Test Generation for Robotized Paint Systems using Constraint Programming in a Continuous Integration Environment

Morten Mossige*‡, Arnaud Gotlieb†, Hein Meling‡
*ABB Robotics, Norway. †Simula Research Laboratory, Norway. ‡University of Stavanger, Norway.
Email: morten.mossige@uis.no, arnaud@simula.no, hein.meling@uis.no

***Abstract*—Advanced industrial robots usually consist of several independent control systems. Particularly, robots that perform process-intensive tasks like painting, gluing, and sealing have dedicated *process control systems* that are more or less loosely coupled with the motion control system. Testing the software for such systems is challenging because physical systems are necessary to test many of their characteristics.**

This paper proposes a method for automated testing of such robot systems. Our approach draws on previous work on continuous integration, combined with constraint programming techniques for test sequence generation. In ABB Robotics' process control system for robotized painting, many tests are only conducted every six months, during the release test. With our automated test approach, we expect to reduce the round-trip time, from code change to test completion, to less than one day.

I. Introduction

There are many challenges in testing an integrated process control system (IPS). One of the challenges is to verify that the timing requirements of distributed operations are not violated. An example from robotized painting is to ensure proper order and timing of events on different nodes, e.g. when to open a valve on one node, relative to starting a motor on another node. Today, testing these issues is mainly done using manual, labor-intensive methods. This implies that the time from a software error is introduced, to the time that the error is discovered, can be quite long, if found at all. It also implies that when an error is reported, the developer(s) responsible for the erroneous code can be in a different context, working on something else, and thus it may take longer to resolve the problem.

By combining ideas from continuous integration (CI) [1] with constraint programming for test sequence generation, we can obtain significantly reduced round-trip time for many sophisticated tests that are only performed prior to release. We expect that these tests can be performed on daily basis, instead of only during release testing, resulting in faster error discovery. Through the development of a mathematical constraint model, we aim to generate sequences of events for driving the IPS into error states. Those error states are specified using specific constraint expressions that are confronted with a constraint model specifying the correct behavior of the IPS. Interestingly, the constraint model is also used to produce expected results of the IPS, that are used as test oracles.

II. Painting with a Robot

One of the key elements in robotized painting is the ability to perform precise triggers along a programmed path, meaning that the robot is able to turn on/off process equipment at correct points along the path. Process equipment used for painting include air and fluid (paint) control, valves, and other physical objects. Many of the processes involved in painting, are relatively slow compared to the movement of the robot arm. One example is the time it takes to turn on/off an airflow, which is in the range 100-200ms, due to filling/emptying hoses of air. For this reason, the IPS must be informed in advance about **when** to apply a specific process value. The robot controller does that by predicting the **time** it reaches the actual point where the spray pattern should be changed. This predicted time is then sent to the IPS together with the process value before reaching the point. Since the IPS receives both the process value to apply, and the time at which the value should be applied, the IPS can compensate for various process delays. With tight synchronization between processes, this technique makes it possible to inject precise triggers along a path, even for slow processes like air control.

III. System Overview

Figure 1 shows how our CI-based test system will be configured. In a real deployment, the robot controller is connected to the IPS master, while for running tests, the Test server bypasses the robot controller, connecting directly to the IPS master. All nodes in the network will be located at different physical locations on the robot or in a control cabinet.

In a paint deployment, the robot controller initiates a change in the spray pattern (brush) by sending a command to the IPS approximately 200ms before the robot arm arrives at the point where the new brush is to be applied. The command consists of a time and a brush number, $[t, b]$, as shown in Figure 1.

Using a *brush table*[1], the IPS master can do a lookup(b) to retrieve a list of process values to be applied for process P_i. The IPS master *may* also adjust the time t before each time/process value pair is sent to the node which handles the physical output. In Figure 1 this is shown as the $[t_i, P_i]$ message sent to Node 1. Internally on each node, the received

[1]The brush table is a lookup table where the brush number b is the key.

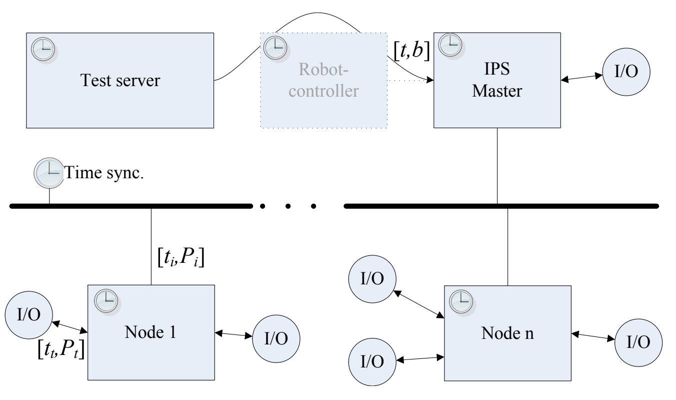


Fig. 1: IPS connection overview

$[t_i, P_i]$ can be additionally modified, $[t_i, P_i] \rightarrow [t_t, P_t]$, before the final P_t is added to a *trigqueue* to be executed at time t_t.

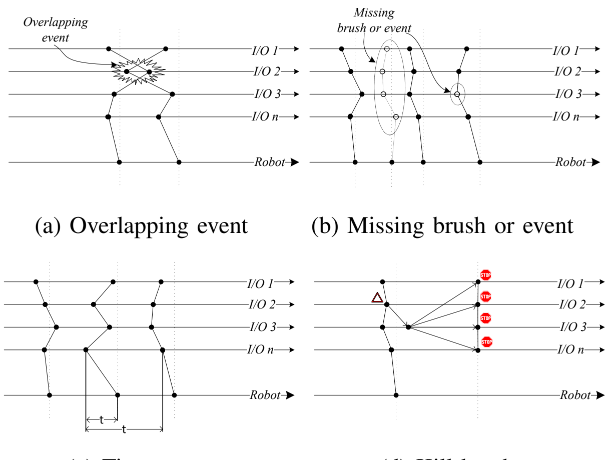
IV. Continuous Integration

CI has since Fowler [1] introduced it become very popular in the software development community. One example is Google, who has the intention of building and performing tests on every check-in to their source repository [2]. Inspired by the ideas of CI, and the advanced capabilities offered by constraint programming, we will build a full scale setup for running CI tests on the IPS system. A schematic overview is given in Figure 1. Note that the robot controller is replaced by a Test server running the constrain model as part of the test framework. In our setup we focus on testing the *process system*, so we regard the messages sent from the robot controller to be correct, and not part of the system under test. That is, the messages sent from a robot controller in a real deployment, is replaced by messages generated by our test framework running on the Test server.

V. Constraint Model

Constraint Programming (CP) is a well-known paradigm introduced twenty years ago to solve combinatorial problems in an efficient and flexible way. Typically, a CP model is composed of a set of variables V, a set of domains D and a set of constraints C. The goal of constraint resolution is to find a solution, i.e., an assignment of variables to values that belong to the domains and satisfy all the constraints. Finding solutions is the role of the underlying constraint solver which applies several filtering techniques to prune the search space formed by all the possible combinations. In practice, the constraint models that are developed to solve concrete and realistic testing problems usually contain complex control conditions (conditionals, disjunctions, recursions) and integrate dedicated and optimized search procedures [3]. In this work, our goal is to develop an industry-strength constraint model able to generate test sequences that to push the system towards error states.

As we saw in the example in Section III, there can be many adjustments/modifications on the initial time t sent from the robot controller, to the actual time t_t, where the process value is applied. The main focus of the constraint model we intend to develop is summarized in the following list:



(a) Overlapping event (b) Missing brush or event

(c) Time error (d) Kill brush

Fig. 2: Main error scenarios for the constraint model. Horizontal lines indicate time. A black dot corresponds to an *event*. A *brush* is visualized by black dots connected with lines.

1) **Overlapping events:** A process value for a brush event may get a time *before* an existing event in the *trigqueue*, see Figure 2a.
2) **Missing brush/event:** In a burst of brushes, verify that no brush-sets are lost, and that each brush is complete, i.e. no events are lost, see Figure 2b.
3) **Check of timing:** Verify the time between events, and between brushes, see Figure 2c.
4) **Kill brush:** Generate brush with error and verify graceful and correct shutdown, see Figure 2d.

VI. Conclusions

ABB has a six month development cycle on the IPS, with a new version released twice a year. Due to the complexity and high demand of manual labour to perform tests, the integration tests of the IPS is performed at the end of each development cycle. A consequence of this is that complex timing related errors that can't be detected by use of other test methods early in the development phase, will be detected very long time after the error is introduced.

By use of CI in combination CP based testing techniques, the time from a source code change to integration testing will be reduced from close to six months to less than a day.

References

[1] M. Fowler and M. Foemmel, "Continuous integration," 2006. [Online]. Available: http://martinfowler.com/articles/continuousIntegration.html

[2] J. Penix, "Large-scale test automation in the cloud (invited industrial talk)," in *Proc. of the 2012 Int'l Conf. on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 1122–1122. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337370

[3] A. Gotlieb, "TCAS Software Verification using Constraint Programming," *The Knowledge Engineering Review*, vol. 27, no. 3, pp. 343–360, Sep. 2012.