Summer2023

RWA67

Building a Manufacturing Robot Software System

August, 2023

Students:
Shreejay Badshah, Arshad Shaik, Tej Kiran, Tim Sweeney, Dhinesh Rajasekaran, Ian Serbin

Instructors:
Dr. Zeid Kootbally
Dr. Craig Schlenoff

Group:
Group 2

Course code:
ENPM663

Contents

List of Figures

1 Introduction

This report summarizes the work done in RWA67 for Building a Manufacturing Robot Software System (ENPM663). RWA67 comprises a Competitor Control System (CCS) that can handle insufficient part, and faulty part challenges in ARIAC competition.

ARIAC (Agile Robotics for Industrial Automation Competition) is hosted by NIST (National Institute of Standards and Technology) to test the agility of the industrial robot system. The purpose of ARIAC is to make industrial robots more autonomous, productive with minimum human input.

In this competition, several types of parts; sensors, pumps, regulators and batteries; in several colors; red, blue, green, orange and purple; are provided in a Gazebo simulation environment to complete orders. The order type can be for kitting, assembly, or kitting and assembly. At competition start, parts are introduced into the environment using a conveyor belt and then are placed on bins for the required orders. A floor robot is provided in the environment to pick parts from the conveyor belt and place them on bins. There 8 similar bins in the environment. After receiving all the parts and storing them on the bins, the floor robot then again picks these parts from the bins and places them on an automated guided vehicle (AGV) tray to fulfill required orders. The floor robot also has a gripper type that can pick and place trays on AGVs before putting parts on them. The fulfilled order is then submitted to the warehouse. Once all orders have been kitted/assembled and shipped to the warehouse, the ARIAC competition is ended. Competition score is computed and determined based on how successful the CCS was at filling orders and managing agility challenges.

Several challenges are introduced in the competition to test the agility of the competitor control system. These challenges are:

1. Faulty Part
2. Insufficient Part
3. High Priority Order.
4. Flipped Part

The focus of this RWA is to address faulty part and insufficient part challenge to complete kitting orders.

2 Problem Statement

For this course the problem was to test the agility of an industrial floor robot to complete kitting tasks with the challenges of:

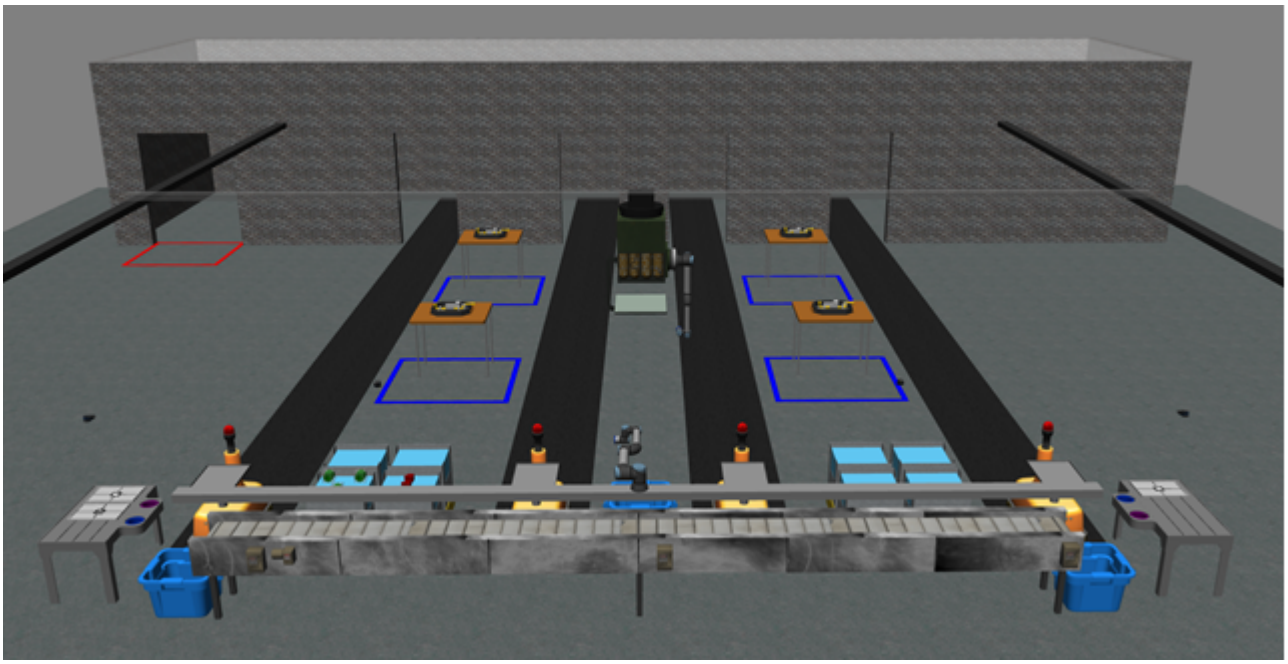


Figure 1: ARIAC Arena

1. Faulty part challenge:
   In this challenge a part placed on certain quadrant of a tray, placed on an AGV, can be deemed faulty. In case a faulty part is detected then CCS must discard this part in discard bin to avoid a penalty.

2. Insufficient parts challenge:
   Sometimes in the competition there are not enough parts to fulfill an order. In this case the order still must be submitted with as many parts available as possible to avoid a penalty.

3 Software Architecture

The software architecture for the project implementation is depicted in Figure 2.

Order Manager: The heart of the architecture is the 'order manager', which manages different services, manages communication with ARIAC, communicates with the motion planning node, etc. A detailed description of the 'order manager' is given in the subsequent corresponding section 6 further down the document.

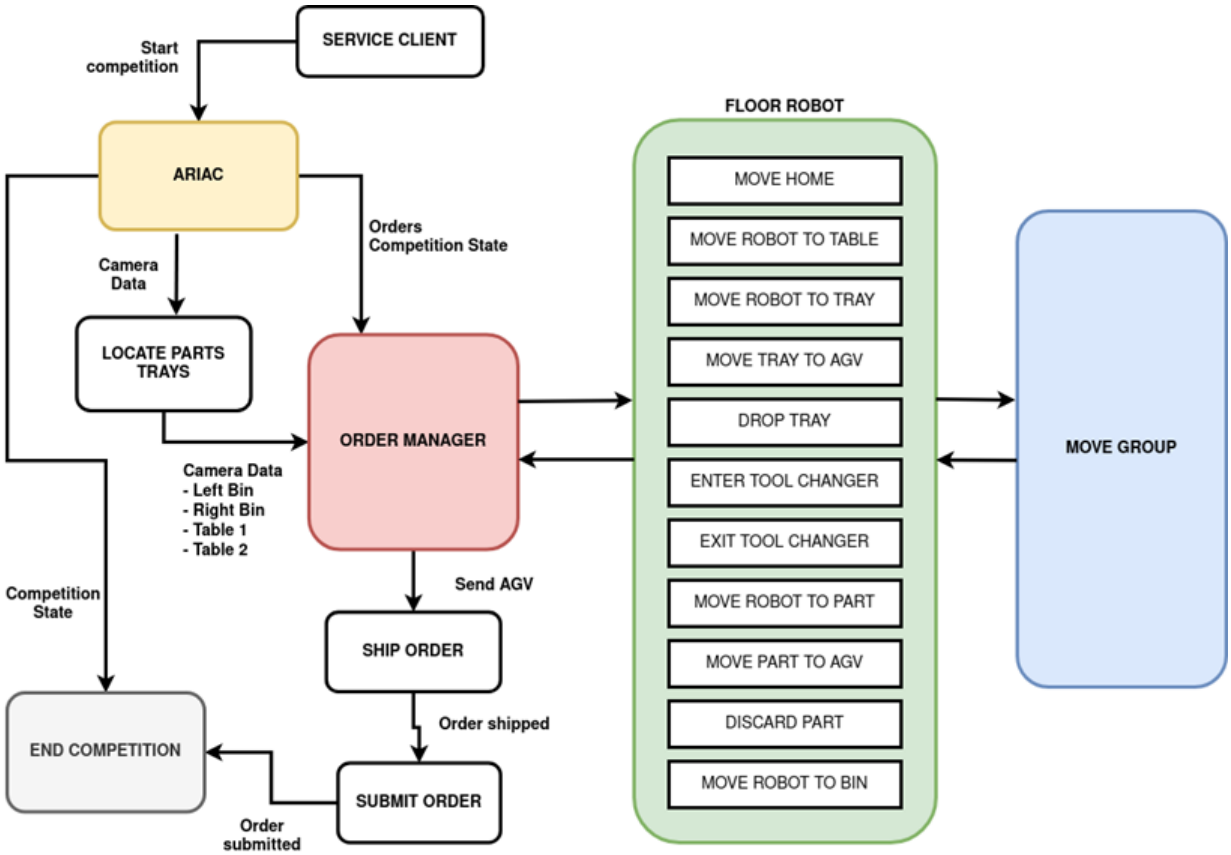


Figure 2: Software Architecture

FLoor Robot: The floor robot node's primary function is to communicate with move group node to generate a motion plan for the robot. The floor robot instantiates services that allow for high level movements to be broken down into smaller movements and communicated with the Move Group. Functionalities of each of the services are as described below:

- Move Home: Moves the floor robot to the home position (center of the rail)
- Move Robot to Table: Moves the robot to the specified tray table/gripper changer station
- Move Robot to Tray: Moves the end effector of the robot to the specified tray
- Drop Tray: Drops the tray
- Enter Tool Changer: Enter the specified tool changer (part or tray gripper) at the specified table
- Exit Tool Changer: Exits the specified tool changer
- Move Robot to Part: Moves the robots end effector to the specified part
- Move Part to AGV: Once the part is picked up, move it to the specified AGV
- Discard Part: Drops the carried part

- Move Robot to Bin: Moves the robot to the center trash bin

Locate Parts and Trays: Locates the part and tray locations/orientations, as determined by the camera sensors.

Service Client:Begins the competition.

Ship Order: Moves the specified AGV to the warehouse.

Submit Order: Submits a transported order for grading.

End Competition: Ends the competition.

Terminologies associated with the architecture are defined below:

Competitor Control System (CCS): The competitor control system (CCS) is the software that is provided by competitors. The CCS is responsible for communicating with the competition environment and executing the tasks.

Order: An order is an instruction containing information on a task (Kitting Task, Assembly Task, or Combined Task,). Each task consists of at least one part of a specific color and type.

Part: Parts are used during pick-and-place operations. There are four available parts (battery, pump, regulator, and sensor) and each part can be one of five possible colors (red, green, blue, orange, and purple).

Trial: Each run of the competition is called a trial. The configuration for that trial is defined by a configuration file (YAML). Competitors do not and must not directly read trial files but needs to use topics and services provided by the competition environment.

Kitting Task: Kitting is the process which groups separate but related parts as one unit. For a kitting task, competitors are expected to:
-Place a kit tray onto one of the four AGVs. -Place parts onto that kit tray in a specific quadrant. -Direct the AGV to the warehouse. -Evaluate the submitted kit for scoring.

Launch file Setup: A single ROS2 launch file is created that starts the competition and runs all the required packages/ nodes as shown below:

```
-moveit
-included_launch
-service_client_exe_node
-locate_parts_trays_exe_node
-order_manager_node
-ship_order_exe_node
-submit_order_node
-end_competition_node
-robot_commander_exe_node
```

4 Package Contents

The project contains two packages, namely:

1. 'Robot msgs'
2. Main Competitor Package

The number of lines of code, for this project is approximately: 5053 The packages tree is shown captured in the below figures:

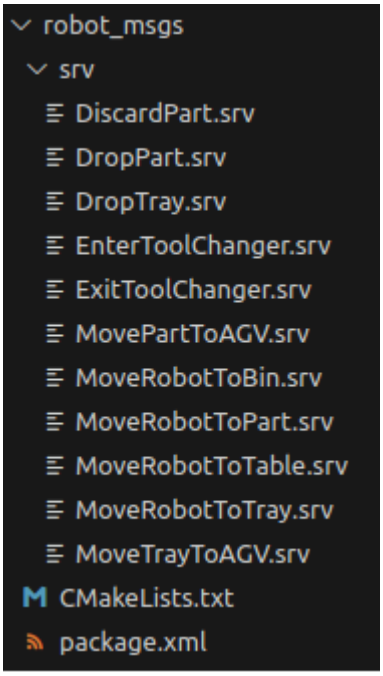


Figure 3: Custom Service Messages

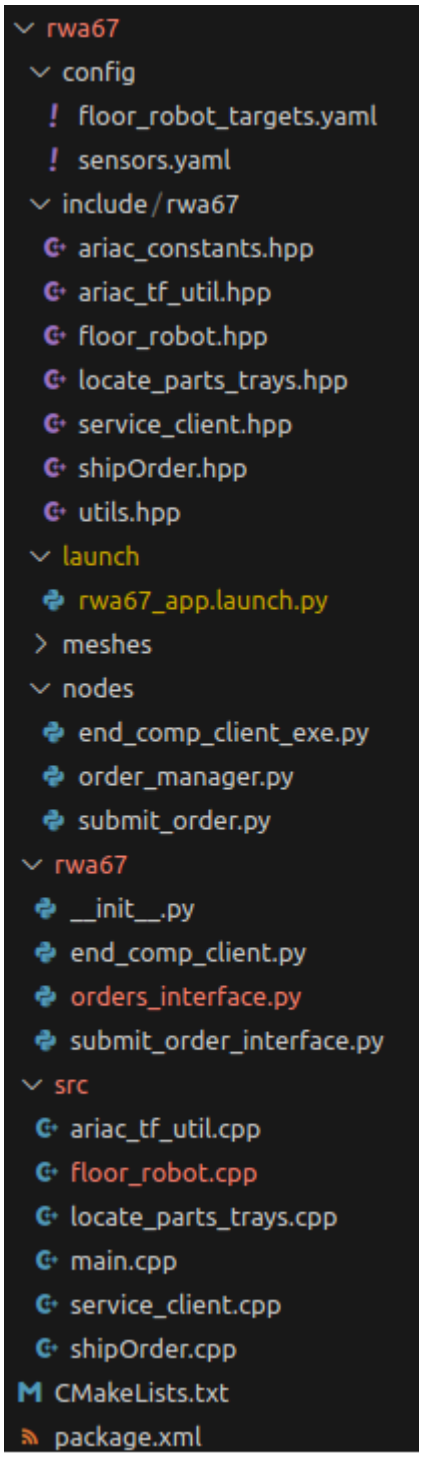
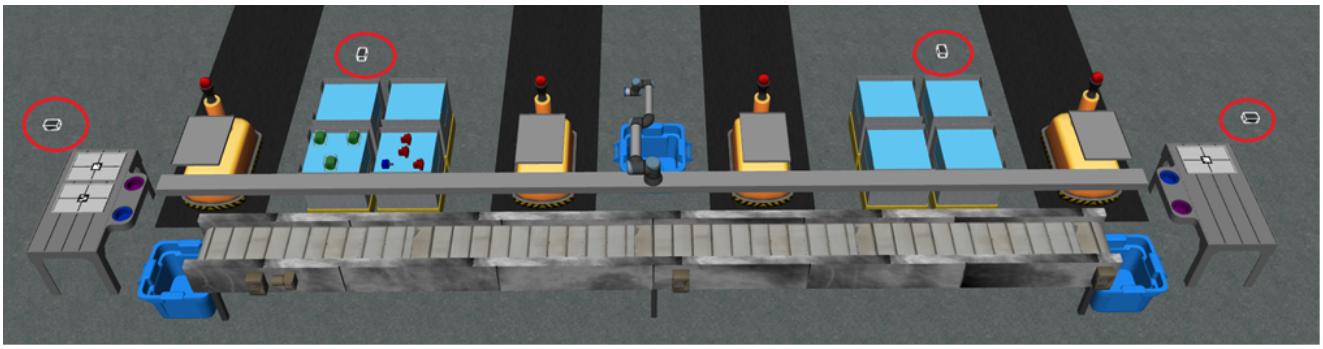


Figure 4: Main Competitor Package

5 Environment Sensors



```
sensors:
  left_bins_camera:
    type: advanced_logical_camera
    pose:
      xyz: [-2.286, -2.96, 1.8]
      rpy: [pi, pi/2, 0]

  right_bins_camera:
    type: advanced_logical_camera
    pose:
      xyz: [-2.286, 2.96, 1.8]
      rpy: [pi, pi/2, 0]

  kts1_camera:
    type: advanced_logical_camera
    pose:
      xyz: [-1.3, -5.8, 1.8]
      rpy: [pi, pi/2, pi/2]

  kts2_camera:
    type: advanced_logical_camera
    pose:
      xyz: [-1.3, 5.8, 1.8]
      rpy: [pi, pi/2, -pi/2]
```

Figure 5: Environment Sensors

The environmental sensors provide the CCS with information necessary to build the world model (part locations, tray locations). As highlighted with red circles, in the figure 5, the following are sensors deployed for the competition.

1. 4x Advanced Logical Sensors
   (a) Provide tray id and pose
   (b) Provide part pose, color, and type
2. AGV Tray Sensors
   (a) Located on each AGV
   (b) Provide quality information of tray and parts

6 Order Manager

The Order Manager node is the heart of the competition package. It monitors and controls the flow of order fulfilment. Based on the priority, the current order is executed until it is

fulfilled. It interfaces with other different nodes including 'Locate parts and trays', 'ship order' and other nodes as shown in the block diagram.

The functionalities of the 'order manager' are depicted through following pseudo code:

```
Fullfill_order:
order = current_order
tray = order.tray
agv = order.agv

tray_pose = search_tray_inventory(tray)
move_robot_to_table()
change_to_tray_gripper()
grab_tray()
place_tray_on_agv()
remove_tray_from_inventory()

move_robot_to_table()
change_to_part_gripper()
for (part in list_of_parts):
part_pose = search_part_inventory(part)
    If (part_pose == None):
   Continue  #part not found... Moving on
    place_part_on_tray()
    perform_quality_check()
    If (faulty):
           Move part to bin
           Add part back to list_of_parts
```

The order manager is subscribed to seven different topics, which can be grouped into three categories:
`/ariac/competition_state`
Begins fulfilling orders

`/ariac/orders`
Adds orders to the queues

`/left_bin_camera_data`
`/right_bin_camera_data`
`/kitting_tray1_camera_data`
`/kitting_tray2_camera_data`
Records the poses of parts and trays. These have been obtained by `locate_parts_trays`, translated to world pose, then republished.

7 Trial Files

Several trial files were developed to test the agility of the CCS. The purpose of these trial files is to create as many challenges in the competition as possible to observe whether the CCS is capable of handling these challenges without crashing.

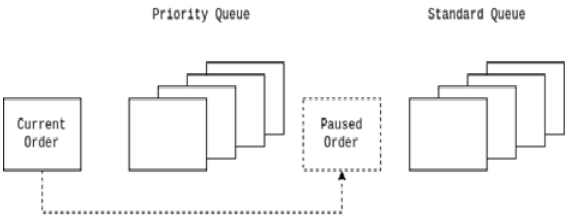


Figure 6: Order Queue

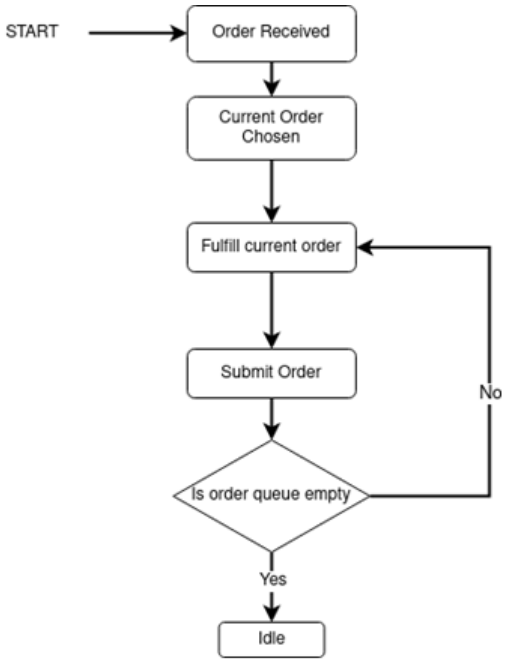


Figure 7: Order Manager

The CCS was tested on the following trial files:

1. `rwa67summer2023.yaml`
   This trial file was provided as a part of RWA67. The trial consists of 3 kitting orders, a faulty part challenge, and an insufficient part challenge.

2. `rwa67_summer2023_SB_test1.yaml`
   This trial file has 4 kitting orders with multiple faulty parts. Since the faulty parts are discarded the CCS also has to handle insufficient part challenge.
   The purpose of this trial file is to handle faulty and insufficient part challenges for multiple orders.

3. `rwa67_summer2023_TS_test1.yaml`
   This trial file has 5 similar parts in the environment for 1 kitting order. Out of 5 parts, 3 are faulty, hence, creating insufficient part challenge.
   The purpose of this trial file is to handle multiple faulty and insufficient part challenges in a

single order.

Figure 8: rwa67_summer2023_SB_test1.yaml

8 Results

The trial files were repeatedly used to check the robustness of trial file. Results of running above configuration files are as below:

`rwa67_summer2023_SB_test1.yaml:`
completion time: 162 sec
Score: 39

`rwa67_summer2023_TS_test1.yaml:`
completion time: 97 sec
Score: 9

Scores and completion time for the both trial files remained similar throughout the testing cycle validating CCS designed to handle two different ARIAC challenges.

9 Difficulties Faced

Numerous difficulties were faced to complete this RWA since it required a number of services for robot actions, moveit package to simulate robot movement in the gazebo environment, proper communication between services (written in C++) and clients (written in python)

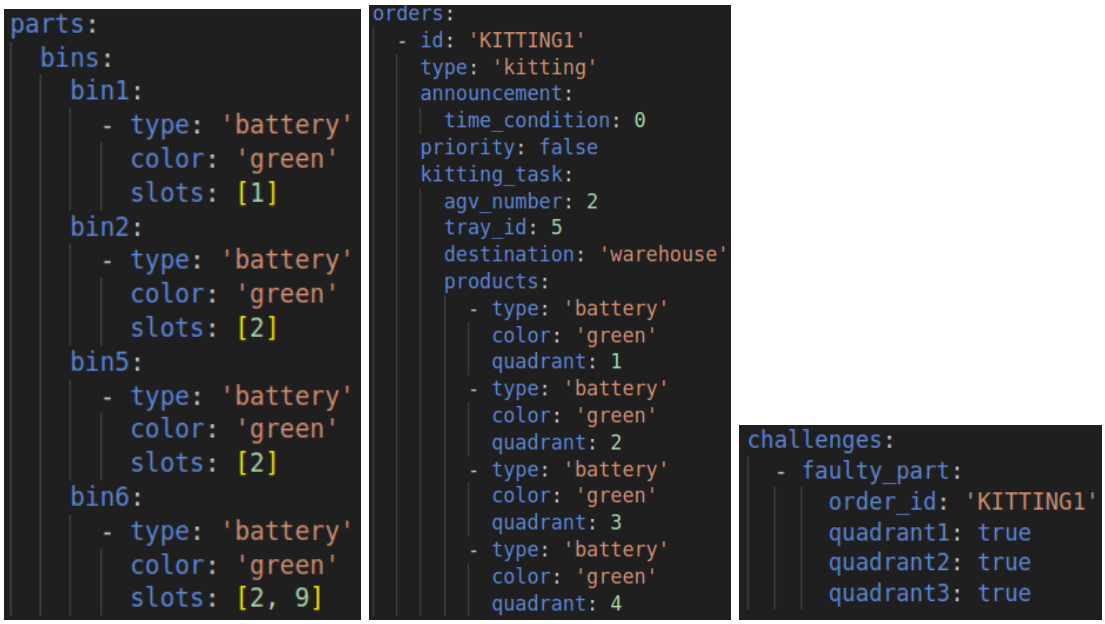


Figure 9: rwa67_summer2023_TS_test1.yaml

and many other programming challenges to create a robust CCS.
Below are some difficulties that were faced and the solutions to overcome them:

1. Breaking down custom services:
   As the floor robot has to pick and place, both parts and trays - by changing gripper type, custom services had to be broken down for a smooth pick and place movement. To drop parts and trays on their new placed, two separate services were called and used - one before the part is dropped and another after the part is dropped. The purpose of breaking down the services here was to deactivate the gripper at a safe position to drop a part/tray.

2. Using appropriate moveit method to add collision objects in the environment:
   Using default addCollisionObject() method to add models in the planning scene was less useful when it came to detaching the object from the planning scene. Because of the nature of the method to run asynchronously, sometimes parts/trays were not added to the planning scene before they were dropped on their new placed. To avoid a crash in the program, when dropping a part/tray, applyCollisionObjects() method was used instead. This method made sure that the model was available in the planning scene right when the method itself was called.

3. Adjusting waypoints:
   Waypoints in service callback methods used to pick up parts (move_robot_to_part_()) were updated with trial and error so that each type part didn't collide with other parts after being picked up from its place on a bin.

10 How can the results be made better?

Due to the time constraints of the class, this CCS is not a completed CCS ready for competition in the ARIAC challenge. CCS agility could be improved by:

1. Tray and Part inventories are determined at start. Ideally, part and tray locations should be continuously updated (bumped parts, dropped parts)

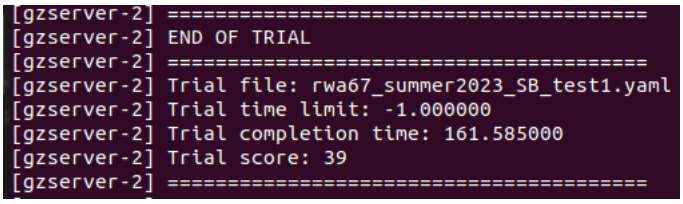


Figure 10: Result of rwa67_summer2023_SB_test1.yaml

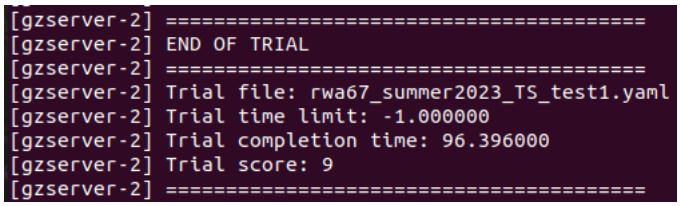


Figure 11: Result of rwa67_summer2023_TS_test1.yaml

2. Integrate ceiling robot to reach all parts in bins
3. Attach trays/parts to AGVs in planning view. Current implementation would cause collisions when using the AGV for a second time.
4. Finish Priority order logic
5. Create order logic for assembly stations

11 Conclusion

In summary, the project accomplishes its objectives of performing the motion planning to complete up to four orders (one per AGV), while demonstrating the ability to sufficiently handle insufficient parts and faulty parts.

12 Contributions to the project

- Shreejay Badshah: Creating services and relevant servers, agility testing, fine-tuning moveit parameters.
- Arshad Shaik: Testing, launch files, presentation and LaTex report generation
- Tej Kiran: Code implementation - Qualitycheck, etc.
- Tim Sweeney: Worked on the logic of the Order Manager, the End Competition node, created test-trials
- Dhinesh Rajasekaran: Testing, documentation
- Ian Serbin: Code implementation - services, order manager

13 Acknowledgements

We would like to thank Prof.Zeid Kootbally and Prof.Craig Schlenoff for their valuable guidance and feedback and their help in completing this project.

14 Resources

- https://ros.org
- https://docs.ros.org/en/galactic/Tutorials.html
- https://stackoverflow.com
- https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation-competition
- `https://github.com/sbadshah96/ENPM663_RWA`