Towards Test-Driven Development for Mobile Robots

Luc Fabresse[1], Jannik Laval[1] and Noury Bouraqadi[1]

I. Test-Driven Development

Test-Driven Development methodology (TDD) is an agile process to build software incrementally [**?**]. The main principle is that the development is driven by tests. Developers use tests as a guide to produce software and ensure its conformity to requirements. This process allows developers to iterate on short development cycles and test functionalities as soon as possible in the development process. Each iteration begins by coding tests that reflect a particular requirement. The goal is to translate requirements into code that tests the software. Then, developers write or adapt the software to conform the tests. Each time a new functionality is integrated, all the tests are re-executed, which assure regression testing.

II. Why TDD for Robotics?

Developing robots combines the challenge of building hardware and the corresponding software. Even in the case of an existing robotic hardware, the development of control software alone is more challenging than plain software development. Indeed, it introduces the hardware integration constraint and the physical problems. Using TDD makes developers to express the robot specification in tests. Then, these covering tests can help at different stages. For example, tests produced during prototyping stage can be reused by the quality assurance (QA) team to verify that a newly produced robot conforms to the specification of its product line. Another example of usage of these tests is the maintenance of a robot. Failing tests can help to quickly and accurately identify defects of a robot. Passing the same tests validates that repair actually solves all problems.

[1] Mines-Telecom Institute, Mines Douai, France, e-mail: firstName.lastName@mines-douai.fr Web: http://car.mines-douai.fr

III. Requirements for Robot Test

In the context of robotics, we have identified four requirements for TDD:

- **Repeatability**: A mobile autonomous robot operates in a dynamic environment. This implies that the source of an error is difficult to identify. For example, an error of a sensor can be hidden by an external event. The repeatability of the tests is necessary to be able to validate the behavior of a tested robot, by ensuring that test conditions are always the same.
- **Reuse**: Tests related to high-level behavior have to be reusable for robots built out of different components but which are supposed to exhibit the same behavior.
- **Safety**: Since tests are run on robots potentially with defects, it is important to ensure that they will not hurt someone, damage themselves, or break something in the environment. This means that tests should be executed in a precise order to first ensure the validity of basic functionalities (sensing activities) before high-level ones (acting activities) that are likely more dangerous.
- **Automation**: Tests should be run with as few human action as possible. Typically, at the end of a product line where all built robots should undergo QA tests, automation is necessary to speed up the test process.

IV. Tools for Robotics TDD

Software TDD relies on a set of tools to help developers. We believe that Robotics TDD requires similar tools. This is why we are currently developing BoTest, a tool to support expressing and running robotics tests.

It is implemented in the Pharo programming language [**?**], on top of the SUnit unit-test framework. So far, it provides support for expressing dependencies between tests. This is useful from

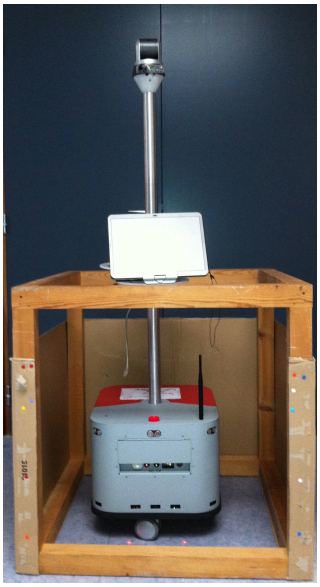

Figure 1: Testing Robot's Laser Rangefinder Inside a Box

the safety point of view as well as for quickly identifying the lowest-level failing test. It also guides testers for setting up the initial physical conditions of a test (e.g. robot pose, obstacles, light). Besides, it enables expressing verifications that involve testers such as measuring the distance travelled by a robot.

As part of our CAIRE project, we are using BoTest to develop the software that controls two *identical* robots we bought. We started by writing tests based on specifications provided by the manufacturer and going up towards our application level. We provide here an example related to the laser rangefinder. We put a robot inside a 1 meter wide cube box (see Figure 1) and check that measured distances are within the expected range. The code below corresponds to the test where the robot is located in the bottom left corner of the box.

```
1  testLaserWhenRobotAtBottomLeftOfTheBox
2    | laser notification |
3    self requestAction: 'Please, put the robot at
       the bottom left of the 1x1 box'.
4    laser := robot laserService.
5    laser enableNotificationsEvery: 10.
6    notification := self notificationOrNilFrom: laser.
7    allDistances := notification distances.
8    self assert: allDistances areLessThan: 0.75
       andGreaterThan: 0.25
```

Test starts by requesting the operator to put the robot at the right position (line 3). Then, we trigger the laser service of the robot to publish notifications every 10 clock cycles (lines 4-5). Each notification gathers all measurements performed in a single laser scan. Last, we check that all measured distances are less than 0.75 meter and greater than 0.25 meter (line 8). Running this test allowed us to identify that the laser was partially covered with the robot structure. Indeed, some measured values were smaller than the minimum expected distance.

V. Summary and Future Work

We argue that robotics software development would really benefit from Test-Driven Development. We believe that better tool support would help to widespread TDD for robotics. We introduced BoTest, our first step towards this end. It helps developers to write *repeatable* and *reusable* tests. It also helps to execute tests in a *safe* order based on requirements expressed by developers.

This work is part of a wider ongoing effort around the CAIRE project which one of the goals is to apply *agile principles* [**?**] to the development of robotics software. This requires to adapt these software technics to the robotics specificities and constraints. An example of such constraints is the physical dimension of robots, which requires human actions to perform tests. We would like to go further in automating such tests and reducing human involvement.

Acknowledgements

This work is supported by Nord-Pas de Calais Regional Council through the CAIRE project (2012-2014).

References

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

[2] J. Highsmith and M. Fowler. The agile manifesto. *Software Development Magazine*, 9(8):29–30, 2001.

[3] O. Nierstrasz, S. Ducasse, and D. Pollet. *Pharo by Example*. Square Bracket Associates, July 2010.