University Of Maryland
College Park

Final Project Report

809 B-Building a Manufacturing Robot Software System

Group 4
Aditya Goswami - 116951968
Markose Jacob - 117000269
Nalin Das - 116698290
Saumil Shah - 116745338
Varun Asthana - 116696500

Dec 05, 2020

Abstract

The emphasis of this project is to understand the importance of agility for a robot working in a warehouse. Implement the logics to achieve agility using the object oriented programming in C++, ARIAC interface, Moveit and ROS. For the scope of this project, the warehouse environment provided in ARIAC 2020 has been used.

Key words : ARIAC, Moveit, Object Oriented Programming, Agility.

Contents

List of Figures

1 Project Objective

The project revolves around the Agile Robotics for Industrial Automation Competition (ARIAC) which deals with building of part kit(s) based on an order (a set of shipments) using the parts available in the environment. These parts are either present on static bins/shelves or on a moving conveyor belt. Once the kit has been built over an Autonomous Guided Vehicle (AGV) then the AGV has to be shipped for final delivery. Each shipment is considered to be fully complete only when all the desired, non-faulty, parts are placed on the AGV tray in the correct pose, and the AGV is shipped. The competition also includes some agility challenges like moving obstacles, faulty parts, sensor blackout, faulty gripper, part flipping, and high priority order. The final implementation of the project includes but not limited to ROS/C++programming, coordinate transformation with TF library, trajectory planning using MoveIt interface, and strategies to tackle all agility challenges for an autonomous system.

2 Environment Setup

The ARIAC environment comes with a basic set-up which includes a conveyor belt, bins, robot (gantry with two UR10 arms attached), and 2 AGVs. Figure (1) is an example of how the environment looks.

Figure 1: Initial Environment Setup

A number of different types of sensors in the environment that have been used are:

- **Logical Camera**: It is used to read the part type and the pose of the

part in the environment, relative to the camera frame.This data is used to pick and place the part. A total of 17 logical cameras have been used (4 over bins, 1 over conveyor belt, 10 over shelves, and 2 over AGVs).

- **Break Beam Sensor**: It is used to detect the presence of moving obstacles in each of the 4 aisles (on the shelves side). To detect the position of the moving obstacle in an aisle, we have used 6 sensors in each aisle.
- **Quality Control Sensor**: It is used to detect if the part placed on the AGV tray is faulty or not. 2 such sensors are used, 1 over each AGV.

3 Challenges

In ARIAC 2020, the following challenges will occur during kit building, so the system needs to be agile and generic to handle all of them. Any combination of the challenges can be present.

3.1 Faulty Part

In this challenge, the task is to avoid using faulty parts to build kits. The quality checking can be done by subscribing to the quality control sensors above each AGV- the topic will provide the pose of faulty parts in its scope. The robot has to replace the faulty one with a new part. Details about our approach will be described in the Approach section

3.2 Flip Part

Of five types of parts in ARIAC kit building (disk, gear, piston-rod, pulley, and gasket), only pulley might be required to be flipped. For such part, the roll value will be described as 'pi' in the .yaml file.

3.3 High Priority Order

The challenge is that while the robot is building a kit, another order with high priority comes up which has to be completed as soon as possible. The challenge is to complete the high priority order and then complete the last ongoing order.

3.4 Faulty Gripper

This problem focuses on a situation when gripper becomes faulty while building a kit. The gripper of the robot arm might malfunction while placing the part in the tray, and the part would be dropped at some random position on the AGV tray. Dropped parts will still end up in the tray but in the wrong pose. We are required to re-position any such part correctly.

3.5 Sensor Blackout

In this challenge, communication with most of the sensors will be lost temporarily, referred to as a "sensor blackout". At the start of the competition, the sensors will be publishing data normally. With a particular condition, most of the sensors will stop publishing for a fixed period of time. This applies to user-defined sensors and sensors that are present by default in the environment. We needed to take necessary actions i.e. wait until the sensors are working again before continuing, or continue building kits during the blackout.

3.6 Moving Obstacles

This agility challenge consists of 2 humans moving on the warehouse floor. For this challenge we needed to find out which aisles are occupied by the moving obstacles, get the location of moving obstacles within the aisles, get the configuration of the shelves to be able to find gaps between shelves, and plan a path to reach the shelf with parts. Figure (2) and Figure (3) shows the environment with moving obstacles.

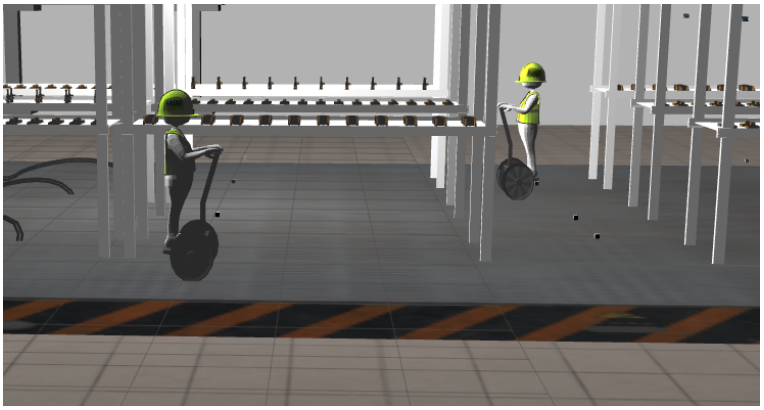

Figure 2: Human Obstacles in the environment

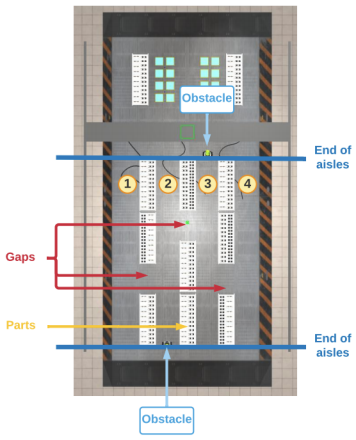

Figure 3: Aisle Layout for Moving Obstacles

4 Challenge Solution

4.1 Our Strategy

As the competition starts, the ARIAC environment is initialized with all the sensors, the robot (gantry and the arms) and parts as discussed in the environment setup section. After all the sensors are up and running, orders with shipments are received. Information such as order number, type of order, number of shipments, and AGV name is read. Along with this information, products information is read as well and then stored in a data structure, this helps in keeping track of all the information about every order and the required parts in the shipment. Following are some strategies that we employed in ARIAC challenge:

1. We kept track of all the parts that appeared on the conveyor belt. We computed the velocity of the belt and kept a track of ROS TIME for each part (time at which the part first appeared under the logical camera), to estimate their position on the belt at any instance. This approach helped to pick the part which appeared first on the belt. In case we miss to pick the part then we query for the next available part. This helped us to be certain that the parts are not exhausted before their requirement, as we are not constrained to pick the part from their spawning location. An example is shown in Figure (4)

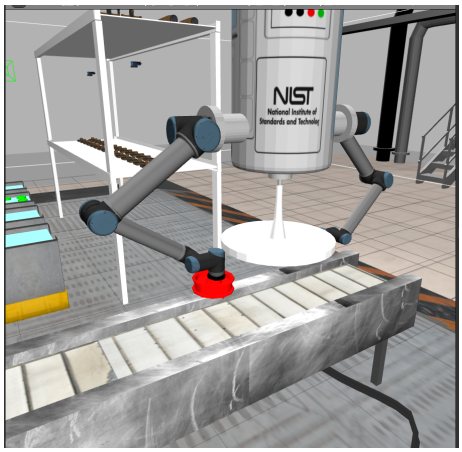

Figure 4: Robot Picks up the part from the end of the Conveyor belt.

2. Highest priority is given to conveyor parts. In case the conveyor part is not yet available, robot proceed to pick a static part. This save the total execution time. But in case moving obstacles are present in the environment, we first deliver all the conveyor parts, as the wait and escape time is high with moving obstacles. This may lead to a situation where the conveyor parts have already been exhausted, if not executed first.

3. In order to resolve the issue of picking the parts from each and every area on the AGV tray (especially top corner), we positioned our robot at an angle of 45 degrees.

4. Issue of wrong part orientation was resolved using quaternion mathematics in *placepart* function.We took into account the initial orientation of part at which it was picked by the robot, and used this to find the relative rotation needed to get the desired target orientation

5. The position of the robot is calculated dynamically according to the position of the part in the environment (with some pre-defined waypoints and offsets).

6. In case of the moving obstacles, the robot finds the best possible path to avoid collision with moving obstacles. Moreover, in case when the robot is in the aisle picking up a part but fails to do so in the fist attempt, then the robot moves back to the nearest gap in-order to avoid any possibility of a collision with the moving obstacle. There it waits for the obstacle to leave the area where the part has to be picked up from and then makes an attempt again to pick it. This is done only in the case where the part is in the aisle of moving obstacle to avoid collision. When there is no moving obstacle then the robot try multiple times to pick the part.

7. Finally, we keep checking the total time elapsed. If it has reached 498 simulation seconds, then we trigger the shipment of the AGVs with the partial order. This approach is useful in the worst case where the robot is not able to complete the order, a partial order fetches us some score whereas no order would fetch zero score.

4.2 Faulty Part Solution

The faulty part challenge is solved using a simple approach- where the part is detected as faulty, then picked again by the robot and discarded (thrown on the floor).

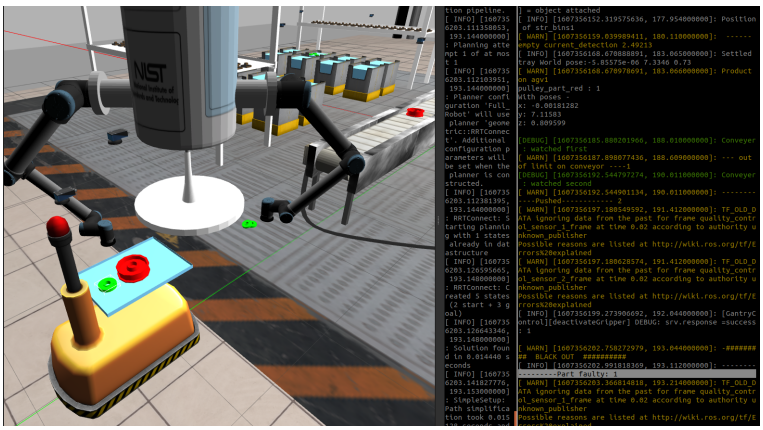

Figure 5: Checking Faulty Part and discarding it on floor.

Detection of the faulty part is done by subscribing to the quality control sensor which are placed on top of each AGV tray. Every time when a part is placed on the tray, output of the sensor is checked if the part is faulty or not. If the part is faulty then the robot picks the part up and drops it on the floor to discard it. Then the part data is added into the pending order list. Now the robot goes to fetch the new part of the same type (since it is following the pending order list) and the process is repeated until the part is found to be not faulty.

4.3 Flip Part Solution

For the challenge where the part is needed to be flipped (which is only pulley part), first we get roll value from the target pose of the part. If the roll value is equal to 'pi', then the part has to be flipped otherwise it does not require flipping before placing. If the part has to be flipped, it is picked up using the

left arm and the right arm gripper is attached to the part while left arm gripper is still attached (see the image below). Then the left arm gripper is deactivated leaving the part attached to right arm and thus the part has been flipped.

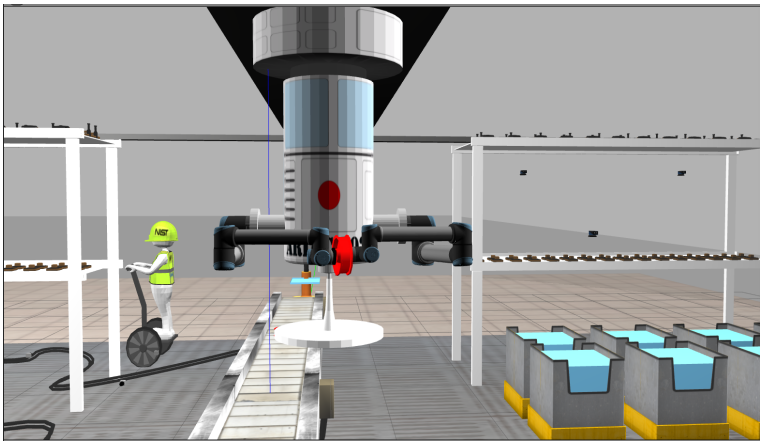

Figure 6: Robot flipping the pulley part

Difficulties faced: If the left arm picks up the pulley not from its center, then it becomes difficult for the right arm to attach to its center. Therefore to overcome this difficulty, we aligned the axis of the grippers on both arms. This ensured that no matter which way the left arm is attached to the pulley part, the right arm will also get attached to it in the same way from opposite side.

4.4 High Priority Order Solution

This challenge is solved by assuming that the second order is a high priority order. Any new order received is added to the pending order data structure. Here stack is used as the data structure. Once an order is pushed into the stack, the robot starts by picking the part at the top of the stack. This way whenever a high priority order is received then order executes the parts of the newly received order since it becomes the topmost element of the stack. This way we were able to indirectly execute high priority order first (although maximum priority is given to the conveyor parts, for which a separate stack is maintained).

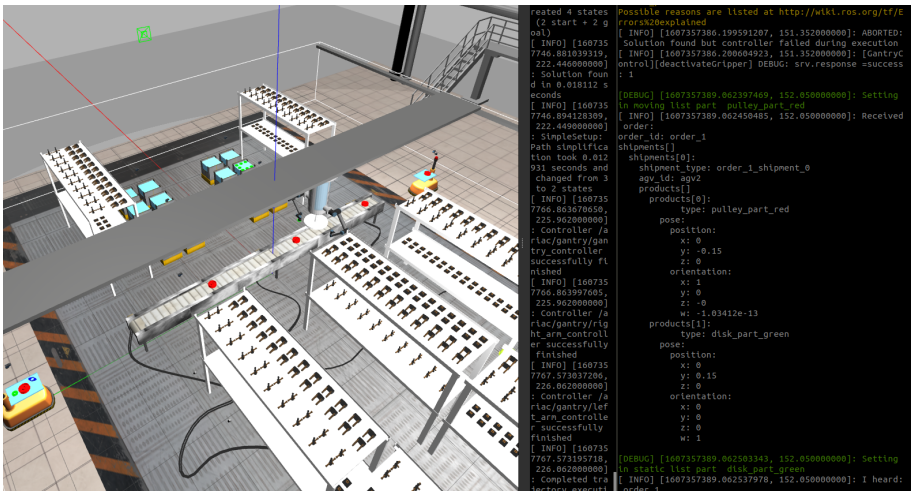

Figure 7: High Priority Order Received shown in the terminal

Difficulties faced: We faced a new challenge when a high priority order has to be built on an AGV which is already in use. This problem was resolved by adding a *clearAGV* function and is checked against the shipment number for the order being built. If a mismatch is found, then this function is called before executing the part of new shipment (high priority).

4.5 Faulty Gripper Solution

In faulty gripper challenge, when the part is dropped on the tray, we get the pose of the part by subscribing to the logical camera (1 over each AGV). Then we match the current pose of the part on the tray to the desired target pose of that part. If both poses are equal then the robot moves onto other parts that need to be picked. Otherwise in the case pose mismatch robot picks that part from the AGV tray and place it in the desired target location. The process is repeated until the current pose is equal to the desired target pose (within the allowed threshold).

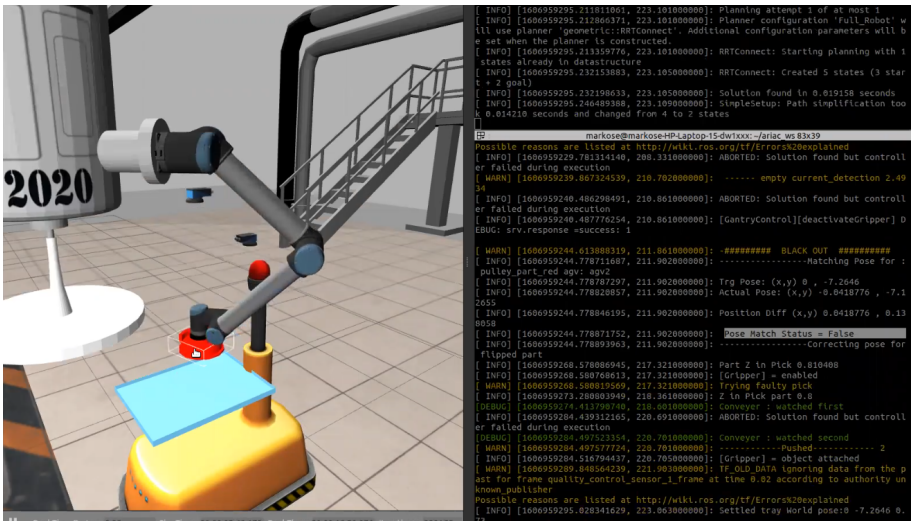

Figure 8: Faulty gripper where the Pose Match Status is *fasle*.

Difficulties faced: We encountered a problem during this challenge that if the part was wrongly placed somewhere in the upper corners of the tray, the arm had difficulty in picking up that part. This is where we positioned our robot at an angle of 45 degree to make the arm reach those corners of the tray.

4.6 Sensor Blackout Solution

The major hurdle in this challenge is to identify if a sensor blackout has occurred. This was resolved by having a counter in one of the callbacks for the quality sensor. If the value of the counter does not change then it means that a sensor blackout has occurred (a warning message is displayed on the terminal as shown in Figure (9)). The robot comes to a halt in its current position until the sensor blackout is over.

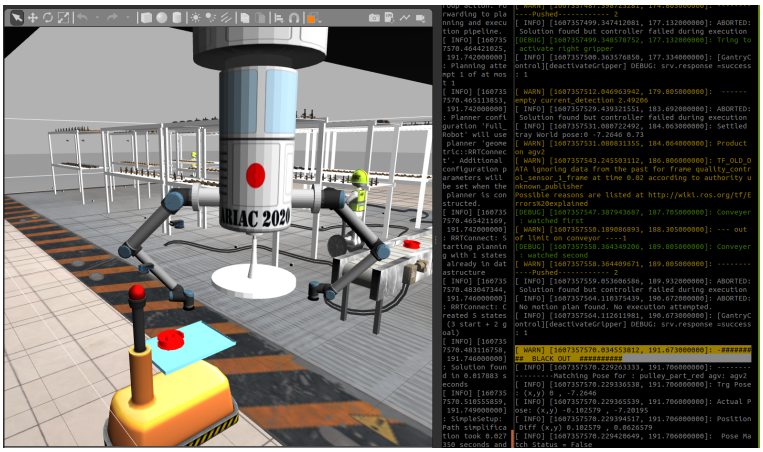

Figure 9: Sensor blackout

4.7 Moving Obstacle Solution

In moving obstacle challenge, our first goal is to identify if there is any moving obstacle at all on the floor (environment). To overcome this, we placed a bunch of breakbeam sensors in the environment (24 in total). If moving obstacles are present in any aisle, then one of the breakbeam sensor (of the respective aisle) will return *true*. This indicates that there are moving obstacles. For this to be noticed (to avoid the situation where the obstacle is not infront of any sensor), we wait for 5 simulation seconds after the start of the code.

Now to further narrow down which aisles have moving obstacles, we named the breakbeam sensors and used this information to detect which aisles are clear and which are not.

Robot moves to the closest shelf gap (if required intermediate gap is also used). Once the *"aisle clear for part pick"* flag is set, robot moves to pick the part. Once the part is picked then the escape behaviour is triggered.

We use the gap between the AGVs and the conveyor belt in case the moving obstacle is on either of the extreme aisles as shown in Figure (10).

Difficulties faced: Sometimes when the robot is not able to pick up the part in first attempt then it kept on trying to pick that part up. But this action is not favourable in this case since we have a moving obstacle coming back towards the location of the robot and will collide. Therefore, the robot goes back to the nearest safe gap and waits for the aisle to be cleared again and then attempts again to pick the part. Further we minimized the action of the re-positioning of the arms to reduce the time used and thus minimize the risk of collision. The robot brings its arms to their default position once it is on top

of the conveyor belt.

Figure 10: Use of the gap when obstacle is in extreme aisle

5 Results

The project has been successfully completed in due time. The package is built in a robust, generalized and agile fashion. The package handles the challenges like flipping parts, faulty gripper, faulty parts, sensor blackouts, high priority order, and moving obstacles very well. We were able to build the kit for different configuration files on different combinations of AGVs. Figure (11) shows the snippet of the final score we had when our code was tested on one of the final testing configuration of the environment.

```
Score breakdown:
<game_score>
        Total game score: [40]
        Total process time: [321.499]
        Arms collision?: [0]
<order_score order_0>
        Total order score: [16]
        Completion score: [16]
        Time taken: [321.498]
        Complete: [true]
        Priority: [1]
<shipment_score >
        Completion score: [16]
        Complete: [true]
        Submitted: [true]
        Product type presence score: [4]
        Product color presence score: [4]
        All products bonus: [4]
        Product pose score: [4]
        Delivered to correct agv: [true]
</shipment_score>

</order_score>

<order_score order_1>
        Total order score: [24]
        Completion score: [8]
        Time taken: [88.403]
        Complete: [true]
        Priority: [3]
<shipment_score >
        Completion score: [8]
        Complete: [true]
        Submitted: [true]
        Product type presence score: [2]
        Product color presence score: [2]
        All products bonus: [2]
        Product pose score: [2]
        Delivered to correct agv: [true]
</shipment_score>

</order_score>
```

Figure 11: Final Score

6 Future Work

- Orientation correction when placing with the right arm end effector
- Implement sensor blackout behavior instead of waiting
- When an high priority order is to be built on an AGV which is already in use, then instead of clearing the AGV we may re-utilize some of the existing parts.
- Update part pose periodically in the data structure for the static parts in the environment. As while picking a nearby part, at times robot may disturb the adjoining parts.
- Use of both arms to complete the order faster.

7 Contribution

Contribution of different team members in the overall delivery of the project is highlighted below.

7.1 Aditya Goswami

1. Challenge Flipping part -Pulley Flipper
2. Faulty part, Faulty gripper agility challenge
3. Report

7.2 Markose Jacob

1. Dynamic waypoints, finding gaps
2. Faulty part, faulty gripper agility challenges
3. Code testing

7.3 Nalin Das

1. Flip part, faulty part, faulty gripper agility challenges
2. Part Availability
3. Doxygen commenting and documentation

7.4 Saumil Shah

1. Order read and store as linked list,, High priority order, Moving obstacles, Sensor Blackout agility challenges
2. Part orientation correction on AGV tray
3. Conveyor part tracking and trigger signal to pick the part from the right location. Retry functionality if part missed to pick by allocating new part of same type from the tracked parts on the conveyor.
4. Overall code architecture
5. Debugging and rectification

7.5 Varun Asthana

1. Order read and store as linked list, High priority order, Moving obstacles, Sensor Blackout agility challenges
2. Check for time limit exceed and ship partially build orders
3. Part orientation correction on AGV tray
4. Next part allotment based on various checks of moving obstacle, high priority order, availability on conveyor and agv availability
5. Overall code architecture
6. Debugging and rectification

References

[1] https://github.com/usnistgov/ARIAC/tree/master/wiki/documentation

[2] Slides and class lectures

[3] https://ros-planning.github.io/moveit_tutorials

[4] http://wiki.ros.org/ROS/Tutorials