de Calais Regional Council through the CAIRE project (2012-2014). We also thank Anthony Fleury for his comments and feedback.

We gratefully acknowledge the sponsoring of ESUG (the European Smalltalk User Group) http://www.esug.org/.

References

[1] Kent Beck. *Extreme Programming Explained*. Addison-Wesley, 2001.

[2] Saddek Bensalem, Lavindra de Silva, Félix Ingrand, and Rongjie Yan. A verifiable and correct-by-construction controller for robot functional levels. *Journal of Software Engineering for Robotics*, 2(1):1–19, September 2011.

[3] G. Biggs. Applying regression testing to software for robot hardware interaction. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4621–4626, 2010.

[4] Ian Yen-Hung Chen, Bruce A. MacDonald, and Burkhard C. Wünsche. A flexible mixed reality simulation framework for software development in robotics. *Journal of Software Engineering for Robotics*, 2(1):40–54, September 2011.

[5] Yun Koo Chung and Sun-Myung Hwang. Software testing for intelligent robots. In *International Conference on Control, Automation and Systems 2007*, pages 2344–2349. IEEE, 2007.

[6] Erich Gamma, Richard Helem, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] Jae-Hee Lim, Suk-Hoon Song, Jung-Rye Son, Tae-Yong Kuc, Hong-Seong Park, and Hong-Seak Kim. An automated test method for robot platform and its components. *International Journal of Software Engineering and Its Applications*, 4(3):9–18, July 2010.

[8] S. Petters, D. Thomas, M. Friedmann, and O. Von Stryk. Multilevel testing of control software for teams of autonomous mobile robots. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 183–194, 2008.

[9] Hong Seong and Jeong Seok. SITAF: simulation-based interface testing automation framework for robot software component. In Florian Kongoli, editor, *Automation*. InTech, July 2012.

[10] Jung-Rye Son, Tae-Yong Kuc, Jong-Koo Park, and Hong-Seok Kim. Simulation based functional and performance evaluation of robot components and modules. In *Information Science and Applications (ICISA), 2011 International Conference on*, pages 1–7. IEEE, April 2011.

[11] Byoungyoul Song, Seungwoog Jung, Choulsoo Jang, and Sunghoon Kim. An introduction to robot component model for opros (open platform for robotic services). In *Workshop Proceedings of SIMPAR*, pages 592–603, 2008.