**Proceedings of the ASME 2019
International Mechanical Engineering Congress and Exposition
IMECE2019
November 11-14, 2019, Salt Lake City, UT, USA**

IMECE2019-11345

PHYSICS–BASED SIMULATION OF AGILE ROBOTIC SYSTEMS

Pavel Piliptchak[*], **Murat Aksu, Frederick M. Proctor, John L. Michaloski**
National Institute of Standards and Technology
Gaithersburg, MD

ABSTRACT

Development and testing of industrial robot environments is hampered by the limited availability of hardware resources. Simulations provide a more accessible and readily modifiable alternative to physical testing, but also require careful design to maximize their fidelity to the real system. In this paper, we describe new progress in building an entirely physics-driven simulation of the Agility Performance of Robotic Systems (APRS) Laboratory at the National Institute of Standards and Technology (NIST). To maximize the accuracy of physics-based interactions in this environment, we develop several general-purpose improvements to the simulation of parallel grippers and multi-object collisions. We use pick-and-place tasks from the APRS Laboratory as well as generic benchmarks to verify the performance of our simulation. We demonstrate that our proposed improvements result in more physically-consistent simulations compared to standard implementations, regardless of the choice of physics engine or simulation parameters.

Keywords: robotics, agility, grasping, pick-and-place, simulation

1 INTRODUCTION

Simulations are invaluable tools in developing industrial robotic systems. In situations where hardware resources are limited or costly to use, simulations can provide a test-bench for developing controllers, debugging software, and testing various work-space configurations [1]. One of the limiting factors to using a simulation in place of a real system is how well it replicates the physical dynamics of the real system. This discrepancy has inspired the development of new rigid-body physics engines that tackle specific problem areas, such as numerically stable contacts and joint articulations [2–4].

However, existing robotics simulators are not a one-size-fits-all solution. A significant amount of effort needs to be put into defining contact behavior, inertia matrices, friction coefficients, and numerical solver parameters to ensure acceptable performance [5]. This effort becomes increasingly laborious as the simulated environment becomes more complex.

In this work, we use the Robot Operating System (ROS) [6] and the Gazebo [7] robotics simulator to build an entirely physics-based simulation of the APRS Laboratory, an environment designed for evaluating the capabilities of robots performing industrial pick-and-place tasks [8]. This environment is fairly difficult to replicate with standard simulators due to the large number of small, low-mass objects that the robot must accurately manipulate. Along the course of building this simulation, we developed several practical and general-purpose improvements to simulating two important components of our pick-and-place task (namely parallel grippers and multi-object collisions).

This work presents the following contributions:

1. A complete simulation of the APRS Laboratory that integrates with existing APRS software
2. A simple controller for simulated parallel grippers that improves grasp stability
3. A method for simplifying contacts between many relatively stationary objects in simulation

[*]Contact author: pavel.piliptchak@nist.gov

In the following sections, we will (1) discuss relevant experiments in physics-based simulation at NIST and abroad; (2) describe the design of our APRS Laboratory simulation; (3) describe our two improvements mentioned above; and (4) present experiments verifying the benefits of these improvements.

2 RELATED WORK

Despite the popularity of Gazebo for robot simulation, there have been comparatively few publications that use it for physics-based grasping and manipulation—particularly in full-fledged, realistic environments. The most notable example is the Defense Advanced Research Projects Agency (DARPA) Virtual Robotics Challenge (VRC) which required grasping as part of several subtasks [9, 10]. Simulations have also been used for the dexterous manipulation of cloth [11], pick-and-place tasks [12], and dexterous reinforcement learning agents [13]. The robots in these simulations generally use high degree-of-freedom (DOF), anthropomorphic hands. They also generally handle palm-fitting objects rather than smaller objects that require fingertip-only grasps. In contrast, the APRS simulation robots use parallel grippers and manipulate peg-like objects with diameters that are under 1 cm.

Another angle of research has been verifying the physical accuracy of the physics engines themselves [14–16] with some works emphasizing grasping in particular [17, 18]. Often, this research seeks out general benchmarks that measure either: 1) how the engine's compute speed scales with simulation complexity; 2) how well the engine conserves energy/momentum; or 3) how far the engine deviates from some ground-truth behavior for a simple dynamical system. We verify the accuracy of the APRS simulation using similar methods (to be discussed in the Experiments section), but focus on benchmarks that are relevant to grasping and pick-and-place tasks. We do not make claims about the general accuracy of the tested physics engines.

NIST has also pursued research in verifying the physical accuracy of robot simulators [19, 20]. This earlier work focused on the simulation of the Talon Robot using the older Karma Physics Engine. The evaluation metrics used in this work were based primarily on mobility with limited discussion of grasping. In contrast, our work simulates a pick-and-place task using Gazebo's variant of the Open Dynamics Engine (ODE) [2, 5]. It focuses more finely on grasp stability and simulator performance.

3 SYSTEM OVERVIEW

In this section, we give a high-level overview of our APRS simulation. We will begin by reviewing the objectives and design of the physical APRS Laboratory. Following this, we will present the design of our simulation along with its interfaces to the existing APRS architecture.

APRS Architecture

The APRS Laboratory seeks to study and measure the "agility" of robotics systems, which widely refers to a system's retasking capability, robustness to failures, and interoperability [8]. As part of this research effort, the APRS Laboratory implements an archetypal agile robotics system that operates on a simple pick-and-place kitting task.

The system consists of a Fanuc LR-Mate 200iD and a Motoman SIA20F robot. These two robots are positioned on opposite sides of a conveyor, which carries several part trays containing gears. These gears must be moved by the robots (both equipped with parallel gripper end-effectors) to a specified kit tray. As the robots perform the kitting task, the system experiences several mock failures, such as a robot breaking down or a part being misplaced.

Reacting to these failures requires an agile approach to sensing, planning, and control—the details of which can be found in [8]. To briefly summarize:

1. The APRS Laboratory maintains an ontology describing relevant kitting objects and concepts
2. An overhead camera system is used to locate instances of objects described in this ontology
3. A planner uses the ontology and instance information to generate a high-level action plan
4. Low-level controllers execute this plan on the robots

In the described architecture, replacing the physical system with a simulation requires three steps: replicating the camera system, replicating the robot grasping dynamics, and creating an interface between the high-level plan and the low-level virtual controllers used by the simulated robots. Virtual sensors were explored in [21], while implementing a shared interface and achieving accurate dynamics is this work's focus.

Simulation Framework

The APRS simulation framework is built primarily using ROS, which is an open-source suite of robotics-related software packages that are interconnected through a shared message-passing framework. Out of the box, ROS packages provide us with the kinematic descriptions of the robots, joint controllers, joint-state information, forward/inverse kinematic solvers, and collision-aware trajectory planning. ROS can also be extended by writing new packages, which is how we implement the kinematic descriptions of the conveyor and parallel grippers in the system.

We use Gazebo, an open-source robotics simulator, to dynamically simulate the system. Gazebo is designed to be compatible with many of ROS's components, including the robot kinematic descriptions, joint controllers, and joint-state publishers. Gazebo also supports multiple physics engines and allows users to tune the behavior of each engine's numerical solver. We will provide a detailed discussion our choice of physics parameters and physics engine in the "Experiments" section.

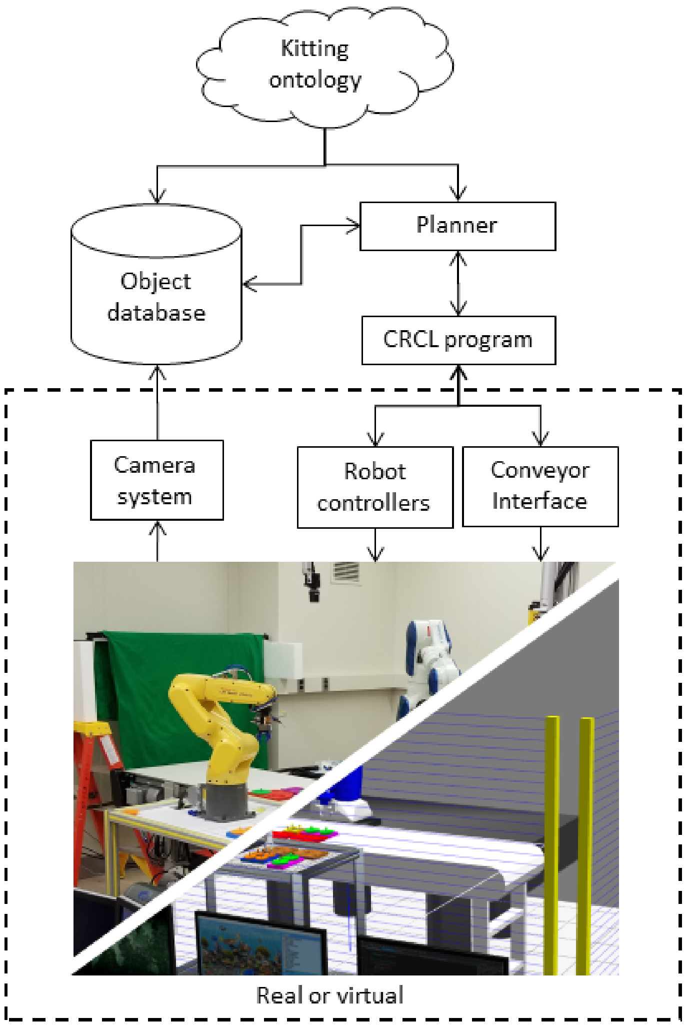


FIGURE 1: APRS system architecture
Components enclosed in dashed box are swappable

Like ROS, Gazebo's functionality can be extended—this time with user-defined plugins, which give access to nearly all aspects of the simulator's physics, sensing, and communication systems. We use plugins to implement both robots' grippers, and the light curtain. The light curtain is implemented as a collection of Gazebo built-in laser rangefinders, whose values are unified into a single output using a plugin. The gripper actuation is handled entirely by a plugin described in the Physics Improvements section. Defining custom behavior using plugins has virtually no performance overhead and can be done in only a few lines of code (as shown in Algorithms 1 and 2).

Our conveyor is implemented using both a ROS controller and a Gazebo plugin. It is modeled as a single planar surface controlled by a linear actuator. The surface needs to apply force to carried objects as though it were moving, while appearing approximately stationary itself (to approximate a realistic conveyor). This is accomplished by using a ROS controller to control the actuator's velocity while using a Gazebo plugin to periodically reset the position of the surface after any small displacement.

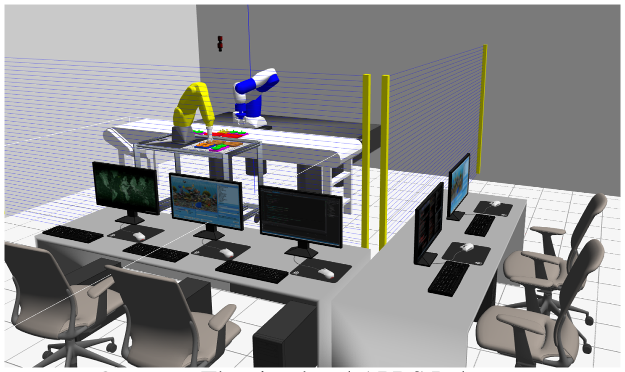


FIGURE 2: The simulated APRS Laboratory

Shared Interface

The physical APRS Laboratory performs low-level control of each robot using proprietary controllers, and operates the conveyor by using a programmable logic controller (PLC) running a Modbus server [22]. These heterogeneous interfaces are unified using the Canonical Robot Command Language (CRCL) [23]. After the APRS architecture synthesizes a high-level plan, this plan is converted into the corresponding CRCL primitives, which are then sent to the appropriate device (either the robot, gripper, or conveyor).

To integrate the simulated robots' controllers with the APRS architecture, we implement a conversion from CRCL primitives to ROS messages, which can be used to specify controller targets and trajectories. The resulting CRCL2ROS library converts CRCL-formatted commands to the ROS-compatible protobuf format [24].

For conveyor commands, we forego using the CRCL2ROS library in favor of re-using the Modbus interface already in-place for the physical conveyor. To do this, we implement a Modbus server in software using the `libmodbus` C++ library [25]. In the physical system, the Modbus server controls the conveyor's velocity by writing to the appropriate registers on the PLC. In the software implementation, the Modbus server controls the simulated conveyor's speed and direction by writing a velocity target to its ROS controller.

4 PHYSICS IMPROVEMENTS

Having established the overall design of the APRS simulation, we will now present two simulation improvements that form the main contribution of this paper. Each improvement targets a component of the simulated system that is prone to unstable, physically unrealistic behavior. "Stability" will be defined and tested more precisely in the "Experiments" section, while this section will focus on the conceptual description of each improvement. We want to stress that these improvements are agnostic to the physics engine used and are simple to implement provided the physics engine supports a programming interface (this is handled by Gazebo's plugin system in our case).

Parallel Gripper Control

The APRS robots perform grasping using pneumatic parallel grippers. A typical approach for defining parallel grippers in software involves mimic joints, where one joint "mimics" a second joint by maintaining its relative position and velocity in joint-space. Unfortunately, many physics engines do not define a joint constraint that describes this mimic behavior directly. Because of this, a popular ad-hoc approach is to use external control plugins [26]. This approach implements a proportional-integral-derivative (PID) force controller that uses the mimic joint's relative joint position as the feedback term. However, there are two problems with using such a control scheme for parallel grasping.

- The output force is 0 N when the error feedback is 0 (i.e., when the gripper's fingers are symmetrically positioned). During grasping, this creates asymmetric forces on the gripper that either cause the grasp to fail or the mimic joint to be pushed to an asymmetric position so that the force generated by the controller matches the force of the original joint.
- Since the original joint's controller has no knowledge of the mimic joint, there is no guarantee that the gripper's fingers will remain symmetric throughout a trajectory if they experience different external forces. For example, if an obstacle blocks the mimic joint but not the original joint, the resulting finger positions would not be symmetric like they would be in the real gripper.

To address these shortcomings, we implement a new control scheme operating on both joints simultaneously:

$$F_1 = F_c + k_p(p_2 - p_1), \tag{1}$$
$$F_2 = F_c + k_p(p_1 - p_2). \tag{2}$$

Here, F_i is the force output for each joint; p_i is the joint-space position of each joint; F_c is a constant force shared by both joints; and k_p is a proportional gain. Using this controller, both joints maintain an identical non-zero force during grasps and are driven to symmetric positions by mimicking each other, satisfying both problems.

This control scheme has an elegant physical interpretation: the F_c term emulates the force generated by the gripper's piston, and the $k_p(\cdot)$ term emulates the normal forces of the gripper's actuator linkage.

While this controller was developed with pneumatic gripper's in mind, it can easily be extended for approximating any mimic joint(s). Given some collection of N mimic joints, a single-joint force-controller F_j, and a relative joint-error force-controller F_s, we can define the control of joint $i \in \{1 \dots N\}$ as:

$$F_i = F_j(p_i) + \sum_{n \in \{1 \dots N\} \setminus \{i\}} F_s(p_n - p_i). \tag{3}$$

In our gripper implementation, $F_j(\cdot)$ was chosen to be the constant force F_c. $F_s(\cdot)$ was chosen to be a proportional controller.

Contact Simplification

The kitting tasks performed by APRS robots involve handling dozens of small objects such as gears and trays. This introduces two challenges to simulation. The first challenge is simulation stability. The stacking of objects, especially objects with large inertia-ratios, is known to cause instability due to over-constraining the linear complementarity problems (LCPs) that are solved by the physics engine at each time step [5]. The second challenge is compute speed, which deteriorates as the number of contacts to simulate increases.

To improve both stability and computational performance, we implement a Gazebo plugin for automatically simplifying contacts between relatively stationary objects. The approach is straightforward in concept:

1. Use Gazebo's built-in `ContactSensor` class to find existing contacts and their contact forces
2. Determine whether the objects in contact are stationary relative to each other
3. Create a virtual, fixed joint between objects satisfying the stationary criteria and disable collisions between them
4. Destroy the virtual joint and re-enable collisions if the objects are no longer stationary, resuming normal behavior

Effectively, this replaces a dynamics problem for the physics engine with a much nicer statics problem for the plugin. The sensing of forces, creation and destruction of joints, and enabling and disabling of collisions can be handled entirely through Gazebo's programming interface. This only leaves the question of how to define the stationary criteria.

For our purposes, two objects are considered stationary if their relative velocity and acceleration are 0. These criteria work fine for creating a virtual joint, but are problematic for deleting it since the virtual joint constrains the relative velocity and acceleration of the objects to be 0. However, we can observe the force needed for the virtual joint to satisfy these constraints and use that as part of the criteria.

More precisely, we can define the net force F_σ on a relatively stationary object as:

$$F_\sigma = F_E + F_C + F_V = 0, \tag{4}$$

where F_E is the external, non-measurable force, F_C is the contact force, and F_V is the virtual joint force. Since only one of either contact forces or joint forces are active at any one time, this equation becomes:

$$F_\sigma = F_E + F_C = 0 \qquad F_\sigma = F_E + F_V = 0. \tag{5}$$

Solving and substituting for F_E, we have $F_C = F_V$ at the time-step that the virtual joint is created. If $F_C \neq F_V$ at a future time-step, then the external force F_E has changed and the original stationary criteria $F_\sigma = F_E + F_C = 0$ is no longer satisfied. The actual plugin uses a threshold on $\|F_C - F_V\|$ rather than a strict inequality, but behaves equivalently.

This plugin offers an improvement over previous dynamics-disabling features found in Gazebo's primary physics engine, which only applied to absolutely stationary objects [5]. It is also fairly physics engine-agnostic and lightweight, provided the physics engine supports measuring constraint forces and dynamically spawning joints. The complete pseudocode is given in Algorithms 1 and 2.

```
Data: GazeboContactSensorMessage Msg,
ErrorThreshold e1, ErrorThreshold e2
Result: VirtualJoint V, ContactForce CF
Executed on new ContactSensorMessage
c1 ← Msg.collision1
c2 ← Msg.collision2
v1 ← GetWorldFrameVelocity(c1)
v2 ← GetWorldFrameVelocity(c2)
a1 ← GetWorldFrameAccel(c1)
a2 ← GetWorldFrameAccel(c2)
if ||v1 − v2|| < e1 & ||a1 − a2|| < e2 then
    CF ← Msg.NetForce
    V ← GazeboCreateNewJoint(c1, c2)
    GazeboDisableContact(c1, c2)
end
```

Algorithm 1: Virtual Joint Creation Callback

```
Data: Virtual Joint V, ContactForce CF,
ErrorThreshold e
Executed on each simulation physics step
if ||CF − V.Force|| > e then
    GazeboEnableContact(V.Collision1,
     V.Collision2)
    GazeboRemoveJoint(V)
end
```

Algorithm 2: Virtual Joint Deletion Callback

5 EXPERIMENTS

In this section, we measure the performance of our two proposed improvements. To do this, we design several scenarios that target the behavior of each improvement in isolation. Each scenario and metric is designed such that the performance of the real system is known to be trivially simple. For example, we know that once a real robot successfully grasps an object, the object will remain completely stationary relative to the gripper during any of the robot's motions (due to the large force exerted by the pneumatic gripper). Now, the objective is to measure whether the simulation violates this "relatively stationary" condition that is known to hold for the real system. Using this approach, we can verify the physical accuracy of our improvements without needing to take any measurements on the real system.

We tested our scenarios primarily with Gazebo's variant of ODE using default simulation parameters. This configuration was mainly used due to technical limitations in Gazebo's support for other physics engines, and because this ODE configuration is fairly common in other work [1].

Parallel Gripper Control

The gripper controller experiments are motivated primarily by the "relatively stationary" condition described above. We consider a grasp more stable and physically accurate if it maintains a constant displacement between the object and the gripper during a motion. Mathematically, we measure this using the Euclidean distance between an initial position and subsequent positions:

$$\delta_t = \|p_t - p_0\|_2 . \tag{6}$$

Here, p is a 3-dimensional relative position and t is a simulation time-step. This metric is similar to Measure C used in [18].

We would also like to incorporate angular displacement into our analysis. To do this, we record the quaternions q_0 and q_t corresponding to initial and subsequent relative orientations between the gripper and grasped object. Thus, we can compute angular displacement as:

$$\theta_t = \arccos\left(2\langle q_t, q_0\rangle^2 - 1\right). \tag{7}$$

We report this value along with δ_t (ideally both should be close to 0 if the simulation is physically accurate). These metrics are computed for our gripper controller along with two baselines:

	Standard	Plugin
Cumulative δ_t	$1.3 \cdot 10^{-2}$ m	$1.0 \cdot 10^{-3}$ m
Real-Time Factor	0.43	0.98

TABLE 1: Multi-object contact simplification experiment

- Constant opposing forces to both joints
- The preexisting mimic joint plugin from [26] with constant force on the mimicked joint

Each controller is attached to a floating gripper (not experiencing any forces). Using each of the three controllers, the floating gripper grasps a conveniently placed APRS gear. Once the gripper is closed, the floating gripper is subjected to a range of accelerations defined in Cartesian world coordinates (emulating the motion of robots performing pick-and-place operations). During these trajectories, the positional and angular displacement metrics are computed and reported using a combination of ROS and Gazebo telemetry.

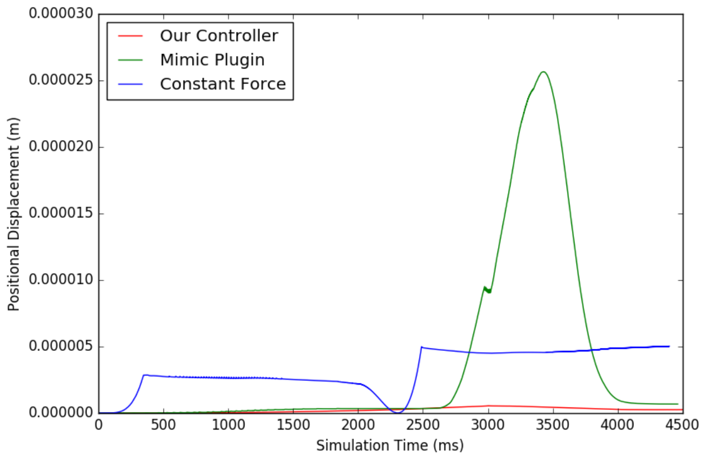
Our results, summarized in Figure 3, show that our proposed parallel gripper controller outperforms both baselines in minimizing positional displacement during Cartesian trajectories. The controller also matches the constant force control scheme in angular displacement. We also note that positional displacement was the largest contributor to objects falling out of the gripper, and that both baselines would drop their objects from much smaller impulses as compared to our proposed controller.

Contact Simplification

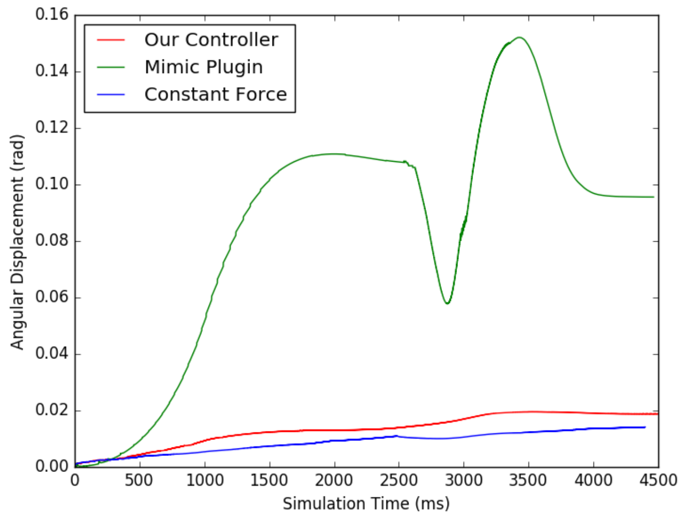
Like our previous test, the experiment for our contact simplification improvement uses another "relatively stationary" condition. This time, we know that gears placed inside trays must be stationary relative to the tray (based on their coefficient of friction and the tray design shown in [8]). Therefore, we can reuse the metrics given by Equations 6 and 7, this time computing the relative displacement from gear to tray (rather than gear to gripper).

The scenario that we used consists of a conveyor carrying multiple trays. Each tray is loaded with gears, reminiscent of a typical kitting task. We choose this particular configuration to highlight the benefits of our improvement to both accuracy and run speed for simulations with scaling complexity.

We summarize our results in Table 1. This data was collected based on a simulation of 10 trays carrying 40 gears, which we believe adequately demonstrates the effect of the plugin on both accuracy and simulation speed. We record the relative displacement δ_t between a gear and the tray at $t = 4$ s. We then sum the δ_t values from all 40 gears to compute a cumulative δ_t value for the scenario. We also record the real-time factor of the simulation.



(a) Positional Displacement

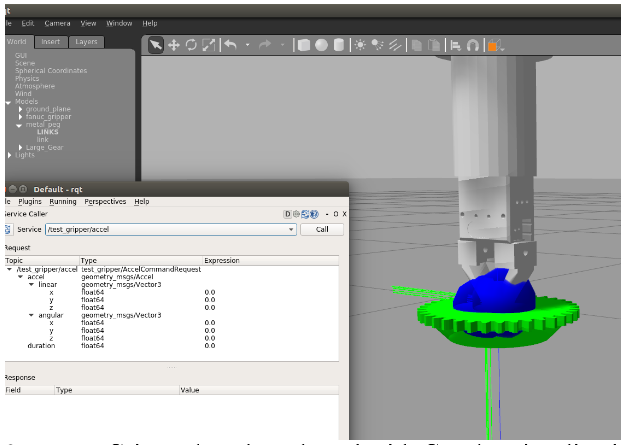


(b) Angular Displacement

FIGURE 3: Relative displacement along a Cartesian trajectory using various controllers

As expected, once the contact simplification plugin creates a virtual joint, the displacement of the gear relative to its tray is fixed. This gives the plugin a slight edge over the standard approach by limiting the effect of any unstable simulation behaviour (like jittering contacts).

The plugin also has a pronounced effect on the simulator's compute speed. On average, every collision between objects in our simulation had about 10 contact points that needed to be processed per time step. The plugin effectively reduced this to one contact point per collision.



Service /test_gripper/accel

Request

Topic	Type	Expression
/test_gripper/accel	test_gripper/AccelCommandRequest	
accel	geometry_msgs/Accel	
linear	geometry_msgs/Vector3	
x	float64	0.0
y	float64	0.0
z	float64	0.0
angular	geometry_msgs/Vector3	
x	float64	0.0
y	float64	0.0
z	float64	0.0
duration	float64	0.0

Response

Field	Type	Value

FIGURE 4: Gripper benchmark tool with Gazebo visualization

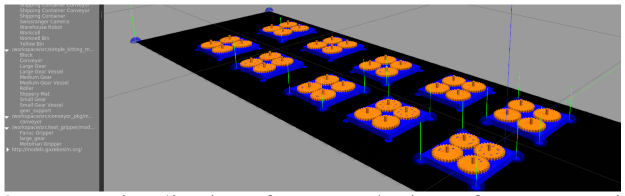


FIGURE 5: Visualization of contact plugin performance testing

6 CONCLUSION

We have presented a physics-based simulation of the APRS Laboratory along with several techniques that enabled it to reach good physical accuracy (as measured by the degree to which the simulation violates static friction conditions). We have demonstrated how our simulation integrates into the overall APRS architecture, and how its design leverages open-source software to include additional industrial components, such as conveyors, grippers, and light curtains.

Given that the APRS Laboratory simulation now achieves good physical accuracy on kitting-related tasks, it can now be extended to simulating entire kitting scenarios currently being developed for the physical APRS Laboratory. To do this, future work will involve improving the software tools needed to create these scenarios, extending the robot agility measures used by the physical lab to the virtual environment, and disseminating the virtual environment to external collaborators. The virtual environment also opens new areas of data-intensive research that are too costly to run on actual hardware, such as controlling robots using reinforcement learning agents or testing the viability of new sensor configurations for computer vision tasks.

Disclaimer

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

REFERENCES

[1] NIST, 2017. Agile robotics for industrial automation competition. `www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation`. Accessed 29-April-2019.

[2] Hsu, J. M., and Peters, S. C., 2014. "Extending Open Dynamics Engine for the DARPA Virtual Robotics Challenge". In *Simulation, Modeling, and Programming for Autonomous Robots*, D. Brugali, J. F. Broenink, T. Kroeger, and B. A. MacDonald, eds., Vol. 8810. Springer International Publishing, Cham, pp. 37–48.

[3] Todorov, E., Erez, T., and Tassa, Y., 2012. "MuJoCo: A physics engine for model-based control". In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp. 5026–5033.

[4] NVIDIA, 2019. PhysX SDK 4.0. `https://developer.nvidia.com/physx-sdk`. Accessed: 29-April-2019.

[5] Smith, R., 2006. Open Dynamics Engine. `www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/ode-devel/work/ode-060223/ode/doc/ode.pdf`. Accessed: 29-April-2019.

[6] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., 2009. "ROS: an open-source robot operating system". In ICRA workshop on open source software, Vol. 3, Kobe, Japan, p. 5.

[7] OSRF, 2014. Open Source Robotics Foundation - Gazebo. www.gazebosim.org.

[8] Kootbally, Z., Schlenoff, C., Antonishek, B., Proctor, F., Kramer, T., Harrison, W., Downs, A., and Gupta, S., 2018. "Enabling robot agility in manufacturing kitting applications". *Integrated Computer-Aided Engineering,* ***25***(2), pp. 193–212.

[9] Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., Carff, J., Rifenburgh, W., Kaveti, P., Straatman, W., Smith, J., Griffioen, M., Layton, B., de Boer, T., Koolen, T., Neuhaus, P., and Pratt, J., 2015. "Team IHMC's lessons learned from the DARPA robotics challenge trials". *Journal of Field Robotics,* ***32***(2), pp. 192–208.

[10] Aguero, C. E., Koenig, N., Chen, I., Boyer, H., Peters, S., Hsu, J., Gerkey, B., Paepcke, S., Rivero, J. L., Manzo, J., Krotkov, E., and Pratt, G., 2015. "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response". *IEEE Transactions on Automation Science and Engineering,* ***12***(2), pp. 494–506.

[11] Bai, Y., Yu, W., and Liu, C. K., 2016. "Dexterous manipulation of cloth". In Proceedings of the 37th Annual Conference of the European Association for Computer Graphics, EG '16, Eurographics Association, pp. 523–532.

[12] Qian, W., Xia, Z., Xiong, J., Gan, Y., Guo, Y., Weng, S., Deng, H., Hu, Y., and Zhang, J., 2014. "Manipulation task simulation using ROS and Gazebo". In 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), IEEE, pp. 2594–2598.

[13] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S., 2017. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". *arXiv:1709.10087 [cs]*, Sept. arXiv: 1709.10087.

[14] Erez, T., Tassa, Y., and Todorov, E., 2015. "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX". In 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 4397–4404.

[15] Chung, S.-J., and Pollard, N., 2016. "Predictable behavior during contact simulation: a comparison of selected physics engines". *Computer Animation and Virtual Worlds,* ***27***(3-4), pp. 262–270.

[16] Peters, S., and Hsu, J., 2014. Comparison of Rigid Body Dynamic Simulators for Robotic Simulation in Gazebo. `www.osrfoundation.org/wordpress2/wp-content/uploads/2015/04/roscon2014_scpeters.pdf`. Accessed: 29-April-2019.

[17] Taylor, J. R., Drumwright, E. M., and Hsu, J., 2016. "Analysis of grasping failures in multi-rigid body simulations". In 2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), IEEE, pp. 295–301.

[18] Kim, J., Iwamoto, K., Kuffner, J. J., Ota, Y., and Pollard, N. S., 2013. "Physically Based Grasp Quality Evaluation Under Pose Uncertainty". *IEEE Transactions on Robotics,* ***29***(6), pp. 1424–1439.

[19] Pepper, C., Balakirsky, S., and Scrapper, C., 2007. "Robot simulation physics validation". In Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems, ACM, pp. 97–104.

[20] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C., 2007. "USARSim: a robot simulator for research and education". In Proceedings 2007 IEEE International Conference on Robotics and Automation, IEEE, pp. 1400–1405.

[21] Aksu, M., Michaloski, J., and Proctor, F., 2018. "Virtual experimental investigation for industrial robotics in gazebo environment". In Proceedings of the 2018 International Mechanical Engineering Congress and Exposition, ASME.

[22] Swales, A., et al., 1999. "Open MODBUS/TCP specification". *Schneider Electric*.

[23] Proctor, F., Balakirsky, S., Kootbally, Z., Kramer, T., Schlenoff, C., and Shackleford, W., 2016. "The canonical robot command language (CRCL)". *Industrial Robot: An International Journal,* ***43***(5), pp. 495–502.

[24] Google, 2019. Protocol Buffers. `developers.google.com/protocol-buffers/docs/overview?csw=1`. Accessed: 29-April-2019.

[25] Raimbault, S., 2016. libmodbus: A Modbus library for Linux, Mac OS X, FreeBSD, QNX and Windows. `libmodbus.org`.

[26] Chatzilygeroudis, K., 2019. Robotics Group Gazebo Plugins. `https://github.com/roboticsgroup/roboticsgroup_gazebo_plugins/`. Accessed: 3-April-2019.