An Approach to Apply Automated Acceptance Testing for Industrial Robotic Systems

1st Marcela G. Dos Santos
Université du Québec à Chicoutimi
Chicoutimi, Canada
marcela.santos1@uqac.ca

2nd Fabio Petrillo
École de Technologie Supérieure (ÉTS)
Montreal, QC, Canada
fabio.petrillo@etsmtl.ca

3rd Sylvain Hallé
Université du Québec à Chicoutimi
Chicoutimi, Canada
shalle@acm.org

4th Yann-Gaël Guéhéneuc
Concordia University
Montreal, QC, Canada
yann-gael.gueheneuc@concordia.ca

***Abstract*—Industrial robotic systems (IRS) are systems composed of industrial robots that automate industrial processes. They execute repetitive tasks with high accuracy, replacing or supporting dangerous jobs. Consequently, a low failure rate is crucial in IRS. However, to the best of our knowledge, there is a lack of automated software testing for industrial robots. In this paper, we describe a test strategy implementation to apply BDD to automate acceptance testing for IRS.**

***Index Terms*—robotics, industrial robots, software testing, automated testing, acceptance testing**

I. Introduction

To raise the software quality, we can apply software testing. Software testing is a process to test software to find and remove as many software failures as possible [1]. One definition of software failure for conventional systems is when the customers' expectations have not been met and–or when the software does not help the customer [2]. Developers and stakeholders can collaborate in conventional software systems to improve software quality using automated acceptance testing (AAT) [3].

We performed an experiment in the Gazebo simulator running in the Robot Operating System (ROS) environment to apply a test strategy implementation to apply BDD to automate acceptance testing for IRS. The robot model used was the Gen3 from Kinova [4] and the end-effector is a gripper, the Robotiq-2f-85 from Robotiq [5].

II. Background

The prerequisite to applying our strategy is the business requirements (BR) and the acceptance criteria (AC). Thus, the business analyst and the technical team (developer and tester) need to rewrite the BR using the BDD template. The outputs of this step are features and scenarios. After that, the technical team writes and executes the automated tests, named step definitions in our approach. The step definitions interact with the code that defines the robot behaviour, named mission in our approach. At first, scenarios will fail because we did not write any mission. So, the technical team must write the mission until the scenario passes.

ROS is a standard framework for developing robotic software. It is a collection of libraries, tools, and conventions. It simplifies the task of creating robot software. A system created with ROS is a system with many different programs running simultaneously and communicating with one another, passing messages.

Gazebo is a popular robotic simulator that can simulate different types of robots with many commonly used sensors, such as cameras, GPS, and IMU. Although Gazebo is an independent project, it is possible to integrate it with ROS through the *gazebo_ros* package and establish bidirectional communication between them (Gazebo and ROS).

To communicate with the Kinova Gen3 robot, we use the API *Kinova Kortex* integrated with ROS through the package *ROS Kortex*. ROS Kortex allows us to use the ROS mechanisms (communication and file system) and Gazebo as a simulator to perform the experiments using Kinova Gen3.

To automate the tests, we use the library *pytest-bdd* [6], the most popular framework in Python that implements a subset of the Gherkin language to enable automating project requirements testing.

To aid other researchers in the reproduction of our results, we share the code, replication instructions and a video to illustrate both case studies performed at `https://github.com/mgdossantos/irc2022-aat4irs`.

III. Approach

Our approach is a test strategy implementation to apply BDD to automate acceptance testing for IRS. The prerequisite to applying our approach is the business requirements (BR) and the acceptance criteria (AC). Thus, the business analyst and the technical team (developer and tester) need to rewrite the BR using the BDD template. The outputs of this step are features and scenarios. After that, the technical team writes and executes the automated tests, named step definitions in our approach. The step definitions interact with the code that defines the robot behavior, named mission in our approach. At first, scenarios will fail because we did not write any mission.

So, the technical team must write the mission until the scenario passes.

The implementation of the approach is organized in three packages: mission, tests, and robot. **Mission** as the package that contained all the files to support the implementation of the code with the robot's behavior as well the mission code. The **robot** package represents the robot in general that can be simulated or physical robot. Finally, we have the package **tests**. Inside the tests, we have the feature and the scenario.

IV. Experiment

In our experiment, the system must pick one-by-one objects placed on a conveyor, transport them, and place them on a delivery table. We move the IRS from Point A (on the conveyor) to Point B (on the delivery table). The video illustrating this case study can be accessed at the following URL: `https://youtu.be/9S0R0mGKA-I`.

Pick-and-place is a typical use case for industrial robots. We break down the pick and place process into four phases: pre-grasping, grasping, transport, and placement [7].

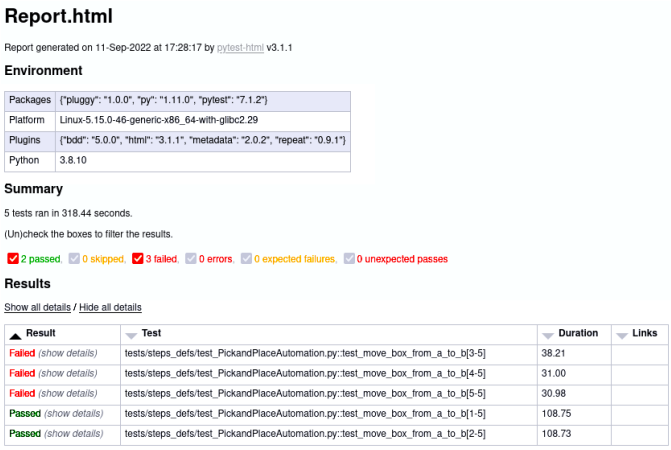
To run our simulation, the first step is to launch the world in Gazebo using all the models (IRS, conveyor, and table). The command is `roslaunch caseStudy spawn_kortex_robot.launch`. Then, we create the object that will be picked and placed by the IRS. Our choice was to create the object in other Python scripts to allow us to create objects with different positions in each simulation. To create the object using our Python script we need to use another ROS command: `rosrun caseStudy box.py`. In each new simulation, we have the box position generated randomly on the X-axis.

We then write the tests. The code for each function is the way that the test framework interacts with the mission. As we are using a simulator, we must define automatic communication between the test and the mission. We used *pytest-gui* library that provides a GUI test runner for Python tests.

In our study, when we run the test for the first time, there is no code to automate the pick-and-place process. Thus, we implement the mission until all the tests pass.

To test a software, it needs to establish an outcome. For CS2, we establish the outcome as the IRS and box coordinates (X-axis) with an acceptable error of 0.02. For example, the function box_position is responsible for interacting with the application to read the box position and make the assertion. The test will pass if the difference between the experiment read-box position (X-axis) and the ideal value (0.55) is less or equal to 0.02.

The next stage was to inject a fault to simulate a noise that can happen in the reading of the box position. We added a randomly generated value between -0.025 and +0.025 in the box position (X-axis). We have access to the test report, in which we have the information related to the tests that pass and not, and the reason why some of the tests did not pass. Figure 1 presents the report overview for this case study, we ran the experiment 5 times.



Report.html

Report generated on 11-Sep-2022 at 17:28:17 by pytest-html v3.1.1

Environment

Packages	{"pluggy": "1.0.0", "py": "1.11.0", "pytest": "7.1.2"}
Platform	Linux-5.15.0-46-generic-x86_64-with-glibc2.29
Plugins	{"bdd": "5.0.0", "html": "3.1.1", "metadata": "2.0.2", "repeat": "0.9.1"}
Python	3.8.10

Summary

5 tests ran in 318.44 seconds.

(Un)check the boxes to filter the results.

2 passed, 0 skipped, 3 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Failed *(show details)*	tests/steps_defs/test_PickandPlaceAutomation.py::test_move_box_from_a_to_b[3-5]	38.21	
Failed *(show details)*	tests/steps_defs/test_PickandPlaceAutomation.py::test_move_box_from_a_to_b[4-5]	31.00	
Failed *(show details)*	tests/steps_defs/test_PickandPlaceAutomation.py::test_move_box_from_a_to_b[5-5]	30.98	
Passed *(show details)*	tests/steps_defs/test_PickandPlaceAutomation.py::test_move_box_from_a_to_b[1-5]	108.75	
Passed *(show details)*	tests/steps_defs/test_PickandPlaceAutomation.py::test_move_box_from_a_to_b[2-5]	108.73	

Fig. 1: Report Overview

The report shows that three tests failed, and two passed. The figure shows the reason why simulation three did not pass, the box's final position was not as expected. We define the expected outcome using a business requirement, the box had to be in a specific position (A) with an acceptable difference of less than 0.02.

V. Conclusion

This study presented a test strategy to apply automated acceptance testing for industrial robotic systems. We presented this experiment that relies on Behavior-Driven Development and implemented it using ROS, Gazebo, and a Python library for BDD (*pytest-bdd*).

In future work, we intend to apply our approach to physical robots. We also want to apply a quantitative measure to evaluate the effort to discover faults. Finally, we want to evaluate our approach by performing controlled experiments with practitioners that perform different roles in robotic system development teams (business analyst, tester, developer).

Acknowledgements

The NSERC Discovery Grant and Canada Research Chairs programs partially supported the authors.

References

[1] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Wiley Publishing, 2011.

[2] R. Chillarege, "What is software failure?" *IEEE Transactions on Reliability*, vol. 45, no. 3, pp. 354–, 1996.

[3] M. Wynne and A. Hellesoy, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2012.

[4] K. Robots, "Discover our gen3 robots — kinova," https://www.kinovarobotics.com/product/gen3-robots, 2022, accessed: 2022-08-19.

[5] Robotiq, "Start production faster — robotiq," https://robotiq.com/products/2f85-140-adaptive-robot-gripper, 2022, accessed: 2022-08-19.

[6] O. Pidsadnyi and A. Bubenkov, "Welcome to pytest-bdd's documentation!" https://robotiq.com/products/2f85-140-adaptive-robot-gripper, 2022, accessed: 2022-09-03.

[7] H. Mnyusiwalla, P. Triantafyllou, P. Sotiropoulos, M. A. Roa, W. Friedl, A. M. Sundaram, D. Russell, and G. Deacon, "A bin-picking benchmark for systematic evaluation of robotic pick-and-place systems," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1389–1396, 2020.