Runtime Verification of the ARIAC competition: Can a robot be Agile and Safe at the same time?

Angelo Ferrando[a], Zeid Kootbally[b], Pavel Piliptchak[c], Rafael C. Cardoso[a],
Craig Schlenoff[c] and Michael Fisher[a]

[a]*The University of Manchester, UK*
[b]*University of Southern California, USA*
[c]*National Institute of Standards and Technology, Gaithersburg, USA*

Abstract

ARIAC (Agile Robotics for Industrial Automation Competition) is a robotic competition which aims to advance robotic agility in industry. Participants in this competition are required to implement a robot control system to overcome agility challenges in a simulated environment. ARIAC comes with a set of score metrics to evaluate the performance of each control system during task execution. In this paper we show how such task-oriented evaluation can be problematic and how the addition of runtime monitors to verify properties given in ISO/TS safety standards can help in reducing the resulting reality gap.

Keywords

Runtime Verification, ARIAC competition, Agile robotics

1. Introduction

Robots are used in industry especially for repetitive tasks. Unlike human beings, robots do not get bored of performing the same task multiple times and can work in non-ideal environments, such as off-shore platforms, radioactive sites, and so on. However, robots are not good at adapting and changing depending on the necessities. While the transition from one task to another may be straightforward for a human, it is usually not the case for industrial robots which might have been designed and built for achieving one and only once specific task. Unfortunately, small to medium enterprises (SMEs) may need such level of dynamism. Consider the example of a manufacturing industry producing two different items, *item1* and *item2*. The company consists of only two robots, one to build *item1* and the other to build *item2*. Supposing the production rate is determined by the items in demand; it may happen there is no demand for *item1* and a high demand for *item2*. In such scenario, the product line for *item1* would stop, while the other one would be overloaded. The ideal solution in this scenario would be to use the unused product line to produce *item2*. This changeover requires the robot specialized in *item1* to start producing *item2*. In other words, this robot needs to be agile. In manufacturing terms, agility refers to the idea of responding effectively to changing customer needs in a volatile marketplace by handling product variety and by introducing new products quickly [1, 2].
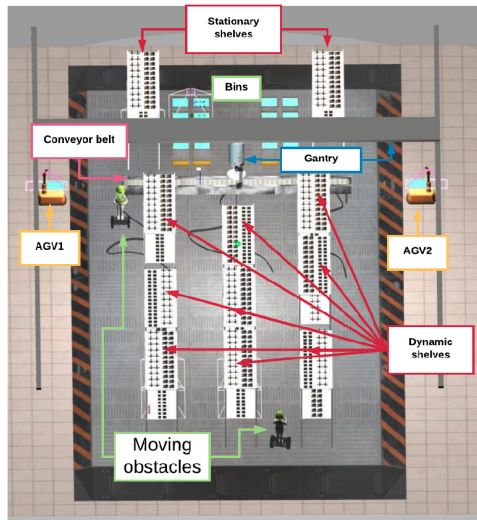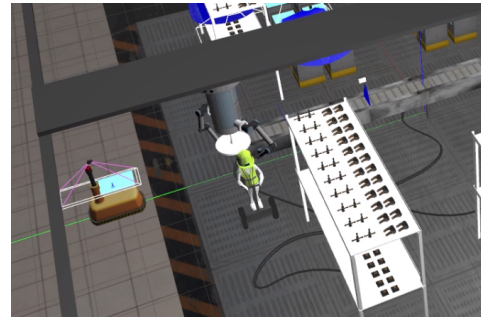
Figure 1: The ARIAC 2020 environment.

Figure 2: Robot about to collide with a human.

2. The ARIAC Competition

Organised by the National Institute of Standards and Technology (NIST, https://www.nist.gov/) since 2017, the Agile Robotics for Industrial Applications Competition [3] (ARIAC, https://www.challenge. gov/challenge/ariac/) is an annual event which brings together researchers and practitioners to tackle challenges that industry is facing. The main goal of ARIAC is to test the agility of industrial robot systems and to enable industrial robots on shop floors to be more productive, more autonomous, and to require less time from shop floor workers. In ARIAC, agility is defined broadly to address: (i) task failure identification and recovery by robots, (ii) automated planning to minimize (or eliminate) the up-front robot programming time when a new task is introduced, and (iii) fixtureless environment, where robots can sense the environment and perform tasks on parts that are not in predefined locations.

The competition participants are required to develop a robot control system for a gantry robot in order to perform kitting in a simulated environment. Gazebo (http://gazebosim.org/), which is an open source robotics simulation environment, is used as the testing platform and the Robot Operating System (https://www.ros.org/) (ROS), which is an open source set of software libraries and tools, is used to define the interfaces to the simulation system.

Figure 1 depicts the simulated environment where the ARIAC competition takes place. The robot used in ARIAC 2020 is of gantry type and can move in the simulated environment to interact with objects in order to perform kitting tasks. A kit is an order for specific items, which can be found on shelves, on the conveyor belt, and in bins. The robot builds kits by picking up all the required items and placing them into one of the two trays located on the Automated Guided Vehicles (AGVs). When an order is completed, the AGV delivers the kit and a final score is given to the participants' systems. The final score takes into account many aspects, such as the type/color of the selected item matches the type/color required by the order; the accuracy of products' pose in the tray; and the time taken by the control system to complete a kit (measured in simulation seconds).

Other metrics are used to determine the quality and performance of each participant's robot controller. However, except for some limited and manually coded control, there is no evaluation of how the robot completed the orders. The score considers if the right item has been brought to the delivery

station, but does not take into account how the robot actually builds the kits. For instance, during kitting, the robot may get very close to people in the environment or even hit them, as seen in Figure 2. Safety distance was not monitored in ARIAC 2020 and collisions with humans could only be reported by ARIAC organisers after the fact. A participant's control system may put the robot in these unsafe situations and still get the highest score. As can be seen, a balance between performance and safety of a robotic system has to be implemented in ARIAC. Above all, there is a need to reduce the reality gap between such implementations and the real world.

3. Safety through Runtime Verification

One way to solve this problem is to give importance to such safety aspects while evaluating a robot controller. To obtain this, we added runtime monitors to check the robot controller's behaviour at runtime. A runtime monitor is a component which observes a running system, and checks if the observed behaviour satisfies the expected behaviour through formal properties. The resulting verification process is called Runtime Verification (RV). With respect to other verification techniques, RV has the advantage of focusing on the current system execution, rather than statically trying to generate all possible ones; this makes it a lightweight approach which can be even applied to large and complex scenarios.

The main reason we chose RV is that it can be applied to black-box scenarios. Since the monitor only needs to observe how the robot behaves in the simulated environment, it does not need to know how the robot takes its decisions under the hood. Consequently, participants' source code is not required and the monitors are totally independent from the robot controller's implementation.

As a proof of concept, we developed a simple monitor to verify that the robot satisfies the safety distance from the simulated humans. ISO/TS 15066:2016 - "Robots and robotic devices — Collaborative robots" addresses the safety issue of robot speed and separation monitoring [4]. ISO/TS 15066:2016 specifies that the minimum allowable distance d_{min} between a robot and a human is

$$d_{min} = k_H(t_1 + t_2) + k_R t_1 + B + \delta \tag{1}$$

t_1 is the maximum time between the actuation of the sensing function and the output signal switching devices to the off state, t_2 is the maximum response time of the machine (i.e., the time required to stop the machine), δ is an additional distance, based on the expected intrusion toward the critical zone prior to actuation of the protective equipment, k_H is the speed of the intruding human, k_R is the speed of the robot, and B is the Euclidean distance required to bring the robot to a safe, controlled stop. We used Equation 1 to synthesise a monitor to verify at runtime that

"*the distance between the robot and the human operator is always greater than or equal to d_{min}*"

Since ARIAC is implemented in ROS, we used ROSMonitoring[1] [5] to deploy our monitors in the system. ROSMonitoring is a portable and formalism-agnostic runtime verification framework; it allows the definition of formal properties with any formalism of choice, and it generates monitors capable of intercepting ROS messages. In our case study, the information items of interest are the robot and human operator speed, and the current distance between the robot and the human. All other parameters of Equation (1) are known at the design time and do not need to be observed.

As mentioned previously, the monitor observes the system while the latter is running. Since the system is implemented in ROS, the monitor observes the messages exchanged among the nodes[2] in the system. The monitor is automatically synthesised by ROSMonitoring, but it requires knowing which messages have to be observed and which property has to be verified.

---

[1]https://github.com/autonomy-and-verification-uol/ROSMonitoring
[2]ROS is node-based; a robot can be composed of different nodes, and the nodes communicate through message passing.

Regarding the messages, in ARIAC there was no explicit representation of robot and human speed, or their distance. More specifically, this information was available in the simulation, but there were no explicit messages exchanged. Since ROSMonitoring monitors intercept messages, messages related to this information were added. A ROS node was created to keep track of where the robot and the human operators are, and at what speed they are moving. This node then publishes this information at a custom rate, which has been fixed to 100 Hz in this specific scenario (100 messages containing information about the robot and human operators are published per second). The resulting message represents a snapshot of the system, and can be intercepted by a monitor to verify the satisfaction of properties.

Previously, we reported the property of our interest in natural language; since ROSMonitoring natively supports RML (Runtime Monitoring Language[3]) [7] to describe formal properties, we formalised our property as a RML specification:

```
gteq_dmin matches {topic:'snapshot', human_operator_speed:h_speed, robot_speed:r_speed,
    distance_robot_human_operator:dist_h_r} with dist_h_r >= (h_speed * (1.0 + 1.5) + r_speed *
    1.0 + 0.0 + 2.6);
Main = (gteq_dmin)*;
```

The first line defines the set of expected events. `gteq_dmin` is the name of the set, `matches` is the pattern definition to which events belong to. Inside the curly brackets we discern the information contained in the event (the ROS message), such as the name of the message `snapshot` (the message topic), and labelled values reporting the human speed, robot speed, and distance between robot and human, respectively. Then, after the `with` keyword, we define the constraint that has to be satisfied by the event; namely, the distance between the robot and the human (`dist_h_r`) has to be greater than or equal to the minimum distance, according to equation (1), with t_1 = 1.0 [sec], t_2 = 1.5 [sec], B = 0.0 [m], and δ = 2.6 [m], deriving from the robot specifics. The second line instead denotes the body of the property, which in this specific scenario consists in observing a sequence of events[4] matching `gteq_dmin`.

The resulting monitor observes events characterising snapshots of the system, and verifies that each event satisfies the distance constrained derived by ISO/TS 15066:2016. If an event fails such a constraint, which means it does not belong to the event set denoted by `gteq_dmin`, an error is raised by the monitor, since an inconsistent event has been observed. In Figure 2 we reported a screenshot obtained with a sample robot controller. Figure 2 shows a situation where the robot is about to hit a human operator; in this situation, the monitor detects this violation of ISO/TS 15066:2016, and throws and logs an error.

4. Conclusions and Future Work

We have briefly presented ARIAC, and how it is used for advancing robotic technologies in the context of agility. We recognised a limitation in evaluating robotic controllers by only focusing on task completion, and how such limitation causes a reality gap. We showed how such reality gap can be reduced by adding runtime monitors verifying the system behaviour at runtime. In particular, we focused on an initial case study where a safety property given in ISO/TS 15066:2016 is used to synthesise a monitor. For future work, we aim to synthesise additional monitors deriving from other relevant standards, and integrate the resulting RV process in the score evaluation of future ARIAC competitions.

---

[3]RML is more expressive than LTL [6] and it can specify regular, context-free and context-sensitive trace languages.

[4]The * in RML has the same meaning of the * in regular expressions.

Acknowledgments

References

[1] P. Lindbergh, Strategic Manufacturing Management: A Proactive Approach, International Journal of Operations and Production Management 10 (1990) 94–106.

[2] H. Sharafi, Z. Zhang, A Method for Achieving Agility in Manufacturing Organisations: An Introduction, International Journal of Production Economics 62 (1999) 7–22.

[3] W. Harrison, A. Downs, C. Schlenoff, The agile robotics for industrial automation competition, AI Mag. 39 (2018) 73–76. URL: https://doi.org/10.1609/aimag.v39i4.2795. doi:10.1609/aimag.v39i4.2795.

[4] J. A. Marvel, Performance metrics of speed and separation monitoring in shared workspaces, IEEE Transactions on Automation Science and Engineering 10 (2013) 405–414.

[5] A. Ferrando, R. C. Cardoso, M. Fisher, D. Ancona, L. Franceschini, V. Mascardi, Rosmonitoring: a runtime verification framework for ros, in: Towards Autonomous Robotic Systems Conference (TAROS), 2020.

[6] D. Ancona, A. Ferrando, V. Mascardi, Comparing trace expressions and linear temporal logic for runtime verification, in: E. Ábrahám, M. M. Bonsangue, E. B. Johnsen (Eds.), Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday, volume 9660 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 47–64. URL: https://doi.org/10.1007/978-3-319-30734-3_6. doi:10.1007/978-3-319-30734-3_6.

[7] L. Franceschini, RML: Runtime Monitoring Language, Ph.D. thesis, DIBRIS - University of Genova, March 2020. URL: http://hdl.handle.net/11567/1001856.