

Team RuBot's Experiences and Lessons from the ARIAC [★]

Si Wei Feng^a, Teng Guo^a, Kostas E. Bekris^a and Jingjin Yu^a

^aDepartment of Computer Science, Rutgers University

ARTICLE INFO

Keywords:

Industrial robot
Robot agility
Robotics challenge
Multi-robot system
Pick-and-place system

ABSTRACT

We share experiences and lessons learned in participating the annual Agile Robotics for Industrial Automation Competition (ARIAC). ARIAC is a simulation-based competition focusing on pushing the agility of robotic systems for handling industrial pick-and-place challenges. Team RuBot started competing from 2019, placing 2nd place in ARIAC 2019 and 3rd place in ARIAC 2020. The article also discusses the difficulties we faced during the contest and our strategies for tackling them.

Video of system sketches: <https://youtu.be/7H7YLeJz2zE>.

1. Introduction

Millions of industrial robots are currently employed in modern factories that significantly improve productivity and reduce repetitive manual labor. In a vast number of industrial robotics applications, the pick-and-place operation plays an indispensable role in moving parts and products.

To help enhance the agility of industrial robots, the National Institute of Standard and Technology (NIST) initiated the Agile Robotics for Industrial Automation Competition (ARIAC) in 2017 [12] and started to give cash awards to top 3 winners one year later. The competition focuses on testing the agility of industrial robot with pick-and-place tasks and has been held for four consecutive years.

Each year's ARIAC comprises of at least two rounds, including one or more qualification rounds, where every eligible person or team can register at the beginning, and a final round among ~5 teams selected based on their performance in previous rounds. The competition has seen an increasing number of registrations. In 2020, over 100 teams registered in the qualification round.

ARIAC is a simulation-based competition using Gazebo [8], an open-source 3D simulator by Open Source Robotics Foundation (OSRF). Gazebo Environment for Agile Robotics (GEAR), designed by OSRF and maintained by NIST, is used as an interface software between competitor's package and the competition environment. In the competition, participants are required to design a system in ROS [25] to fulfill a set of shipments. One or more robots are provided, and participants are to decide among a variety of sensor choices including break beam, proximity sensor, laser scanner, camera, and so on, to deploy in the environment. The anticipated system should be able to pick products from a set of places including bins, shelves, and a conveyor belt. The products are then to be placed on the tray of an Autonomous Guided Vehicle (AGV) to fulfill a shipment. Scoring metric of ARIAC takes into account the number of products

successfully shipped, the pose of the shipped product, the time used for each shipment, and the costs of sensor deployment. The contest aims to simulate the real industrial environment for pick-and-place to the greatest possible extent by considering an exhaustive list of events that could occur in a warehouse environment and selecting a set of agility tasks to fulfill. So far, organizers of ARIAC have selected eight agility challenges based on industry feedback and brought them into the competition scenarios. These eight scenarios are: presence of faulty products, insufficiency of products, flipped products, in-process order update, sensor blackout, dropped products, in-process high priority order interruption, and moving obstacles.

Rutgers Algorithmic Robotics and Control Lab (ARCL) started competing in ARIAC since 2019 with its Team RuBot entry, consisting mainly of PhD students. Team RuBot placed 2nd in ARIAC 2019 and 3rd in ARIAC 2020. In this article, we share our journey through the competitions and our lessons learned in navigating the challenging multi-objective optimization tasks. To that end, we start with a brief overview of ARIAC and related research. We then outline our system design for ARIAC 2019 and ARIAC 2020. As these sketches are unfolded, we discuss the challenges we faced and how we addressed them.

2. Background


Since the inception of ARIAC, the essential task for the required robotic systems has been picking a set of products (pulley, disk, piston rod, gear and gasket) from a conveyor belt, bins, or shelves and placing them on a tray for delivery. In addition to the main pick-and-place task, eight agility challenges, listed below, were provided to test the agility of the systems designed by competition participants.

1. Faulty Products. Some products are faulty. After a faulty product is placed on the AGV and detected by the sensor in the AGV, it should be removed from the shipment.
2. Insufficiency of products. When an insufficient number of products is present, competitors should consider the alternative of submitting a shipment with the available products.

[★] Our submissions to ARIAC are completely open sourced at Github.

Team RuBot 2019: <https://github.com/ustcsiwei88/RuBot>

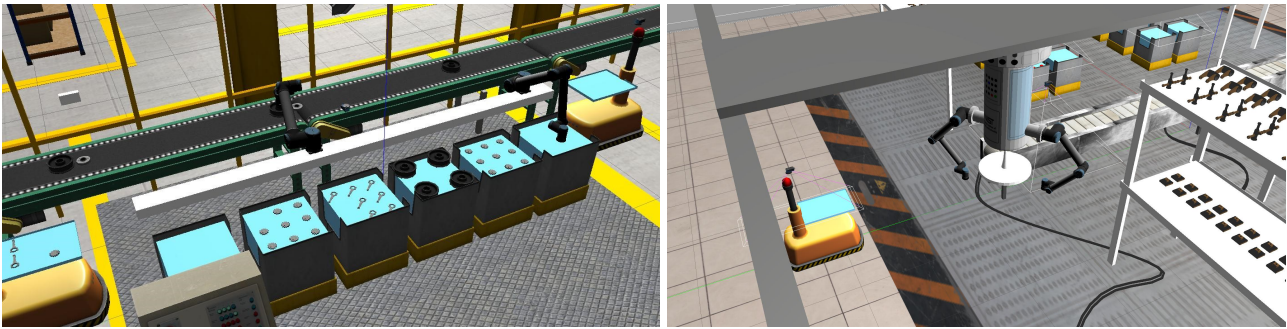
Team RuBot 2020: <https://github.com/ustcsiwei88/RuBot2020>

 siwei.feng@rutgers.edu (S.W. Feng);

gt286@scarletmail.rutgers.edu (T. Guo); kostas.bekris@cs.rutgers.edu

(K.E. Bekris); jingjin.yu@cs.rutgers.edu (J. Yu)

ORCID(s):



(a) ARIAC 2019 with two UR-10 arms

(b) ARIAC 2020 with a 15-DoF gantry robot

Figure 1: ARIAC Environments

3. Flipped products. Some products must be flipped first before being put onto the shipment tray. This often requires the collaboration between two arms.
4. In-process order update. After the announcement of a shipment and before the submission of the shipment, an order update may happen that can change the type of products needed for the shipment.
5. Sensor blackout. During the competition, some sensors may stop working for a period of time.
6. Dropped products. The grippers on the arms are simulated to have the possibility of dropping the products they have grasped.
7. In-process high-priority order update. During the assembling of an order, a higher priority order may come in that ideally should be handled immediately.
8. Moving obstacles. Obstacles, e.g., human beings, are simulated to move in the environment. Collisions with these moving obstacles should be avoided.

In 2019, the robots provided were two UR10 robotic arms mounted on a shared linear actuator (see Fig. 1a), making each unit seven degrees of freedom (7-DoF). In 2020, a 15-DoF robot was provided. The 15-DoF robot consists of a gantry base with 3-DoF and two UR10 arms mounted on it, each of which has 6-DoF. This made it possible for the robot to move around in the workshop (see Fig. 1b). People could use motion planning libraries like MoveIt [4] or simply publish a joint trajectory, which consists of a sequence of waypoints and timestamps to the joint trajectory controller, to control the robots. In both years, the end-effectors of the robots provided were vacuum grippers, controlled through the interactions with GEAR.

Participants can choose from a set of sensors and place them inside the environment. Each of the sensors provided is associated with a cost of deployment as follows.

\$100: Break beam, proximity sensor and laser profiler.

\$200: Depth camera.

\$500: Logical camera and RGBD camera.

The scoring metrics of ARIAC considers various aspects; readers are referred to [24] for detailed explanations. Here, we give a brief explanation for the scoring metric equation:

$$Trial\ Score = AAC * (CF * AVG(CS) + \sum_i EF_i * CS_i)$$

AAC Arm/Arm Collision. 0 when arm/arm collision occurs, 1 otherwise.

CF Cost Factor. It is inversely proportional to the total sensor cost.

AVG Average over all shipments.

CS_i Completion Score for shipment *i*, which is defined as the number of correct products delivered plus the number of products in the correct pose, plus bonus points.

EF_i Efficiency Factor for shipment *i*. This factor is inversely proportional to the time cost from order issue to order delivery.

Related Work. Robot pick-and-place systems [2, 3, 11, 14, 22, 23, 27] have attracted enormous attentions from both researchers and industrial practitioners. Whereas components like motion or grasp planning [21], object rearrangement [10, 19], 3D pose estimation [17], picking from conveyor [9, 15] and so on are investigated in individual studies, large scale physical deployment of holistic systems have made great achievements, such as the Kiva system from Amazon [26] and the DARPA Robotics Challenges [1, 7, 16, 20]. Even with so many successes in both academic research and industrial systems, it has been pointed out that many characteristics of robotics systems, although not novel, still remain largely unexplored in literature [6]. Current robotics systems have a long way to go before reaching their physical capabilities; aspects like long programming time, failure to quickly adapt to changing environments or failures, and scarcity of abilities trained on industrial robots impact productivity and agility of the system [18]. As such, NIST's ARIAC competition timely promotes agility, which is essential for bringing out the true potential of highly capable robotic hardware.

3. System Overview

Competitors in ARIAC are tasked to design a software in the form of an ROS package and provide necessary scripts for installing dependencies and running the software. The software must be capable of retrieving the sensing data and

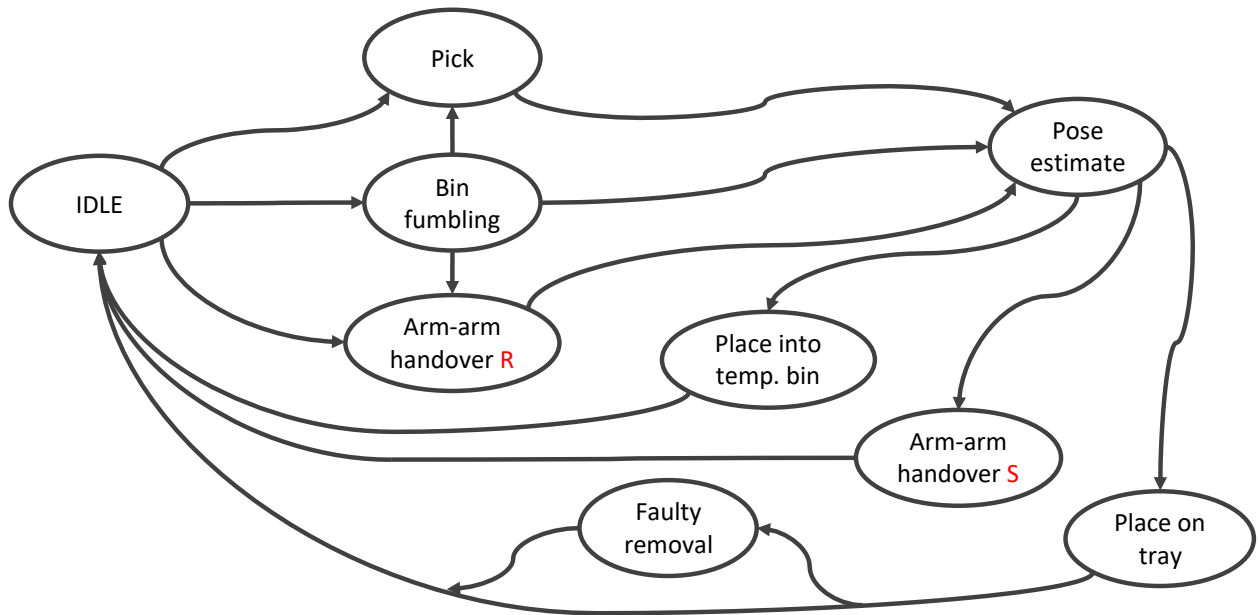


Figure 2: State diagram for a single UR10 arm for ARIAC 2019.

control the robots in Gazebo simulations within an Ubuntu Bionic and ROS Melodic environment. Team RuBot's submissions were implemented in C++ for both ARIAC 2019 and ARIAC 2020. No auxiliary software dependencies exist in our robot control sub-system.

Team RuBot takes a hierarchical approach that can be dissected into two levels or phases: high-level task planning followed by low-level motion planning. The former performs high level planning, i.e., telling the robot what to do next. In our implementation, one or more Finite State Machines (FSMs) were constructed for robot task allocation. To fulfill these tasks, the latter low-level motion planning (i.e., trajectories planning of the two UR10 arms and the 15-DoF gantry robot and controlling the grippers) is necessary.

While the competition allows a fairly flexible sensor selection from a large group of sensors including break beam, laser scanner, proximity sensor, and so on, well encapsulated "logical cameras" are also provided which are able to carry out pose estimation precisely in simulation. We opted to heavily rely on "logical cameras" for pose estimation. Laser scanners were also occasionally used to help with object detection and simple classification of objects on the moving conveyor belt (based on heights).

3.1. High-Level Task Planning

From the start of our participation of ARIAC, we concluded that it is convenient and reliable to encode the core logic required by the ARIAC task as an FSM with predefined states and interconnections.

Generally, the tasks of the robot fall into two classes. The first contains static tasks such as picking products from static shelves, bins, or flipping a product. The second contains dynamic tasks that come during the run time, for example the removal of faulty parts after they have been detected, pick-

ing products from moving conveyor belt when they arrive, handling order update, and so on. Most of the tasks could be handled in a single state, while others need to be refined into smaller tasks. Dealing with order updates would be a good example for the later, because each product in an order update will create two sub tasks: picking the product from the tray and putting the product back to their bins or shelves.

3.1.1. RuBot 2019

In ARIAC 2019, two UR10 robotic arms are provided, which are mounted on a shared linear actuator (Fig. 1a). The workspaces of these two robotic arms are separated enough to allow us to handle them separately by adding some locks for entering the shared region. Each robot is responsible for the tray near the end the actuator on its side.

In Fig. 2, we show the state diagram designed for a single robotic arm. The two arms are controlled by two separate FSMs with similar structures. Each robot starts in the **IDLE** state. From here, it may go to the state of receiving a product from the other arm (**Arm-arm handover R** state), or go to the state of picking a product from the conveyor belt or the tray (**Pick** state), or start searching for a product inside the bins (**Bin fumbling** state).

It is worth noting that, to reduce sensor cost, no sensor is put above the bins which means the robot arms will need to search in the bin for potential product. This creates the **Bin fumbling** state. Since searching in the bin costs time and is of lower priority, the state could be interrupted by products from the conveyer belt or an order update, which set the state to **Pick** state, or interrupted by the signals sent from other arm, which set the state to **Arm-arm handover R** state.

After going through one of the above three states and the corresponding manipulations, the robot will have a product grasped in the gripper. The robot then brings the product

to the “logical camera” for precise product classification and position estimation (**Pose estimate** state). This state is a pivotal state as the whole system relies on the single “logical camera” to estimate the pose of products for precise placement of product onto the tray. It also helps with making small position tuning for the Arm-arm hand-over task.

Once pose estimation is complete, there are three potential states to go to: 1. sending the product to the other arm while deciding whether to flip the product (**Arm-arm handover S state**), 2. placing the product onto the tray of an AGV for shipment (**Place on tray** state), or 3. placing the product into temporary bins (**Place into temp. bin** state). When the arm finishes these manipulations, it will return to the **IDLE** state to take new tasks. The arm may enter a **faulty removal** state to remove the faulty product in the tray, if any.

3.1.2. Dual-Arm Collaboration in RuBot 2019

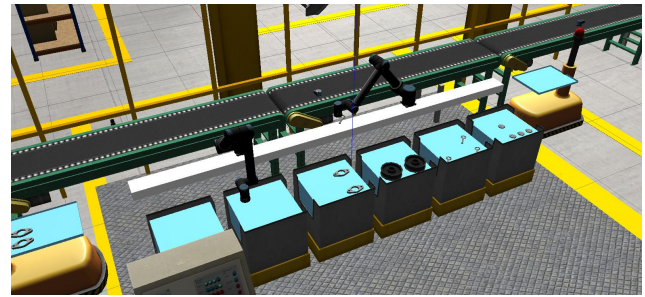
Two special states in the state diagram are the states of Arm-arm handover when the synchronization of the state machines of the two arms are needed. Specifically, when one arm is waiting at **Arm-arm handover S** state to send a product to the other arm, the other arm needs to get to **Arm-arm handover R** state quickly to receive it.

In the Arm-arm handover states, two types of handover manipulations were defined. The first type of manipulation is a combination of actions shown in Fig. 3a and Fig. 3b where one arm puts the product on the actuator while the other arm comes to fetch the product later. In the second type, one arm directly gives the product to the second arm which flips the product during the transfer. When a product needs to be flipped, the latter type of manipulation should be carried out.

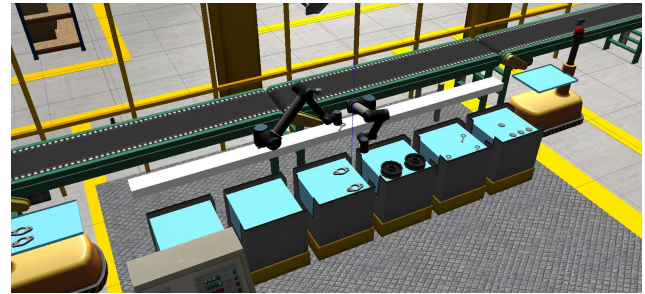
In early development, infinite loops were observed during the simulation because of incorrect action priority settings. The reason behind this phenomenon, which is quite subtle, was that each arm considers cooperating with the other arm to be more important. It happens that when arm 1 needs a product and arm 2 has that product, arm 2 will give that product to arm 1, but later arm 1 finds out arm 2 also needs that product, arm 1 will give the product back to arm 2. This can turn into an infinite loop. To resolve this, four prioritized arm actions were defined after the **Pose estimate** state to prevent this from happening:

1. Place the product grasped onto the tray for which the arm is responsible.
2. Give the flipped product grasped to the other arm directly (Fig. 3c).
3. Place the product grasped on the actuator and notify the other arm to fetch it (Fig. 3a).
4. Place the product grasped to a temporary bin as the product is no longer required for shipment.

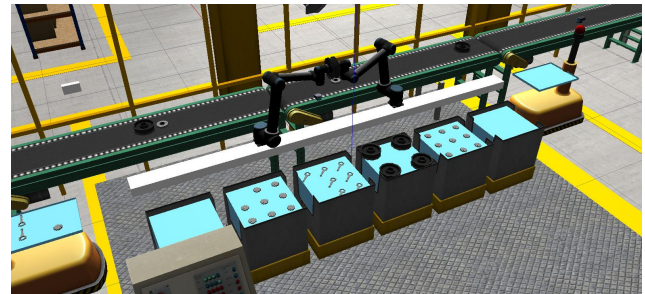
For example, if arm 1 holds a not flipped pulley part which both tray 1 and tray 2 need for shipment, then arm 1 will choose action 1, which places the part on tray 1, instead of action 3, which places the part on the actuator in order to give it to arm 2, because action 1 has higher priority. The priority selection ensures that positive progress is always made.



(a) Arm 1 puts a product in the actuator



(b) Arm 2 fetches the product from the actuator



(c) Arm 2 gives the product directly to arm 1

Figure 3: Three arm-arm handover manipulations

3.1.3. RuBot 2020

In ARIAC 2020, two UR10 robotic arms are mounted on a 3-DoF gantry base to obtain a single 15-DoF robot (Fig. 1b). To simplify the control of the 15-DoF robot, we assume that only one arm can be controlled at one time. The complexity of planning and collision avoidance of allowing the two arms picking or placing products simultaneously is henceforth eliminated, though simultaneously pick-and-place will fully utilize the advantage of the dual-arm robot. With the simplification, we use a single FSM to encode the logic and control the robot.

To reduce the number of states in the state machine, thus reducing interconnections among states, we merged a set of similar tasks into the same state. For example, we merged four types of picking operations (pick from bin, shelf, tray, conveyor belt) into the same picking state. As we put more cameras in ARIAC 2020, classification and pose estimation can be carried out at almost everywhere where a product could appear. As a result, there is not a standalone state for pose estimation.

Initially, the FSM starts from the **IDLE** state. As the robot picks an item from the conveyor belt, bin, or shelf, it

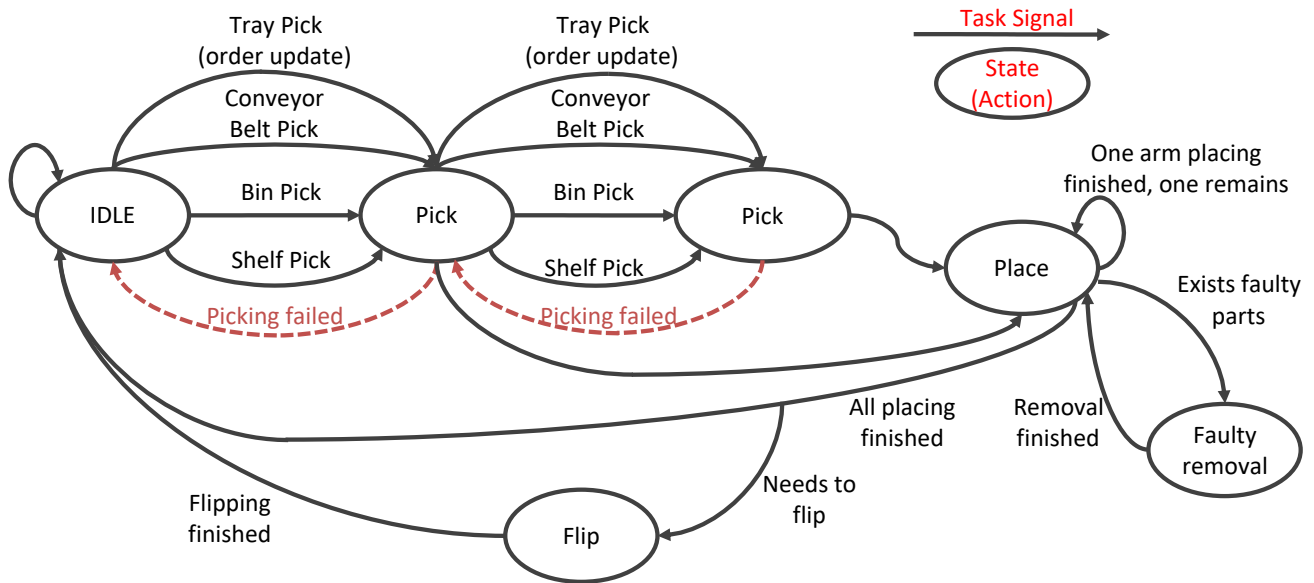


Figure 4: State diagram for the 15-DoF robot for ARIAC 2020

goes to the **Pick** state. In particular, it is also possible that the robot needs to pick an item from the tray in the **Pick** state if an order update exists.

Then, if there is nothing else to pick, it goes into the **Place** state in which the robot place an item on the tray for shipment in the case of fulfilling an order or put the item back to a bin if it corresponds to an order update.

As the robot has two arms, it is possible for the robot to carry two items to the destination; another **Pick** state is added to take advantage of the dual arm setup. Once a picking failed, the system will revert back to the previous state.

In the **Place** state, the robot may need to place one or more items onto the tray for order fulfillment or bins for putting products back during order update. This may cause the **Place** state to repeat one more time. At the same time, if the robot finds out that the product just placed is faulty, it will immediately go into the **Faulty removal** state to use the same gripper to pick the product from the tray and throw it away. After all of these are finished, the state goes back to the **IDLE** state. Along the way, the robot may enter the **Flip** state to flip a product, if required.

3.2. Low-Level Motion Planning

3.2.1. Grasping

For grasping products, a vacuum gripper is attached to each UR10 arm. The grippers provide messages indicating whether a product has been attached to it, which can be used for failure detection.

Grasping a product, though basic, is not a simple task. For example, picking a product from a moving conveyor entails much more than simply letting the vacuum gripper reach the product, even when the object is a simple disk. On the other hand, the products to grasp in the environment are not too complicated and always lie flat on a surface. This allows us to take a simplified ad-hoc approach to let the gripper go

straight downwards towards some product in order to grasp it. For instance, if the product is static, the gripper will go along the trajectory shown in Fig. 5a. In the case when the product is on a moving conveyor, the gripper will start from the top of the product and move along with the product at the same horizontal speed while moving downwards, as shown in Fig. 5b.

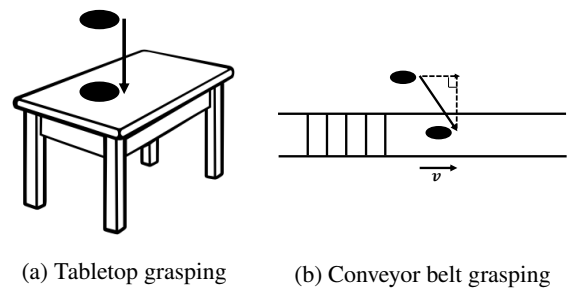


Figure 5: The vacuum gripper is always facing towards the product when grasping. The black solid arrows show the direction of gripper movement.

3.2.2. Trajectory Planning

In practice, arm trajectories are often simply specified by human experts, using pre-computed waypoints or using pre-computed roadmaps to navigate. In fact, human guidance in robot trajectory planning has proved to be effective [5] besides pure planning. In our approaches, pre-computed intermediate waypoints were employed to navigate the robots to the destination, in place of using a planning package like MoveIt [4]. The numbers of waypoints for the robotic arms in our approaches are listed in Table 1 and 2.

These waypoints of the robotics arm were designed based on human experiences. The values of the linear (prismatic) joints in ARIAC 2019 and the three gantry joints in ARIAC

#waypoints	Action
3	pick from conveyor belt
13	pick a product from the actuator trying 5 locations
5	pick from tray
4	search a product in the bin
4	place a product on the actuator
4	place a product on a tray
1	send a product near to the camera for classification
2	throw a faulty product
2 (arm 1) + 3 (arm 2)	flip a product

Table 1

Numbers of waypoints designed for the arm to complete each action in RuBot 2019.

#waypoints	Action
5	pick from the conveyor belt
4	pick from a bin
7	pick from an upper shelf
9	pick from a lower shelf
4	pick from a tray
3	place a product on a tray
4	place the product back to the upper shelf or bin
1	throw a faulty product
8 (arm 1) + 10 (arm 2)	flip a product

Table 2

Numbers of waypoints designed for the arm to complete each action in RuBot 2020.

2020 are chosen as selected configurations near the destination. Because the mobility for the robot in the workshop come mostly from these joints, choices of these joints' values need to conform to the arm trajectories. This is the main way for the robots to avoid collisions in our approach. Time stamps to reach these waypoints are chosen empirically to ensure feasibility of the trajectories.

A physical UR10 arm can be controlled by URScript, a script language that commands the arms. Forward Kinematic (FK) and Inverse Kinematics (IK) are implemented fairly efficiently in the firmware. However, for simulation in Gazebo, the FK and IK need to be computed by the program as there is no URScript engine simulation. Due to the design of UR robotic arms, the IK of UR10 has closed form expressions [13] and can be even simpler when the vacuum gripper is always facing downwards. However, there could be as many as 8 solutions among which we need to choose one that does not have collisions with the gantry base or the environment.

Before the competition starts, we pre-compute the waypoints of these trajectories using IK functions of UR-10 [13] and corresponding Denavit-Hartenberg (D-H) parameters. With the pre-computation for each action, there is little cost for trajectory planning at run time.

4. Lessons Learned

4.1. Learning from Failures

In 2019 and 2020 ARIAC final rounds, 15 tests covering all the agility challenges were performed on the competitors' software. From the logs traces of the finals tests, we could identify where the systems failed. As a matter of fact, our systems are still far from being perfect. In 2019, 4 out of 15 tests were not successfully carried out, while the number of failed runs increased to 5 in 2020.

Compared with other mistakes like lacking precision for products' poses on the tray or missing products for a shipment, system failures are much more serious because the whole control logic cannot proceed once a system failure occurs, resulting in a zero point for that run. Some screenshots in Fig. 6 and Fig. 7 illustrate some typical failures experiences by our system.

In ARIAC 2019, the submitted software scored 0 point in 4 tests in total. In test 3 and 9, a robot got stuck when picking a product from the conveyor belt. When the arm has grasped a gasket part (Fig. 6a), the collision created by the product and the environment made the predefined trajectory prone to getting stuck. In test 8 (Fig. 6b), the arm did not realize the product it grasped has fallen and kept waiting for the product information from the camera. In test 10 (Fig. 6c), a failure happened when one arm was handing over a product to the other arm. The two suction-based grippers coupled together because the arm which was supposed to receive the product from the other arm did not realize that it had grasped the other gripper instead of the product.

In ARIAC 2020, there are a total of 3 tests (test 4, 8, 15) where we failed to deliver anything due to a bug in handling parts on the conveyor belt. Specifically, when an item moved to the end of the conveyor belt, we did not delete it from the queue for storing products on the conveyor belt. As a result, impossible trajectory points were given to the gantry controller and the robot was unable to reach it, and our system froze up (Fig. 7a). The system failed to be robust in another way as it neglected unknown products (Fig. 7b), a disc in the given scenario, in the shipment orders or in the environment. The disc products did not appear in the task description, and our system would not detect any of the disc products. As a consequence, accessing undefined areas (e.g., trying to access a nonexistent array storing the positions of disc products) happened in run time when there exists an unknown disc product in a shipment order. So, in test 9, the existence of disk parts/products lead the system off the designed functionality.

Sometimes, the product destinations are not reachable from overhand grasps. The IK for grasping the red gasket product in Fig. 7c, for instance, is not solvable if the grasping is in a top-down manner towards the center of the product. In RuBot 2019, this issue was resolved by distinguishing these destinations and find a pose near the target but still reachable. In RuBot 2020, this issue was left unaddressed. As a result, a set of "NaN" values (e.g., calling $\text{acos}(1.01)$ without checking whether there is a solution will get NaN as result)

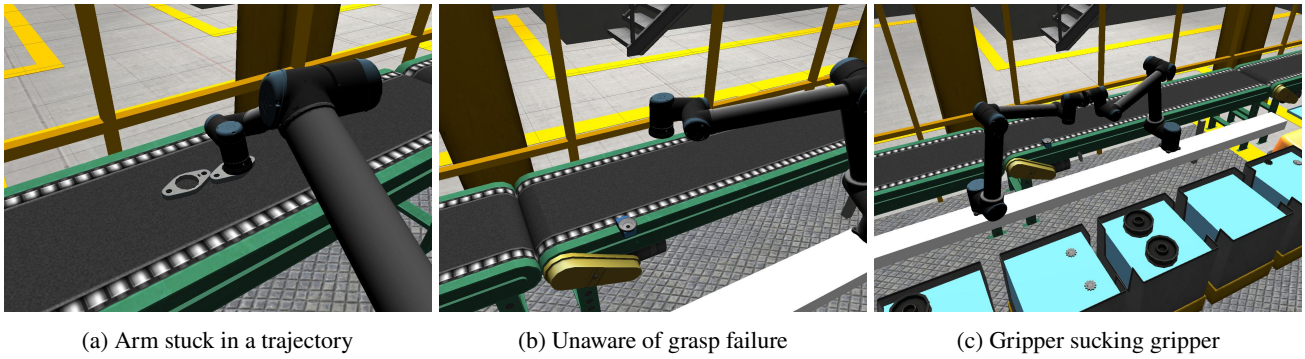


Figure 6: Some system failures in RuBot's ARIAC 2019 entry.

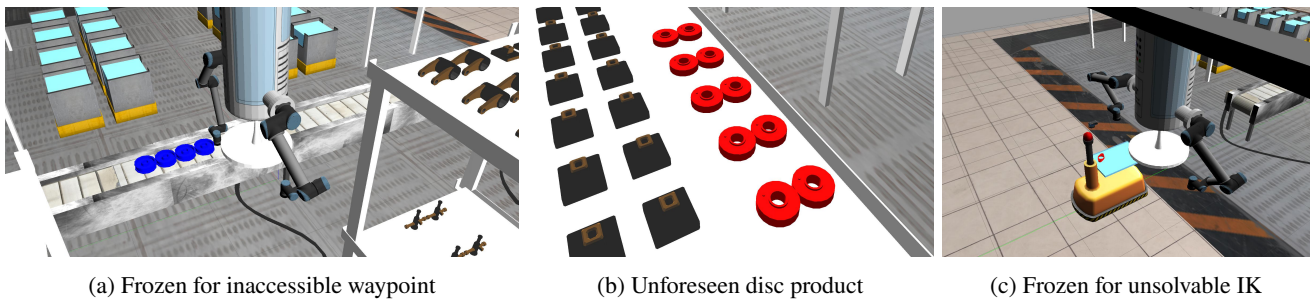


Figure 7: Some system failures in RuBot's ARIAC 2020 entry.

were set as the joint angles to the joint trajectory controller, leading to the system failure shown in Fig. 7c.

Looking back into these issues, most of them seem preventable with a more careful design and sufficient tests. On the flip side, to be less optimistic, considering all aspects of a system and thoroughly carrying out robustness tests are time consuming and error prone in general. This means designing resilience into the system can be important. That is, the system should be designed to be able to identify errors and recover from them to some extent. In our system, if only we had designed a simple mechanism for the robot to restart, for example, closing the vacuum gripper and going back to its original place whenever it has been stuck or frozen somewhere long enough, over half of the system failures could have been avoided. An important lesson for us is to always add such recovery module when running a robotics system.

4.2. Selection of Planning Method

As mentioned, we used pre-computed trajectories for our motion planning realization. This choice benefits us from the beginning as it is easier to start with, which gives us more time to focus on the core logic of the robots' actions. However, relying on pre-computed trajectories and human-defined waypoints turned out to require a lot of energy and lack scalability despite our efforts in encapsulating the planning function and reuse certain trajectories. We believe that using a stand-alone trajectory planner or pre-computation backed by a general planner would be a better approach going forward.

4.3. Pose Representations

Pose estimation is probably the most important module of a pick-and-place system after planning. In ARIAC, the focus is not perception. Due to the focus, well encapsulated "logical cameras" were provided for retrieving the exact poses and types of the products in their ranges of visibility. The remaining task for us is to choose the proper representations of the poses, i.e. pose of the grippers, cameras, products, and desired product destinations on the tray.

In the message given from the task publisher, poses of products are represented as a combination of a 3-D position and a quaternion $w+x\cdot i+y\cdot j+z\cdot k$, ($x^2+y^2+z^2+w^2=1$). Quaternions are quite convenient when doing frame transformation (e.g., from the camera's frame to the delivery tray's frame) as multiplication and inversion are the only things needed. But in certain cases, conversion from quaternions to other representations are needed.

First, in deciding whether a product is flipped, orientation representation of the product is converted into a rotation matrix to determine the direction of the transformed z axis. Second, though using Euler angles (or RPy specifically) introduces many problems (like quaternion to RPy is not unique and have singularity issues), yet it is more readable and easier for debugging. In our solutions, poses of products on the tray is represented with a yaw angle. A Boolean variable indicates whether the product is flipped.

4.4. Sensor Choices

The scoring metrics [24] of ARIAC favor solutions whose sensors cost less. In our 2019's solution, only a laser scanner and a logical camera were used in our entry. And in 2020's

solution, 9 logical cameras were used.

By using fewer sensors, we risk not having enough information about the environment. As an example, in our solution for ARIAC 2019, no sensor was put on top of the bins. The robot arms had to search or fumble around in the bin to grasp a product or to find out that the bin is empty. The significantly lower sensor cost, as compared with other teams, turned out to benefit us overall.

5. Conclusion

Considering the amount of effort we put in, our team excelled in producing an agile and well-functioning system. Our submissions proved to be competitive, placing among the top three in both 2019 and 2020. We have summarized the approaches we took, including our task and motion planning architecture, pose representations, and the choices of sensors. Our system's failure modes, which we learned the hard way, suggest that it is important for systems to be able to agilely recover from errors and continue. In sharing our lessons from our participation, we hope it will be of benefit to future ARIAC competitions and the larger robotics community.

Acknowledgement

The authors thank the organizers of the ARIAC for providing opportunities, scoring and feedbacks for our submissions: Anthony Downs, Dr. William Harrison, Dr. Zeid Kootbally, Dr. Craig Schlenoff from NIST, and Shane Lorentz from OSRF as well as the expert judges for the final round. This work is supported by NSF awards IIS-1734419 and IIS-1845888.

References

- [1] Atkeson, C.G., Benezon, P.W.B., Banerjee, N., Berenson, D., Bove, C.P., Cui, X., DeDonato, M., Du, R., Feng, S., Franklin, P., Gennert, M., Graff, J.P., He, P., Jaeger, A., Kim, J., Knoedler, K., Li, L., Liu, C., Long, X., Padir, T., Polido, F., Tighe, G.G., Xinjilefu, X., 2018. What Happened at the DARPA Robotics Challenge Finals. Springer International Publishing. pp. 667–684.
- [2] Björnsson, A., Jonsson, M., Johansen, K., 2018. Automated material handling in composite manufacturing using pick-and-place systems – a review. *Robotics and Computer-Integrated Manufacturing* 51, 222–229.
- [3] Brooks, R.A., 1983. Planning collision-free motions for pick-and-place operations. *The International journal of robotics research* 2, 19–44.
- [4] Chitta, S., Sukan, I., Cousins, S., 2012. Moveit! *IEEE Robotics & Automation Magazine* 19, 18–19.
- [5] Chong, J., Ong, S., Nee, A., Youcef-Youmi, K., 2009. Robot programming using augmented reality: An interactive method for planning collision-free paths. *Robotics and Computer-Integrated Manufacturing* 25, 689–701.
- [6] Eppner, C., Höfer, S., Jonschkowski, R., Martín-Martín, R., Sieverling, A., Wall, V., Brock, O., 2018. Four aspects of building robotic systems: lessons from the amazon picking challenge 2015. *Autonomous robots* 42, 1459–1475.
- [7] Feng, S., Whitman, E., Xinjilefu, X., Atkeson, C.G., 2015. Optimization-based full body control for the darpa robotics challenge. *Journal of Field Robotics* 32, 293–312.
- [8] Gerkey, B., Vaughan, R.T., Howard, A., 2003. The player/stage project: Tools for multi-robot and distributed sensor systems, in: *Proceedings of the 11th international Conference on Advanced Robotics*, pp. 317–323.
- [9] Han, S.D., Feng, S.W., Yu, J., 2020. Toward fast and optimal robotic pick-and-place on a moving conveyor. *IEEE Robotics and Automation Letters* 5, 446–453.
- [10] Han, S.D., Stiffler, N., Krontiris, A., Bekris, K., Yu, J., 2017. High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods, in: *Robotics: Science and Systems*.
- [11] Han, S.D., Stiffler, N.M., Krontiris, A., Bekris, K.E., Yu, J., 2018. Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps. *The International Journal of Robotics Research* 37, 1775–1795.
- [12] Harrison, W., Downs, A., Schlenoff, C., 2018. The agile robotics for industrial automation competition. *AI Magazine* 39, 77.
- [13] Hawkins, K.P., 2013. Analytic inverse kinematics for the universal robots ur-5/ur-10 arms. Technical Report. Georgia Institute of Technology.
- [14] Huang, B., Han, S.D., Boularias, A., Yu, J., 2020. Dipn: Deep interaction prediction network with application to clutter removal. *arXiv preprint arXiv:2011.04692*.
- [15] Islam, F., Salzman, O., Agarwal, A., Likhachev, M., 2020. Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt, in: *Robotics: Science and Systems*.
- [16] Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., et al., 2015. Team ihmc's lessons learned from the darpa robotics challenge trials. *Journal of Field Robotics* 32, 192–208.
- [17] Kaipa, K.N., Kankanhalli-Nagendra, A.S., Kumbla, N.B., Shriyam, S., Thevendria-Karthic, S.S., Marvel, J.A., Gupta, S.K., 2016. Addressing perception uncertainty induced failure modes in robotic bin-picking. *Robotics and Computer-Integrated Manufacturing* 42, 17–38.
- [18] Kootbally, Z., Kramer, T.R., Schlenoff, C., Gupta, S.K., 2018. Implementation of an ontology-based approach to enable agility in kit building applications. *International Journal of Semantic Computing* 12, 5–24.
- [19] Krontiris, A., Bekris, K.E., 2015. Dealing with difficult instances of object rearrangement, in: *Robotics: Science and Systems*.
- [20] Krotkov, E., Hackett, D., Jackel, L., Perschbacher, M., Pippine, J., Strauss, J., Pratt, G., Orłowski, C., 2017. The darpa robotics challenge finals: Results and perspectives. *Journal of Field Robotics* 34, 229–240.
- [21] LaValle, S.M., 2006. *Planning algorithms*. Cambridge university press.
- [22] Lozano-Pérez, T., Mazer, E., Jones, J.L., O'Donnell, P.A., 1989. Task-level planning of pick-and-place robot motions. *Computer* 22, 21–29.
- [23] Moghaddam, M., Nof, S.Y., 2016. Parallelism of pick-and-place operations by multi-gripper robotic arms. *Robotics and Computer-Integrated Manufacturing* 42, 135–146.
- [24] NIST Engineering Laboratory Intelligent Systems Division, . Ariac rules. <https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation-competition/rules>.
- [25] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y., 2009. Ros: an open-source robot operating system, in: *ICRA workshop on open source software*, Kobe, Japan. p. 5.
- [26] Wurman, P.R., D'Andrea, R., Mountz, M., 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29, 9–9.
- [27] Zeng, A., Song, S., Yu, K.T., Donlon, E., Hogan, F.R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., et al., 2018. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching, in: *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE. pp. 1–8.