Proceedings of the 2018 International Mechanical Engineering Congress and Exposition
IMECE2018
November 9-15, 2018, Pittsburgh, PA USA

IMECE2018-87686

VIRTUAL EXPERIMENTAL INVESTIGATION FOR INDUSTRIAL ROBOTICS IN GAZEBO ENVIRONMENT

Murat Aksu
National Institute of Standards and Technology
Gaithersburg, Maryland USA

John L. Michaloski
National Institute of Standards and Technology
Gaithersburg, Maryland USA

Frederick M. Proctor
National Institute of Standards and Technology
Gaithersburg, Maryland USA

ABSTRACT

Measuring the agility performance of the industrial robots as they are performing in unstructured and dynamic environments is a thought-provoking research topic. This paper investigates the development of industrial robotic simulation algorithms for the effective application of robots in those changing environments. The distributed framework for this investigation is the Robot Operating System (ROS) which is extensively used in robotic applications. ROS-Industrial (ROS I), which extends the capabilities of ROS to manufacturing, allows us to interoperate between industrial robots, sensors, communication buses and other kinds of automation tools. Gazebo is used as the open-source 3D simulator to design a virtual industrial robotic system, which is a prevailing tool as a node in the ROS environment. An effort is underway to replicate the in-house experimental robotic kitting lab with a graphical physics simulation that can be shared worldwide. This graphical physics simulation is not tied to a specific robotic control system. An experimental approach will be presented detailing the issues related to a physics based simulation of kitting with multiple collaborative robots, multiple tools, parts, tool changers, safety system, and sensors. In this realm, the ability for the simulation environment to encompass the current system as well as additional more complex sensors and actuators will be discussed. To make this simulation environment more realistic, Gaussian noise will be introduced to the data generated by virtual sensors. We expect that this experimental approach will be a seamless way for users to verify and validate their control systems even if they do not have a physical robot at their facilities.

INTRODUCTION

Industrial robots have traditionally been applied to automate tasks that are dirty, dull, or dangerous, and have been a key driver of continued productivity growth in high-volume applications such as automotive and electronics manufacturing. Worldwide, there were about 1.8 million installed industrial robots at the end of 2016, with a 10 % annual growth rate since 2010 [1]. In these high-volume applications, long up-front programming times are acceptable. However, as manufacturing requires more flexibility to support quickly changing product requirements in a high-mix, low-volume environment, robots need to become more agile. Agility in this context refers to the ability of a robot to be rapidly re-tasked for new activities without being taken offline for programming, the ability of a robot to recover from errors or uncertainty in the environment, and the ability to move applications between robots from different vendors without the need for program translation.

We have focused on how to represent the knowledge needed to achieve robot agility, the system architecture and component integration, planning, sensing and control, and how to measure the agility performance of robotic systems [2]. In support of this work, we have relied on simulation as a tool for the development of robotics systems to find ways to make better-informed decisions. In our laboratory, simulation allows tests to be run without contending for scarce time on physical robots, and to conduct tests safely without risk to damaging robots and tooling or injuring people. Simulation has also been used as the basis for competitions on robot agility, enabling competitors to practice at their facilities and compete in a controlled and instrumented environment. These simulations have used the Gazebo physics-based simulation package [3],

supplemented with the Robot Operating System (ROS) open-source framework for robot control [4].

There are several reported works aimed at simulation and robot control which are based on Gazebo and ROS. Aguero *et al.* [5] presented a Gazebo simulation platform for a cloud-hosted humanoid robot simulation with a wide range of sensors, controllers, and actuators to address the challenge of real-time task-oriented rescue robot competition for the Defense Advanced Research Projects Agency (DARPA) Virtual Robotics Challenge (VRC), and showed that simplified dynamics while maintaining sufficient accuracy is feasible. Swanson *et al.* [6] studied a Hardware-in-the-Loop (HIL) driving simulator that served as a training platform for driving performance, and presented a driver-in-the-loop simulation environment in which the driver and vehicle hardware components interacted with each other using ROS and Gazebo. The authors emphasized the effect of computer processor choices on decreasing the latency in the simulator and increasing the system fidelity. Fernandes *et al.* [7] analyzed the simulation of autonomous control of a robotic car using ROS and Gazebo, a research challenge of the Brazilian National Institute of Science and Technology on Embedded Critical Systems (INCT-SEC); however, very little information was provided on the driving simulation environment itself. Qian *et al.* [8] investigated the simulation of a robotic arm for manipulating objects by building a model of a pick-and-place robot with seven degrees of freedom, and demonstrated methods to implement robot control in a short period of time using ROS and Gazebo.

Unlike the research reported here, most of the simulation studies in the literature deal with non-industrial robotics applications. Some of the novel contributions presented in our research are a standard for sending commands and receiving status between an industrial robot and controller, a method of providing noisy sensor information to the object recognition system in our laboratory, and an overall effort to replicate the entire NIST agility laboratory with physics-based simulation to make it available to external collaborators allowing them to test their algorithms worldwide. The following sections will provide some background in the research, and describe how physics-based simulation of system components has helped in the research efforts.

Knowledge Representation

To help automate the planning of robot activities, a model of the robot's attributes, capabilities, and environment is needed. IEEE 1872, the Core Ontology for Robotics and Automation (CORA), is a standard for representing this information [9]. CORA provides definitions for general concepts for robotics, to support automated reasoning about robot activities, and as the basis for exchanging information about robotics.

Supplementing CORA, the authors have developed a messaging language for sending commands to robots and receiving real-time status from them. This messaging language, the Canonical Robot Command Language (CRCL) [10], is an eXtensible Markup Language (XML) Schema Definition (XSD) for information used to integrate and task robots independent of their internal programming language. Command message content includes:

- setting units, speeds, accelerations, and tolerances,
- setting robot parameters,
- performing Cartesian motions,
- performing joint-level motions,
- operating an end effector,
- configuring status reports,
- getting a status report immediately,
- displaying messages, and
- pausing or stopping motion.

Planning

A hallmark of agile robot systems is their ability to automatically plan and replan their activities in a dynamic and changing environment. To test this ability, the authors have used the Planning Domain Definition Language (PDDL) [11] as a source for specifying planning test inputs and results. PDDL is a language for encoding information needed for general planning problems, such as vehicle routing [11] and robot assembly [12]. Planners that use PDDL refer to models of objects, predicates about objects that can be true or false, the initial state and goal state of the world, and actions that are available to move objects and transform the initial state of the world to its desired goal state. These are placed into a domain file (for predicates and actions), and a problem file (for objects and world states). Given these files, a PDDL planner generates a series of actions to solve the planning problem. PDDL thus provides a standard way to measure the performance of planning algorithms in terms of time taken, memory used, quality of plan, or other metrics.

Figure 1 shows an excerpt from a PDDL domain file for a kitting application. The `:types` are tags for the resources referenced by the problem specification and the resulting plan file. The `:predicates` identify tags for resources whose state will be queried when evaluating the predicates. Predicate definitions are not shown, but are essentially lists of conditions to be evaluated by the application that executes the plan.

```
(define (domain kitting-domain)
  (:types
  EndEffector
  EndEffectorChangingStation
  EndEffectorHolder
  Kit
  KitTray
  LargeBoxWithEmptyKitTrays
  LargeBoxWithKits
  Part
  PartsTray
  Robot
  StockKeepingUnit
  WorkTable)
(:predicates
  ; part is held by endeffector
  (part-has-physicalLocation-refObject-endEffector
```

```
    ?part - Part ?endeffector - EndEffector)
  ; parts tray contains part
  (partsVessel-has-part ?partstray - PartsTray
    ?part - Part) ... )
```

Figure 1. Sample PDDL domain for a robotic kitting application.

Figure 2 shows an excerpt from a PDDL problem file that is to be solved by a planner, to take the kitting application from a stating initial state `:init` to ending goal state `:goal`. The planner's objective is to determine a series of actions that take the system from the initial state to the final state as efficiently as possible.

```
(define (problem kitting-problem)
  (:domain kitting-domain)
  (:objects
    robot_1 - Robot ... )
  (:init
    (endEffectorHolder-has-endEffector
tray_gripper_holder part_gripper)
    (partsVessel-has-part  part_small_gear_tray
small_gear_1) ... )
  (:goal (and
    (= (final-quantity-of-parts-in-kit kit_s2b2) 4)
    (= (quantity-of-parts-in-kit
      sku_part_small_gear kit_s2b2)
       (capacity-of-parts-in-kit
      sku_part_small_gear kit_s2b2)) ) ) )
```

Figure 2. Sample PDDL domain for a robotic kitting application.

Figure 3 is a sample plan showing actions for locating, picking, and placing a part in one step of an overall larger plan for kitting.

```
(look-for-part robot_1 large_gear_1
  sku_part_large_gear kit_s2b2 part_gripper)
(set-grasp robot_1 large_gear_1
  sku_part_large_gear part_gripper)
(take-part robot_1 large_gear_1 sku_part_large_gear
  part_large_gear_tray part_gripper kit_s2b2)
(look-for-slot robot_1 large_gear_1
  sku_part_large_gear kit_s2b2 part_gripper)
(place-part robot_1 large_gear_1 sku_part_large_gear
  kit_s2b2 part_gripper work_table_1
  part_medium_gear_tray)
```

Figure 3. Sample PDDL plan.

As noted by Mösenlechner and Beetz [13], the logical PDDL model of actions that trigger known changes in state is not well suited to autonomous robot systems, where the outcome of actions may not be predictable. Another problem is that the high-level nature of PDDL states for the initial conditions, goal conditions, preconditions, and postconditions do not incorporate finer-grained detail whose slight variation could lead to different choices of actions. As described in the next section, the first shortcoming of PDDL has been overcome by incorporating continuous replanning when actions do not result in the predicted outcome. The second shortcoming has been addressed to by choosing actions with parameters at a level of resolution low enough be adjusted through real-time sensor feedback from vision.

System Architecture

Work supporting CORA and CRCL has taken place in the Agility Performance of Robotics System (APRS) laboratory at the National Institute of Standards and Technology (NIST) [2]. The lab contains two industrial robots, a Fanuc LR-Mate 200iD and a Motoman SIA20F. The robots share tooling for open-and-close gripping and vacuum gripping. The primary application is kitting, where parts are moved from their initial location in storage trays to a final target arrangement in kit trays. Overhead cameras in the work volume are used to determine the location of parts, storage trays, and kit trays. This laboratory is shown in Figure 4.

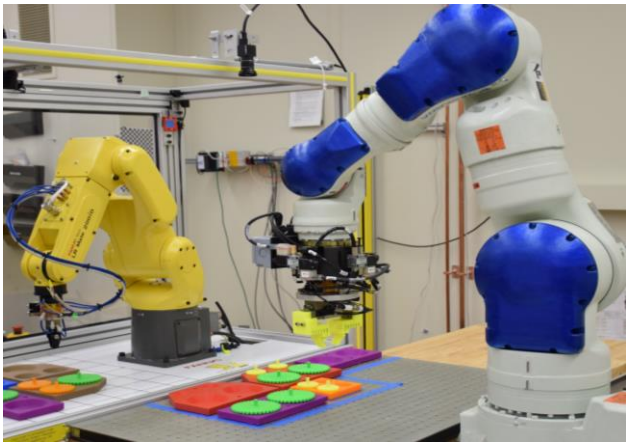

Figure 4. APRS Laboratory Workcell.

Figure 5 shows the APRS system architecture. The purpose of the system is to put together kits of parts based on a request composed by the operator, shown at the top of the figure. The resulting PDDL goal is a set of kits and their contents of parts. This goal is sent to the PDDL planner, which consults the definitions of actions, preconditions, and postconditions in the kitting problem domain and determines a feasible sequence of actions that achieve the kitting goal. These actions are sent to the PDDL executor, which fills in actual values for part and kit locations based on the current state in the world model database. This database is continually updated with the locations of parts as measured by the object recognition system. After the actions are instantiated, the resulting CRCL program is sent to the CRCL client for execution. This client steps through the program, sending messages to the robots and grippers and monitoring execution status until the program has completed. Any failures are reported by the client, which triggers replanning if possible until the kitting request is fulfilled, or stopped due to unrecoverable problems.

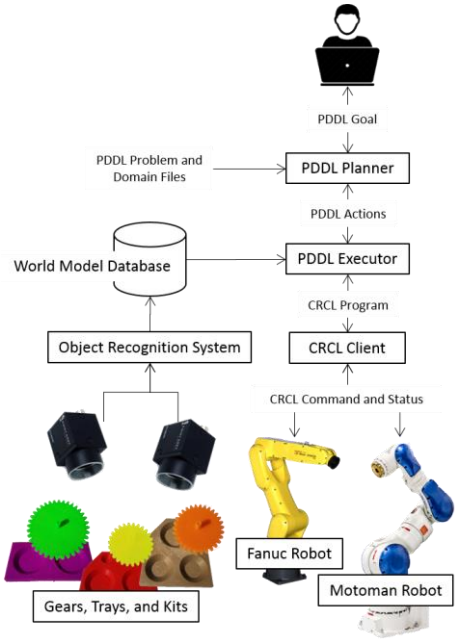

Figure 5. System Architecture.

Measuring Agility

To advance the state of robot agility, a series of competitions was organized that measure the effectiveness of planning systems to rapidly re-task robots without the need for human intervention. Such tasking includes the ability of robots to recover from errors such as dropped parts, the ability of perception systems to identify problems, and the ability of manipulation systems to reposition objects for better error recovery. The Agile Robotics for Industrial Automation Competition (ARIAC) is sponsored by NIST in collaboration with the Open Source Robotics Foundation (OSRF), developers of the Robot Operating System (ROS) [4] and the Gazebo physics-based simulation environment.

Competitions are organized around a kitting application, where robots are given the task to move objects from a set of trays to a goal kit. Virtual sensors for determining object locations include cameras, beam break detectors, laser range scanners, and laser line curtains. Teams are given the flexibility to choose which sensors to use. Costs are associated with the sensors selected and factored into the scoring metrics.

THIS RESEARCH

The goal of this research is to use a physics-based simulation to stand in for the APRS laboratory environment, and use the uncertainty in part location and activity completion to test the ability of the planning system to recover from failures. The intent of the simulation is to provide the following enhancements:

- the ability to test strategies for sensor-based recovery from errors in a repeatable environment;
- enabling hybrid real-virtual operation, where one real robot and camera and one simulated robot and camera can be used simultaneously;
- and to provide additional implementations of CRCL-conforming robots to validate this specification.

Gazebo was selected as the simulation environment [3]. Gazebo provides realistic visual rendering of physical scenes, linked to one of a set of configurable physics engines that update the state of objects in the simulated world according to physics principles such as friction, inertia, and gravity.

Figure 6 shows a Gazebo visualization of the physics-based simulation of a kitting activity used in the ARIAC competition.

Figure 6. Gazebo Simulation of Robot Kitting in the ARIAC Competition.

Simulating the APRS Environment

The NIST agility kitting robot control laboratory was ported to a physics-based simulation environment. Simulation can be kinematic or physics-based. Kinematic simulation uses visualization of the sequence of operations to verify correctness. Physics-based simulation models the physical elements' interactions and collisions and the effects of physical properties such as gravity, friction, and inertia. The intent of physics-based simulation is to study control and sensing, reveal inaccuracies, and verify correctness. For example, placing of a "gear" into a slot holder in a visualization could overlay two images at the bottom of the slot (the gear and holder) without repercussions. However, in the case of physics-based simulation, the gear would "bounce" out of the slot as it is physically impossible for a solid object to atomically combine with another solid object.

Figure 7 shows the Gazebo physics-based simulation of the agility lab. The simulation modeling includes the two robots, the agility lab physical space, which consists of the tables, the walls, and finally the gear, kitting, and tray objects. The two overhead vision cameras and enclosing safety system are not visualized in the simulation; however, the camera images are simulated by a Gazebo plugin that can be used to test the actual robot planning and control system. Robotiq two-finger gripper models available on the Internet were used, saving the effort of

modeling the custom 3D-printed fingers used in the lab. Grippers and the robot base location are handled in a kinematic ring (discussed later), which makes substitution of different grippers and relocation of the robots in the agility lab done with a different transform.

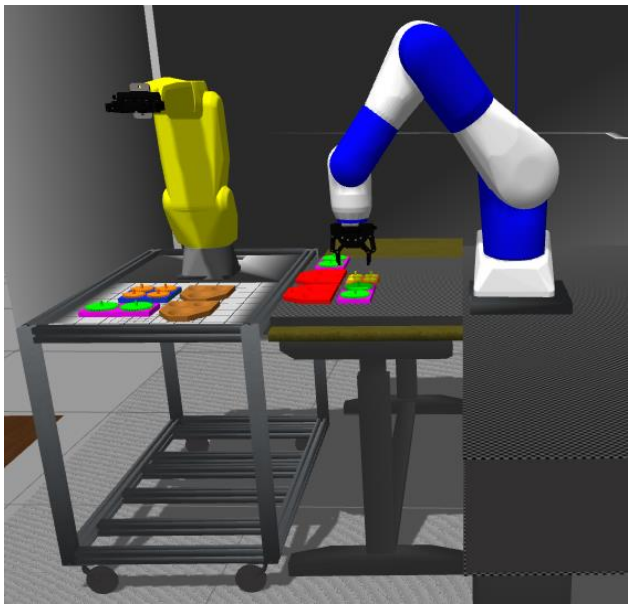

Figure 7. Agility Lab Physics-Based Simulation

Modeled objects in the simulation environment.

The Gazebo physics-based simulation relies on either Gazebo Simulation Description Format (SDF) or ROS Universal Robotic Description Format (URDF) to model robots and other world elements. Both SDF and URDF are XML file formats that describe objects and environments for robot simulators, visualization, and control [14, 15]. Both URDF and SDF include mechanisms to describe links, joints, kinematic chain relationships, the limits and capabilities of the joints, obstacle volume of a link, and a visual representation of each link. SDF includes mechanisms to describe physical elements such as mass, inertial frame, gravity interaction, among a multitude of physics descriptors. Often, ROS URDF was converted into Gazebo SDF format. Thus, in the physics-based agility simulation, the robot and gripper, the kitting objects, and the agility lab were defined with either SDF or URDF.

For example, the agility simulation includes gears, holders, and kits that were originally modeled for 3D printing, but the Computer Aided Design models were translated into STL format (i.e., stereolithography), and modified to satisfy the Gazebo world (for example, adding mass and inertial frames while adjusting the origin coordinate frame).

Robot Control

The agility robot control application domain is multi-axis, coordinated motion control. In addition, process control is necessary to handle input/output, and auxiliary equipment. Representative robot controller applications include manipulation, assembly, and collaboration.

The robot controller software modules include: (1) Joint control, which performs servo control of axis motion by transforming incoming motion goal points into set-points for the corresponding actuators, (2) Cartesian motion planning, which coordinates the motions of an individual joint by transforming an incoming motion segment specification into a sequence of equi-time-spaced setpoints for the coordinated axes, and (3) Task Coordinator modules, which sequence operations and coordinate the various motion, sensing, and event-driven control processes. Overall, the robot control architecture was based on open-source components. Its source code is available for public use.

The inverse and forward kinematics use OpenRAVE IKFast software to solve the forward and inverse kinematics [16]. In general, IKFast can analyze the robot kinematics, solve the kinematics equations, and write the solution to a C++ file.

Because the NIST agility robot motion control relies on Cartesian based straight-line motion and assumes a collision-free industrial environment, trajectory planning is done with "Gotraj" [17]. Gotraj computes a smooth trajectory based on either time or dynamical properties (velocity, acceleration, and jerk). Gotraj assumes a final velocity of zero. Users can append poses onto the Gotraj motion queue that will result in additional trajectories. Gotraj supports a "stop" motion directive that will generate a trajectory which will stop as soon as possible given the motion dynamical properties (velocity, acceleration, and jerk).

Another common robotic concept that was required to handle different robots, grippers, and robot locations within the simulation world was the kinematic ring. Thus, any CRCL commanded robot Cartesian motion in world coordinates is then expressed in terms of kinematic ring made up of a series of homogeneous matrix transforms from the base robot frame, including the robot transformation, and robot gripper transform. With this mechanism, it was easy to shift between world and robot coordinate frames. This is important as although the robot and kitting objects are modeled precisely, the location of the robot base and the gripper tooling can vary, and kinematic rings offer a convenient scheme for resolving the variances.

Integration with CRCL. At NIST, kitting commands to the robot(s) are expressed in CRCL, which is a messaging language for controlling a robot that is executed by a low-level device robot controller. CRCL is an abstraction of the robot control capabilities that is fully defined in an XML schema. CRCL contains the ability to command robot joint, Cartesian position and gripper control. In addition, CRCL supports status streaming or service requested communication patterns. CRCL uses XML messages for communication, which typically uses a stream based Transmission Control Protocol (TCP) socket to communicate the XML.

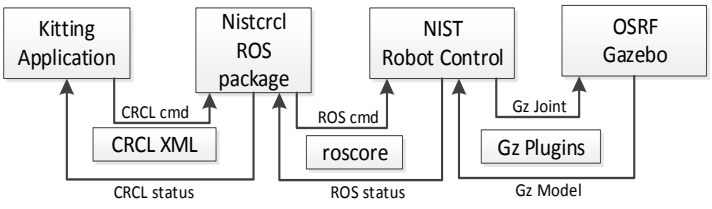

Figure 8. ROS Nistcrcl Package Architecture.

Figure 8 shows the ROS Nistcrcl package architecture used to handle CRCL communication in the physics-based robot agility simulation. The ROS Nistcrcl package is a ROS node that was developed to listen and broadcast CRCL XML command and status messages, while translating CRCL from/to ROS representation and communication ideology. The Nistcrcl ROS package uses several open-source software technologies to adapt CRCL into ROS. The CodeSynthesis XSD tool generates a C++ object model from XSD that is used to parse and serialize CRCL XML. CodeSynthesis relies on the Xerces XML parser. The Boost C++ library Asynchronous IO (Asio) was used to handle socket command and status communication with the CRCL client. The ROS message infrastructure (based on the Google protobuf open source communication scheme) was used to build "custom" ROS topics that encapsulated CRCL functionality. Two ROS custom messages were developed - one for CRCL command messages and one for CRCL status messages.

Gripping objects. In general, an end effector is the device at the end of a robotic arm, meant to interact with the environment. Kitting is concerned with grasping objects and relocating the objects. This can be done with a vacuum gripper, or a gripper with two or more fingers. We are interested in the case of using grippers to achieve object manipulation (i.e., grasping and releasing).

One gripper used for simulation was the Robotiq's 2-Finger Adaptive Robot Gripper. An open source ROS URDF description existed to describe the kinematics and visualization, which simplified implementation.

Another gripper used in the simulation was the NIST in-house 3D printed parallel jaw gripper. Grasping objects highlights the difficulty of physically modeling gripper and grasped object interaction, because explicit enumeration of friction, collision, and other dynamical behavior elements play an important role in grasp control. By comparison, in the real world, grasping generally ignores the interaction of the underlying physics.

Calibration

The APRS has four coordinate systems: the Fanuc and Motoman robots, and the two cameras located above their work volumes.

Calibration of camera coordinates to robot coordinates is done following a four-point registration procedure:

1. A marker object (e.g., a gear) is placed at a location in the work volume of one of the robots.
2. The object detection system's coordinates for the center of the marker object are recorded.
3. An operator guides the robot to the center of the marker object, and the robot controller's coordinates of this point are recorded.
4. This procedure is repeated for a total of four locations, placed near the corners of the work volume.
5. Offset transformations from object detection system coordinates to robot coordinates are computed at each of the four locations.

These offset transformations are determined only occasionally, when the robots or cameras are repositioned. Table 1 shows the results of four-point registration for the Motoman robot.

Table 1. Calibration Offsets from Object Recognition System to Motoman Robot.

X (mm)	Y (mm)	X offset (mm)	Y offset (mm)
535.9	157.7	0.0	5.0
723.1	-400.3	-9.0	-7.0
523.8	-106.7	-4.0	0.0
747.3	155.4	5.0	0.0

During operation, object locations from the object detection system are interpolated between these four registration transforms, resulting in a location of the object in robot coordinates. This interpolation is done by weighting each of the contributions of the registration points by their inverse distance to the point whose offsets are to be interpolated, normalized by the sum of the inverse distances. The problematic infinities for the values of the inverse distances at the registration points themselves are avoided by algebraically changing the formula to that in Eqn. (1):

$$\delta_p = \frac{\sum_{i=1}^{n} \delta_i \prod_{j \neq i}^{n} d_j}{\sum_{i=1}^{n} \prod_{j \neq i}^{n} d_j} \qquad (1)$$

where δ_p is the interpolated offset, the set of δ_i are the offsets at the n registration points, and set of d_j are the distances from the point to be interpolated to each of the registration points. This equation works for offset translation vectors as well as offset orientations, when orientations are represented as rotation vectors.

The four-point registration procedure is also used to determine the transformation between the Fanuc and Motoman robot coordinate systems. Using this procedure, a best-fit transform is computed following Horn's solution to the absolute orientation problem [18].

Inverse error compensation. It is desirable for the simulated behavior to match the real-world behavior, so that the four-point registration procedure need not be repeated in simulation and consequently require the control system to

switch between real and simulated operation. Therefore, the simulated object detection system needs to adjust its output with the inverse of the interpolated transform. Figure 9 shows a map of the magnitude of the compensating error throughout a region bounded roughly by the registration points.

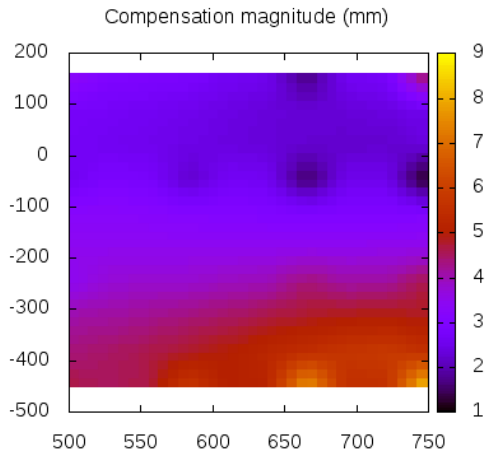

Figure 9. Inverse Error Compensation

The inverses of these positional compensation errors will be applied to the reporting of simulated object locations, as detailed in the following sections.

Object Recognition and Reporting

There are two methods for recognizing objects: ground truth with noise, and camera image generation with noise. For the first method, the computer vision system is not used, and its output (object locations) is simulated directly. For the second method, synthetic camera images are produced from the simulation, and fed to the computer vision system instead of images from real cameras. The goal of the simulation is to be able to provide realistic object data streams to the CRCL executor, without reconfiguring the executor based on whether real or simulated sensor data is used. Figure 10 shows view of the objects in the real object detection system.

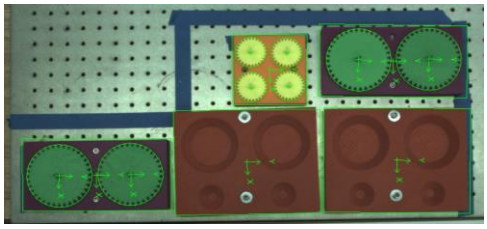

Figure 10. Objects Appearing in the Object Detection System.

The output characteristics of the object recognition system were first determined based on a test of its measurements of a single object. Figure 11 shows a plot of the X-Y positions of 1487 measurements of the position of a single object by the object recognition system. The figure shows clustering of points and does not suggest any well-known underlying distribution. For the purposes of simulating randomness consistent with the variation shown, a normal distribution was fit to the data, with a mean and standard deviation of $\mu = 416.65$ mm and $\sigma = 0.28$ mm for X, and $\mu = 342.50$ mm and $\sigma = 0.33$ mm for Y. The following sections describe how the simulation was developed so that it produces sensor data consistent with that produced by the actual sensing system.

Ground truth with noise. Gazebo provides the true locations of all objects in the world through a ROS topic updated at the simulation frequency, measured to be about 2 milliseconds. These ideal locations of the gears, kits, and trays are fed into an application that adjusts the poses of each object with noise representative of that measured by the real object detection system.

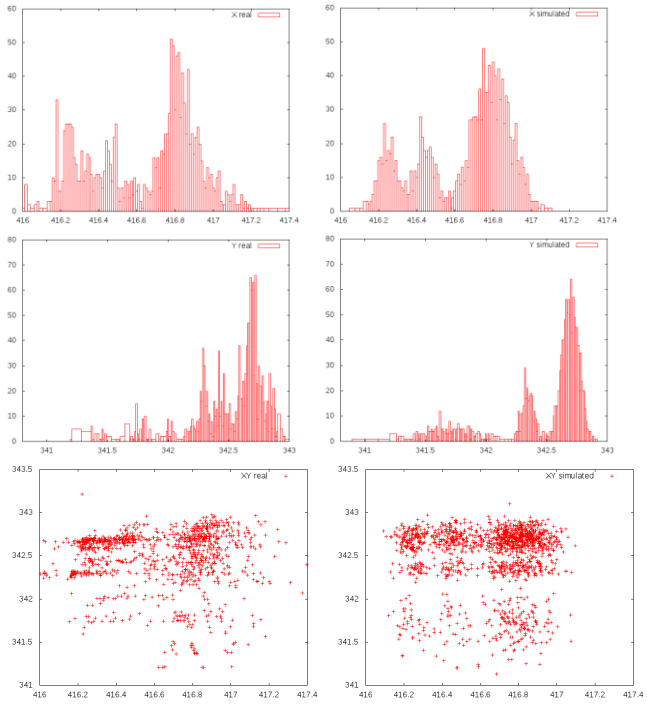

Figure 11. X-Y Location Variance from Object Recognition System, Real and Simulated. Left Images Are Real, Right Images Simulated.

The noise-adding application was customized so that it replicates the clustering exhibited by the real object detection system. For each of the X and Y distributions shown in Figure 11, three normal distributions were composed with means, standard deviations, and relative contributions that approximated the clusters. This empirical customization, while not perfect, replicates the behavior of the system to a degree that is visible to the CRCL executor and has the same qualitative influence.

Camera image generation with noise. With this method, the existing object recognition is used, and synthetic camera images are provided to it as if they originated from

actual cameras. Gazebo is equipped with a camera sensor plugin which publishes images on a ROS topic. The overhead physical camera was emulated by introducing noise to the data streams generated by the virtual camera to help making the synthetic camera images more realistic because the Gazebo camera sensor model views the virtual world flawlessly. Gazebo also provides a sensor noise model which can add Gaussian noise parameters to the virtual camera sensor [19]. To achieve the goal of making our world experiment closer to the realistic environment, we implemented these Gaussian noise parameters and used three different values for noise standard deviation: σ = 0.007, 0.09 and 0.3.

Figure 12 shows images retrieved from the sensor with varying levels of Gaussian noise. The image on the left depicts standard deviation of σ = 0.003. In the central image, the virtual camera has standard deviation of σ = 0.03, which illustrates moderate level of noise. Finally, the image on the right represents the virtual camera with σ = 0.3 standard deviation, which shows high level of noise. A value of σ = 0.007 is considered to be reasonable for a decent digital camera [20].

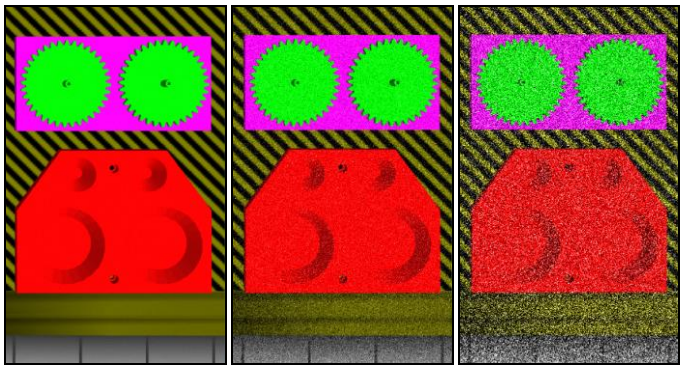

Figure 12. Image Data of the Virtual Camera Sensor with Low (on the left), Moderate (in the center), and High Level of Noise (on the right).

Sets of 1634 sequences of each of these three noisy camera streams were input to the object recognition system, which calculated three lists of the X-Y points. These are shown in Figure 13. Like Figure 11, these data showed normal-like distributions of the X and Y values, although the simulated images were much closer to a normal distribution and did not exhibit the clustering shown from real camera images. To more accurately replicate this clustering, the Gazebo noise model must be extended with a plug-in that allows for arbitrary customization, as noted in the description of future work in the concluding section.

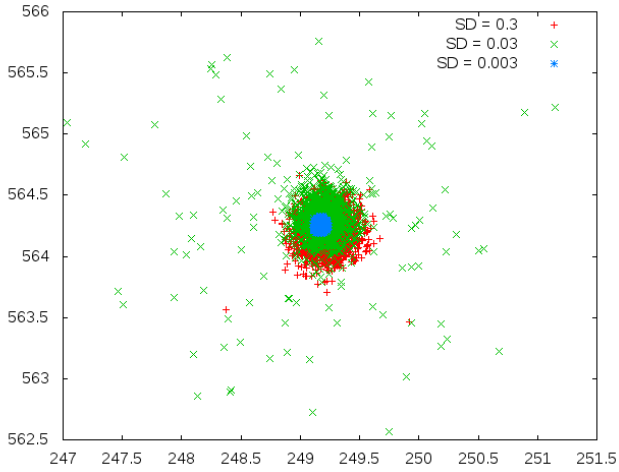

Figure 13. X-Y locations from the object recognition system for high (red), medium (green), and low (red) levels of simulated camera noise.

RESULTS AND CONCLUSIONS

This paper describes how the Gazebo physics-based simulation was applied to model the environment, objects, and robots in a laboratory used to measure the agility performance of robot systems. Simulation is an effective means that allows researchers to test their systems without tying up the actual robots, relieving the pressure on scarce resources and enabling safer and more repeatable testing. The goal was to replicate the noisy behavior of the real world in simulation, so that compensations and calibrations already built into the real planning and control system would function without modification. This necessitated performing effectively the inverses of the compensations and calibrations within the simulation. These apply to both the locations of the objects to be manipulated, and the poses of the simulated robots. Two methods of object pose adjustments were implemented, one that bypassed the object detection system and simply applied a noisy transform to the ideal poses, and one that included the object detection system and provided it with noisy camera images. Testing validated that the approaches achieved the intent, but could be improved with better registration of the simulated world with the physical world.

Toward the objective of testing strategies for sensor-based recovery from errors in a repeatable environment, the simulation is set up to allow either scripted configuration of initial testing conditions, or interactive placement of objects and robot starting locations. The integration of ROS with Gazebo allows for the timestamped logging of simulation states that can be compared between tests. Enabling hybrid real-virtual operation was achieved with minimal impact on the workcell configuration. CRCL interfaces and object detection reporting from the simulation followed the real robot protocols, with the selection of the real or virtual targets localized in a controller configuration that can be changed while the system is running. Another enhancement arose from the need to build a CRCL interface onto the Cartesian motion planner used with Gazebo.

This provided another validation test of CRCL and its command-state protocol. While no changes to the CRCL XSD were required, the additional independent testing pointing to some assumptions on how to start and stop motions between pick-and-place activities, which were clarified in the documentation.

The overall research contribution is the use of simulation to supplement validation testing of a standard for knowledge representation of industrial robot tasks, CRCL. The use of simulation allows for more repeatable testing, enabled by the world model state inspection and logging features of Gazebo and ROS. A unique aspect of the research is the dual method of providing noisy sensor information to the object recognition system, using both noisy camera images and noisy object states, that attempt to match the clustering effects shown by the real system.

Future work includes doing better registration, including more capabilities of the real testbed such as shared tooling and vacuum gripping, and building simulated cameras that better match the actual cameras used. Ultimately, the simulation will be made available to external collaborators, such as those participating in the ARIAC challenges, to allow them to test their algorithms prior to running on the actual hardware in the APRS.

Disclaimer: Certain commercial/open source software, hardware, and tools are identified in this paper to explain our research. Such identification does not imply recommendation or endorsement by the authors or NIST, nor does it imply that items identified are necessarily the best available for the purpose.

REFERENCES

1. International Federation of Robotics, *Executive Summary: World Robotics 2017 Industrial Robots*. 2018: ifr.org.
2. Kootbally, Z., et al., *Enabling robot agility in manufacturing kitting applications.* Integrated Computer-Aided Engineering, 2018(Preprint): p. 1-20.
3. Koenig, N. and A. Howard. *Design and use paradigms for Gazebo, an open-source multi-robot simulator*. in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. 2004. IEEE.
4. Martinez, A. and E. Fernández, *Learning ROS for robotics programming*. 2013: Packt Publishing Ltd.
5. Agüero, C.E., et al., *Inside the virtual robotics challenge: Simulating real-time robotic disaster response.* IEEE Transactions on Automation Science and Engineering, 2015. **12**(2): p. 494-506.
6. Swanson, K.S., et al. *Extending driving simulator capabilities toward hardware-in-the-loop testbeds and remote vehicle interfaces*. in *Intelligent Vehicles Symposium Workshops (IV Workshops), 2013 IEEE*. 2013. IEEE.
7. Fernandes, L.C., et al. *Intelligent robotic car for autonomous navigation: Platform and system architecture*. in *Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on*. 2012. IEEE.
8. Qian, W., et al. *Manipulation task simulation using ROS and Gazebo*. in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*. 2014. IEEE.
9. Schlenoff, C., et al. *An IEEE standard ontology for robotics and automation*. in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. 2012. IEEE.
10. Proctor, F., et al., *The Canonical Robot Command Language (CRCL).* Industrial Robot: An International Journal, 2016. **43**(5): p. 495-502.
11. Fox, M. and D. Long, *PDDL2. 1: An extension to PDDL for expressing temporal planning domains.* Journal of artificial intelligence research, 2003.
12. Guimarães, W.H.P., et al., *Analysis of automated planning applied to an assembly and disassembly robot system.* 2013.
13. Mösenlechner, L. and M. Beetz. *Using Physics-and Sensor-based Simulation for High-Fidelity Temporal Projection of Realistic Robot Behavior*. in *ICAPS*. 2009.
14. Meeussen, W., J. Hsu, and R. Diankov, *URDF-Unified Robot Description Format*.
15. OSRF. *SDF*. 2014 [cited 2018 April 18]; Available from: http://sdformat.org/spec.
16. Diankov, R., *Openrave, ik fast module, openrave documentation*. 2016.
17. Proctor, F. *go motion*. [cited 2018 April 18]; Available from: https://github.com/frederickproctor/gomotion.
18. Horn, B.K., *Closed-form solution of absolute orientation using unit quaternions.* JOSA A, 1987. **4**(4): p. 629-642.
19. Newman, W.S., *A systematic approach to learning robot programming with ROS*. 2017, Boca Raton: Chapman & Hall/CRC ©2017. 530 pages.
20. OSRF. *Sensor Noise Model*. 2014 [cited 2018 April 24]; Available from: http://gazebosim.org/tutorials?tut=sensor_noise&cat=sensors.