# **ENPM 809B** - Building a Manufacturing Robot Software System

## **Final Project Report**

Group 3:

Eashwar Sathyamurthy (UID - 116946148)
Mushty Sri Sai Kaushik (UID - 116917094)
Akwasi A. Obeng (UID - 11700801)
Shantam Bajpai (UID - 116831956)

## Table of Contents:

## List of Figures:

## Introduction

The ARIAC competition is a project that involves order fulfilment in a fully automated manufacturing environment. The competition includes a dual UR10 arm gantry robot that can move along the environment that has 11 product shelves including 3 dynamic shelves, 16 product bins and 2 AGVs. The order that needs to be fulfilled can contain any of the following parts in any quantity: piston, pulley, disk and gear. The part colors could be in red, green or blue. The environment also has moving obstacles in any of the four aisles.

The user can utilize information obtained from any of the following range of sensors that can be placed in the environment: logical camera, break beam sensor, quality control sensor, laser profiler, depth camera, proximity sensor, RGB-D camera. The competition also covers several agility challenges in order to simulate real world situations in manufacturing conditions like: faulty part, faulty gripper, flip part, high priority order, obstacle avoidance, sensor blackout and conveyor belt part pickup. The competition was done in ROS melodic and simulated using Gazebo 9.14

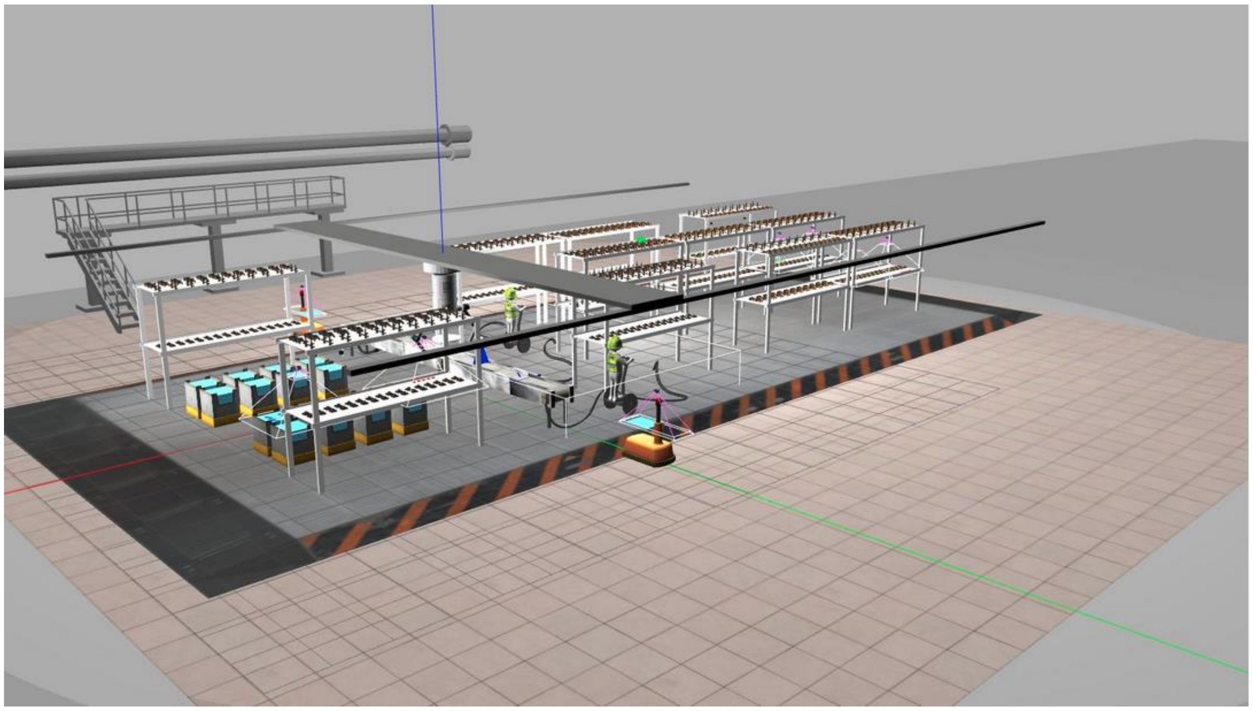


**Figure 1: ARIAC 2020 Competition**

## Task level planning

The ARIAC challenge has a primary objective of fulfilling an order in a manufacturing environment. In order to achieve the task of building the required kit, the tasks have been further broken down into four key tasks: read order, pick & place, read sensor output and plan path. These tasks are further broken down as shown in the figure below. We approached the fundamental tasks of performing the final broken down tasks and built up towards having the final kit built.

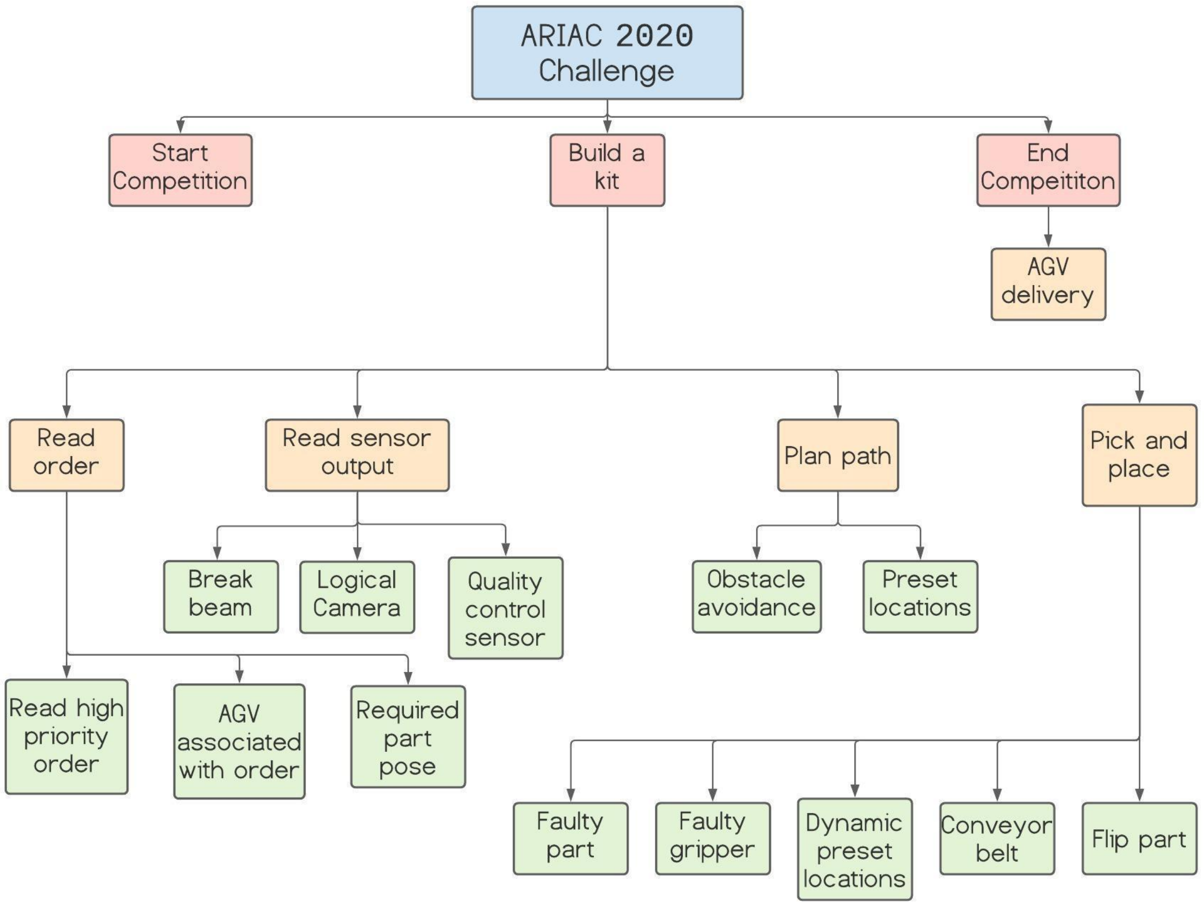


**Figure 2: Block Diagram of Task level planning**

## Architecture

We have used both reactive and hierarchical architectures depending on the agility challenge.

1. **Faulty Part:** We have used reactive architecture to remove the faulty part from the agv but to replace the part, we used hierarchical architecture as the gantry needs to plan to go near the part location.
2. **Faulty Gripper:** For faulty gripper, we used reactive architecture as it just needs to pick and place the part in the right position and orientation.
3. **Conveyor Belt:** Picking parts from the conveyor belt uses reactive architecture.

4. **High Priority order**: For high priority order, we used both reactive and hierarchical architecture
5. **Flip Part**: For flip part, we used reactive architecture.
6. **Obstacle Avoidance**: We used hierarchical architecture as obstacle avoidance involves path planning.
7. **Sensor blackout:** For sensor blackout, we using only **Plan↔Act** architecture, and sensing is inactive.

The below diagram provides a higher level implementation of solution in the form of a flowchart. The block process part is again represented by a flowchart considering all the agility challenges into consideration. The flowchart of each agility challenge is further provided in the agility challenges section.

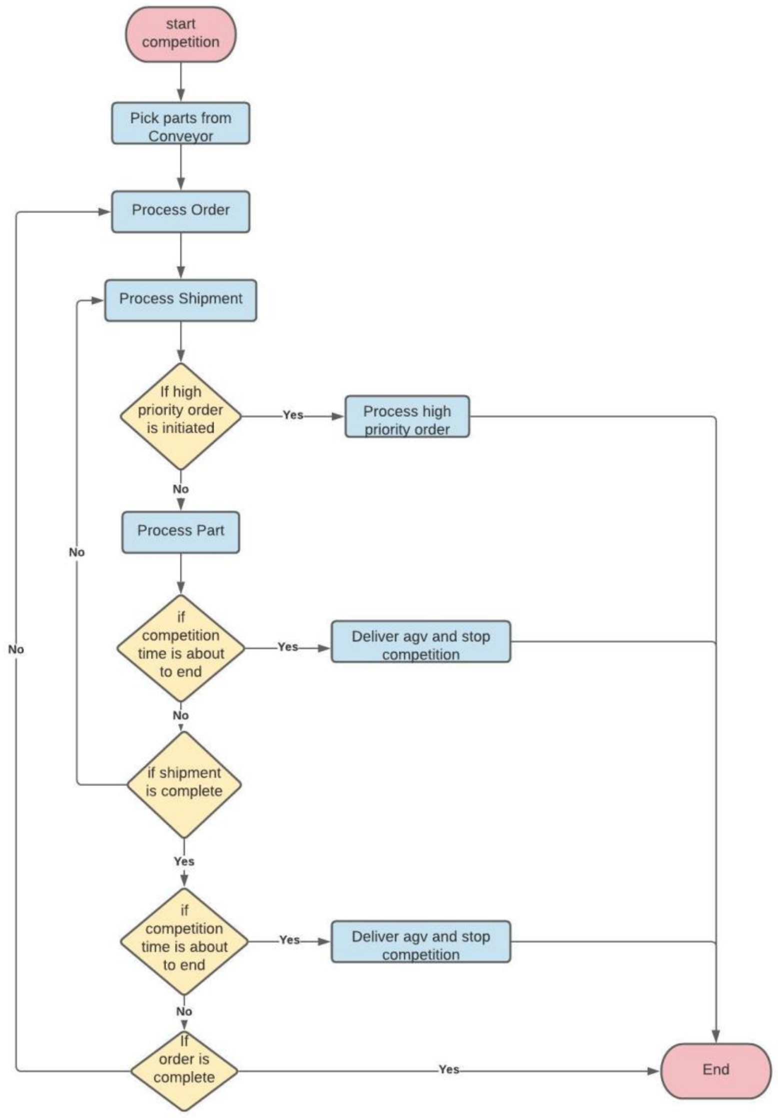

start competition
Pick parts from Conveyor
Process Order
Process Shipment
If high priority order is initiated
Yes
Process high priority order
No
Process Part
No
if competition time is about to end
Yes
Deliver agv and stop competition
No
No
if shipment is complete
Yes
if competition time is about to end
Yes
Deliver agv and stop competition
No
If order is complete
Yes
End

**Figure 3: Competition Implementation Flowchart**

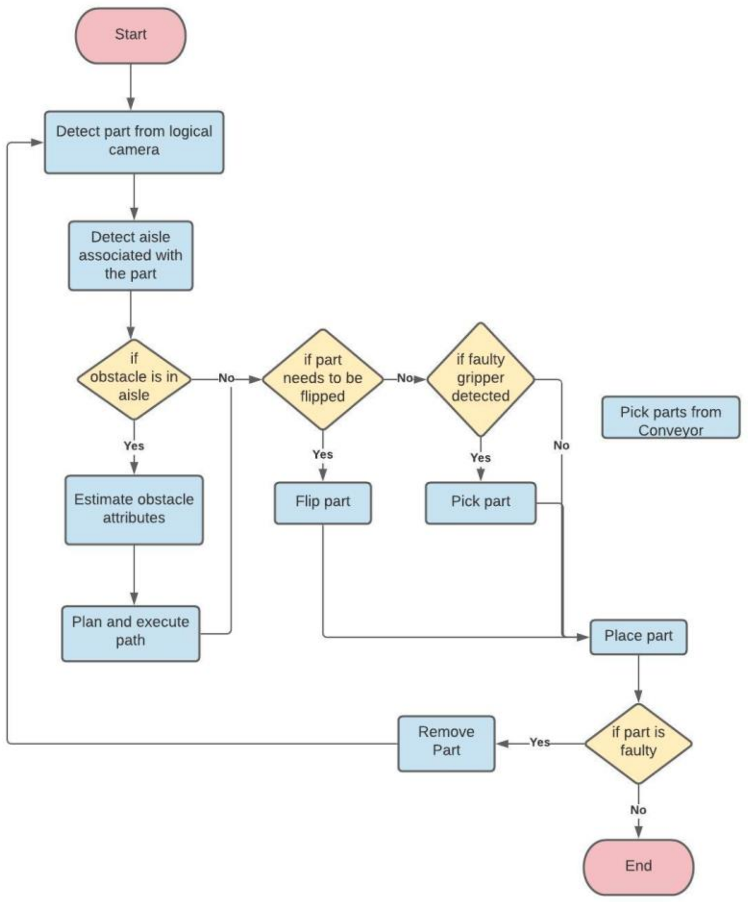


**Figure 4: Process Part Flowchart**

## Agility Challenge

### a. Faulty Part

#### Description:

In this challenge, some of the parts that need to be placed in the AGV are faulty parts and they need to be removed from the AGV and replaced by the same part that isn't faulty.

#### Approach:

In order to perform the faulty part challenge, we have accessed the quality control sensor on both AGVs that send information about whether the recently placed part is faulty or not. If it is found to be faulty, the part is discarded in the environment and replaced by the same part. This process continues until all the parts are not faulty.

## Implementation:

1. We keep track of the current part being placed in the AGV.
2. Once the part has been placed we query the Quality Control sensor to check if there is a faulty part in the AGV tray.
3. If it is found that the Quality Control sensor output changes to true after placing the previous part, we pick up the part from the position it is placed on the AGV tray.
4. We set up a safe location to drop the part on the floor of the environment, near the AGV and deactivate the gripper with the faulty part.
5. Once the faulty part has been discarded, we query the logical camera to find the location of the parts similar to the faulty part and complete placing a new replacement part in the AGV.
6. Again we check if the part is faulty and repeat the procedure until all the parts in the AGV tray are not faulty.

## Problem Faced:

1. We were not able to pick the faulty part, if the part was present at the corners of the agv tray. The Moveit could not find a path to pick the part.
2. We could not pick the flipped pulley faulty part from the agv tray as we needed to add a threshold to pick the flipped pulley faulty part.

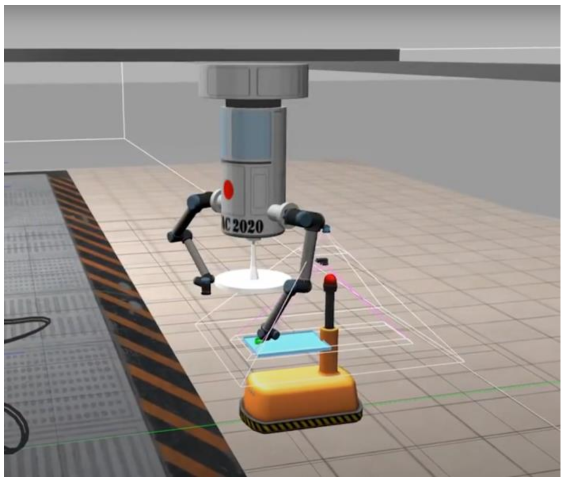


**Figure 5: Picking the faulty part**

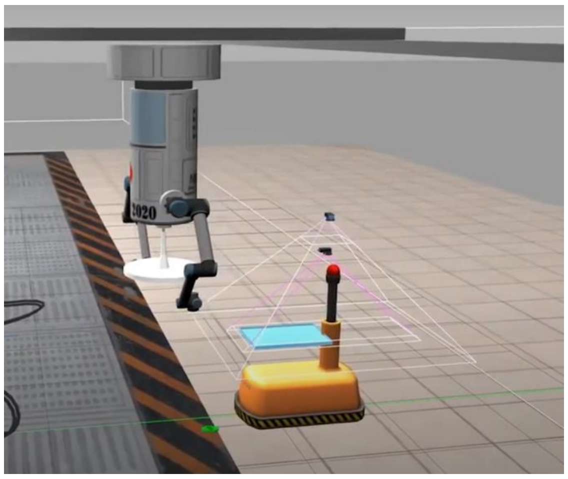


**Figure 6: Dropping the faulty part**

### Flowchart:

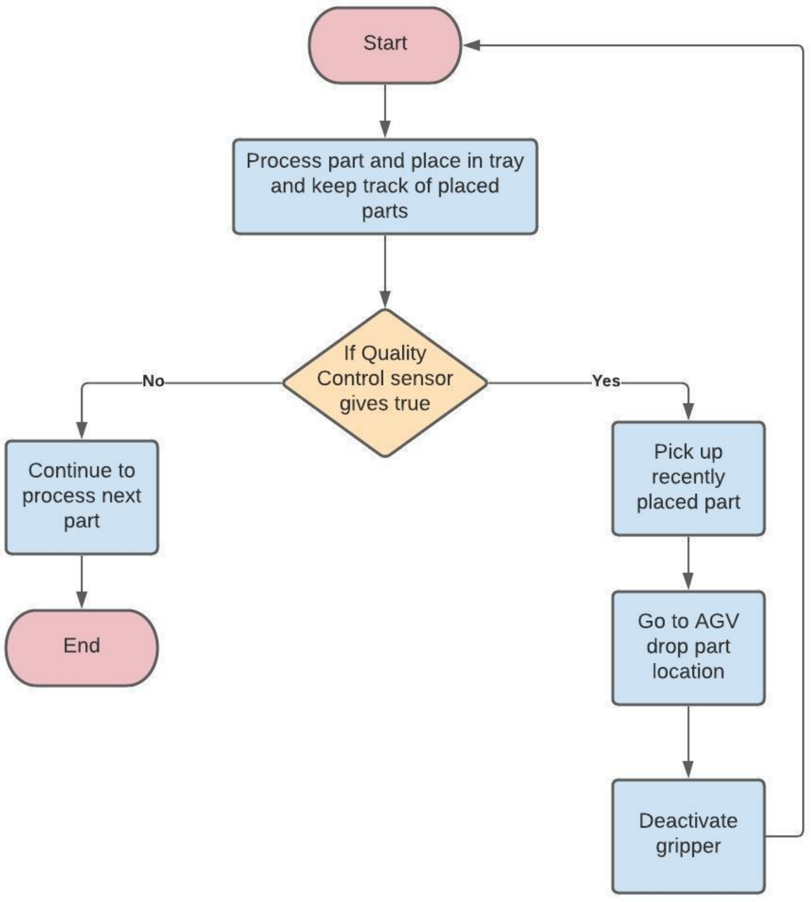

Start
Process part and place in tray and keep track of placed parts
If Quality Control sensor gives true
No
Yes
Continue to process next part
End
Pick up recently placed part
Go to AGV drop part location
Deactivate gripper

**Figure 7: Flowchart of the Faulty Part implementation**

## b. Faulty Gripper:

### Description:

In this challenge, the gripper gets deactivated before placing the part on the AGV when delivering the part. This causes the part to be placed in the wrong position and orientation on the AGV.

### Approach:

To correct this issue, the logical camera is used to check if the faulty gripper has occurred based on the location of the placed part. The part is then re-picked from the wrong position and placed in the appropriate position.

## Implementation:

1. We need to keep track of the placed part on the AGV.
2. Once the part is placed, the current position of the part is obtained from the AGV logical camera.
3. The Euclidean distance between the expected part position and the actual part position is calculated to check if the difference is greater than a set threshold.
4. If it is found that the Euclidean distance is greater than the threshold, the part is picked from the position and attempts to place it in the expected position obtained from the orders.

## Problem Faced:

1. Picking the dropped part in the agv tray was difficult when the part was dropped on the edge of the tray.

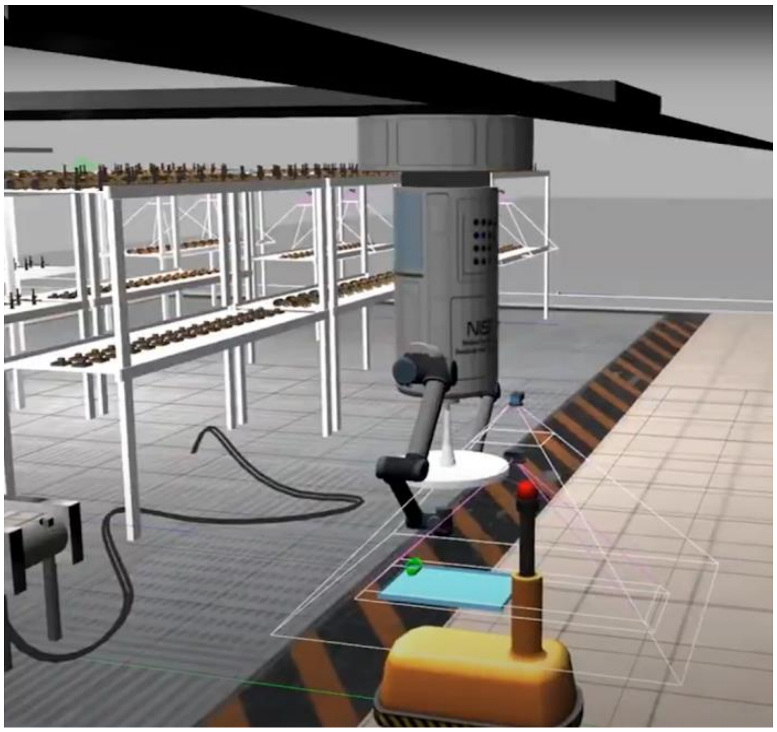


**Figure 8: Picking part from edge of the agv tray**

### Flowchart:

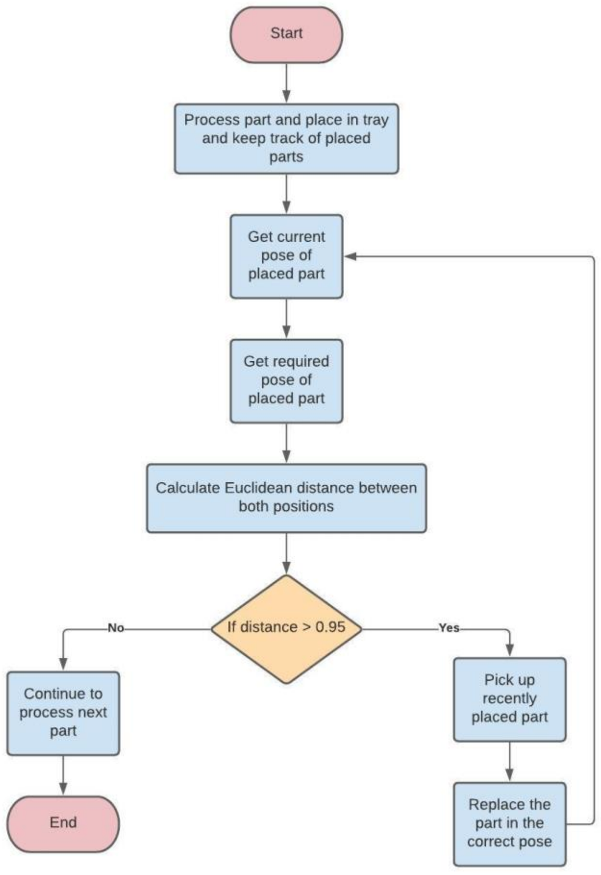

Start
Process part and place in tray and keep track of placed parts
Get current pose of placed part
Get required pose of placed part
Calculate Euclidean distance between both positions
If distance > 0.95
No
Yes
Continue to process next part
End
Pick up recently placed part
Replace the part in the correct pose

**Figure 9: Flowchart of the Faulty Gripper implementation**

## c. High Priority Order

### Description:

In this challenge, a new order will be published to ariac/orders topic while the current order is being executed. The gantry should stop executing the current order and deliver the high priority order as quickly as possible and then complete the previous order.

### Approach:

We have divided the high priority order challenge into two categories. First, when the high priority order needs to be delivered in the same agv as the previous order. Second, when the high priority order needs to be delivered in different agv. The first category is a bit complex when compared to the second.

## Implementation:

1. Determining whether high priority order has occurred by checking the size of orders.
2. If it occurred, we need to store the current state of the previous order before moving to the high priority order.
3. Then we need to check whether the high priority order needs to be delivered in the same or different agv as the previous order.
4. If the high priority order needs different agv, then we need to execute the order.
5. If the high priority order needs the same agv, then we need to remove the unwanted parts in the agv which are not needed for the high priority order.
6. Then, we need to reposition the common parts of both orders in the agv according to the position and orientation specified on the high priority order.
7. We then need to process the remaining parts of the high priority order.
8. After completing the high priority order, the gantry needs to move to the stored state of the previous order and complete it.

## Problem Faced:

1. If the high priority order needs to be delivered to the same agv as the previous order, it becomes very complex to deliver the agv with correct parts especially integrated with sensor blackout.

## Flowchart:

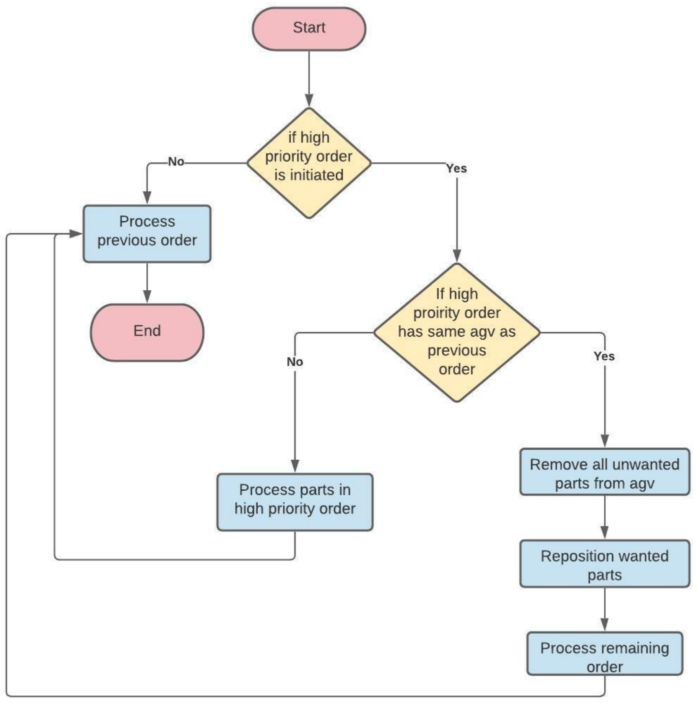

Start
if high priority order is initiated
No
Yes
Process previous order
End
If high proirity order has same agv as previous order
No
Yes
Process parts in high priority order
Remove all unwanted parts from agv
Reposition wanted parts
Process remaining order

**Figure 10: Flowchart of the high priority order implementation**

## d. Conveyor Belt

### Description:

Parts will start spawning in the conveyor belt some time after the competition begins. The gantry has to pick the part from the moving conveyor belt if it is needed for the completion of the order.

### Approach:

We first placed a logical camera over the conveyor belt to determine the type of part being spawned in the conveyor belt. Then, we placed a break beam sensor along the conveyor in such a way that the part will hit the beam everytime it crosses the break beam sensor. First, we have to wait for simulation seconds for the logical camera to detect a part. As soon as the logical camera and break beam detects the part, the gantry goes to a predefined preset location to pick the part.

### Implementation:

1. As soon as the competition began, we waited for 18 simulation seconds to check if any part is spawned on the conveyor belt.
2. If the logical camera detects a part, the gantry waits till that part hits the break beam.
3. As soon as the part hits the break beam, the gantry goes to predefined preset location, picks and places the part in bin13.
4. The gantry will try to pick up 3 parts from the conveyor belt.
5. We have allocated 90 simulation seconds for the gantry to pick 3 parts from the conveyor belt and place it in bin13.
6. After 90 simulation seconds is reached, the gantry deactivates its gripper and goes to the start configuration and starts executing the order.

### Problem Faced:

1. Picking the parts from the conveyor belt was difficult especially when the piston part was spawned. The width of the piston part made it difficult for the gantry to pick the part.

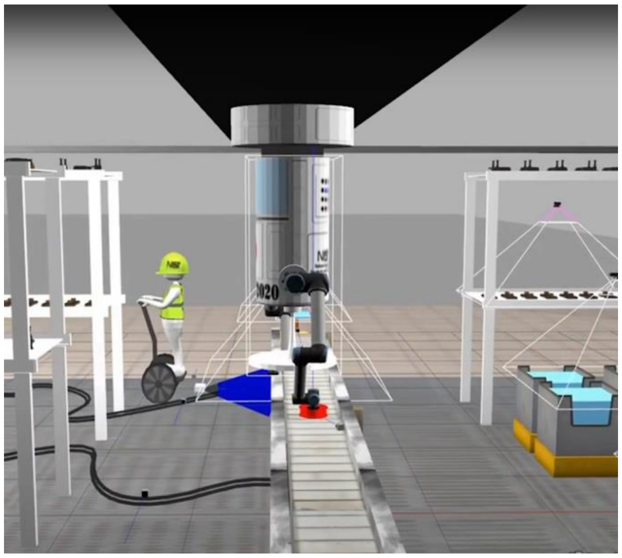


Figure 11: Picking part from conveyor belt

## Flowchart

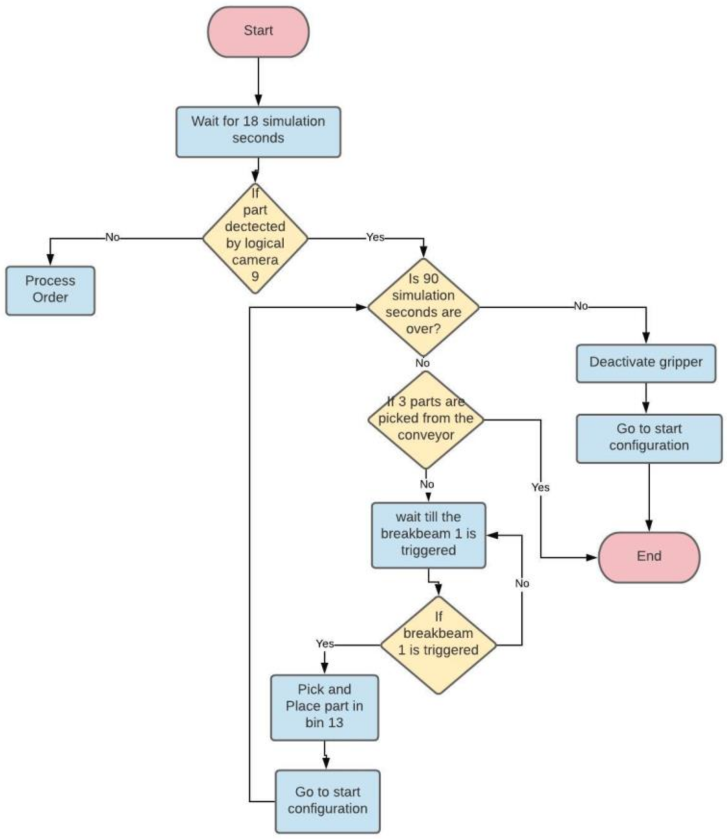

Start
Wait for 18 simulation seconds
If part dectected by logical camera 9
No
Yes
Process Order
Is 90 simulation seconds are over?
No
No
Deactivate gripper
Go to start configuration
If 3 parts are picked from the conveyor
No
Yes
wait till the breakbeam 1 is triggered
End
No
If breakbeam 1 is triggered
Yes
Pick and Place part in bin 13
Go to start configuration

Figure 12: Flowchart of the conveyor implementation

## e. Obstacle Avoidance

### Description:

In this challenge, the gantry maneuvers the human obstacles and operates safely.

### Implementation:

1. We made sure that the gantry can pick parts(both further and near) from the shelf with both orientations. For eg, The gantry can pick part on the left side of shelf A for both near and further parts and can also do the same on the right side of shelf A.
2. To pick parts the are two cases

**Part is located in an aisle with no obstacles**

-> No planning needed. Just go through aisle and pick the part

**Part is located in an aisle with obstacles**

-> Part can be picked from an aisle with no obstacle

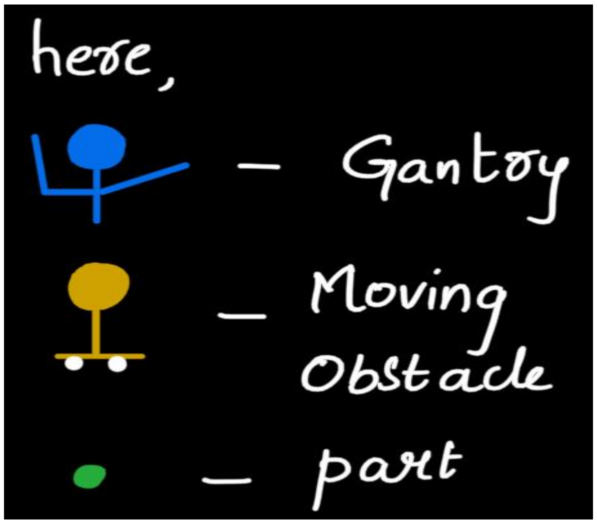


Figure 13: Labels

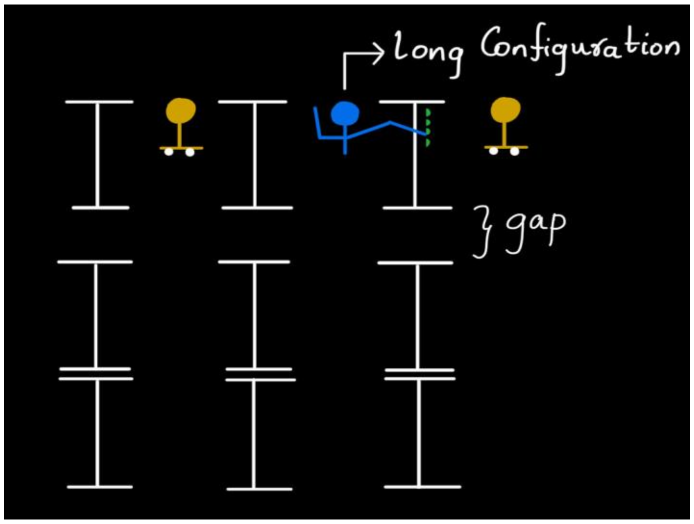
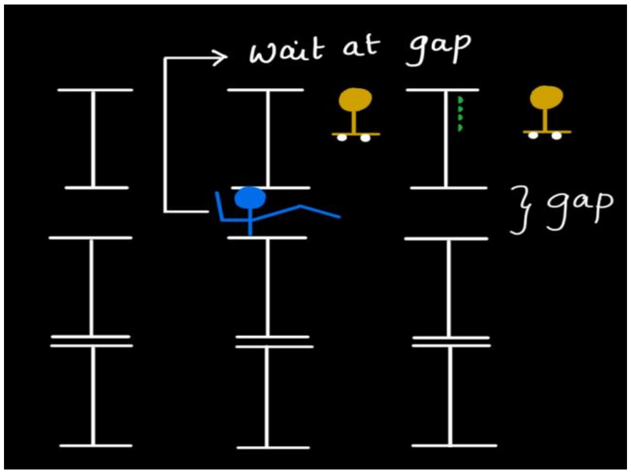


Figure 14: Picking part from aisle without obstacles

->Part cannot be picked from an aisle with no obstacles. Go to closest aisle gap with no obstacles, plan and pick part.



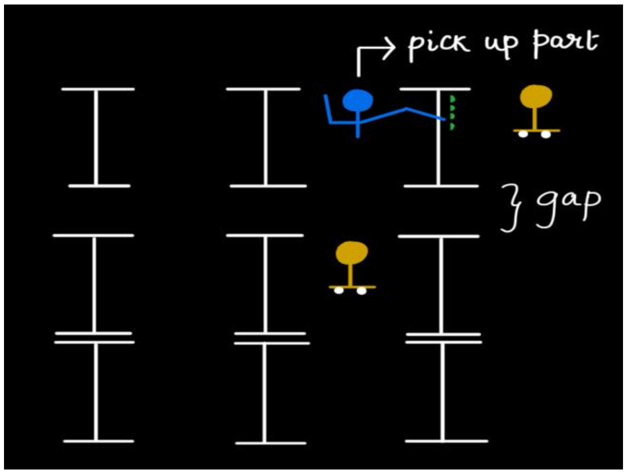


**Figure 15: Robot waiting at closes aisle gap**

**Figure 16:Robot picking up part**

In the planning stage, we estimate the obstacles attributes(ie move  time, wait time and also the direction of the obstacle). Robot should move and pick up the part when it is safe to do so.

## f. Sensor Blackout:

**Store detected parts** - Parts detected under each of the logical cameras are  stored in a  map data structure. Update data structure when new parts are detected and remove parts when parts are processed.

Precarious situations to be concerned about sensor blackout:

**Sensor blackout occurs after placing faulty part**- Continue to process part and before delivering agv, make sure to replace all faulty parts.

**Duplicates in published parts** - Published parts fluctuate a bit and as a result, it is possible to have duplicates of the same part stored in the  map. The solution used was to threshold the range for storing parts in the map and also restrict the size of the map.

**False parts detected** - As the robot moves, it is possible for the part to be detected under different logical cameras and this results in having duplicate parts in the map. Only add detected parts when the robot is in the start configuration. Do not add parts when the robot is moving. Detected parts have different configuration - It is possible that the robot may have difficulties when picking up a part. The robot may drag parts across and this has the effect of changing the poses of previous parts under the logical camera. As a result, the parts stored in the map may have incorrect(poses) and not the updated ones. The solution is to periodically update the poses when there is no sensor blackout.

## e. Flipped Part

### Description:

In this Challenge we flip the pulley part before it was placed on the AGV for delivery.

### Approach:

To successfully flip the pulley part we watched the Ariac 2020 video and tried to emulate the same.

### Implementation:

1. First we move to the pulley part location and pick the part using the left arm.
2. Then we move to the AGV location where the part needs to be placed for delivery.
3. After determining that the part needs to be flipped using RQT we determined the preset configurations for both the right and left arms.
4. Now move to the preset flipped configuration, activate the right gripper and deactivate the left gripper.
5. Finally place the flipped part on the AGV.

### Flowchart

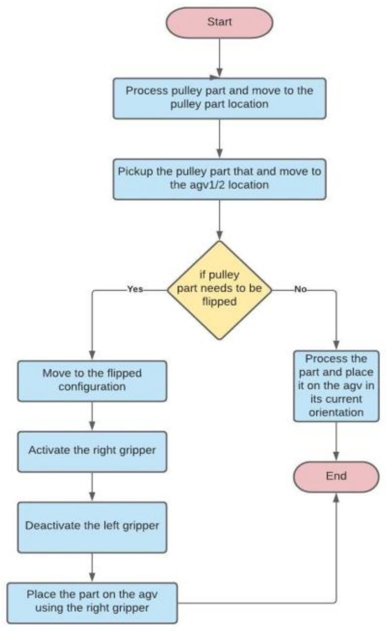

Start
Process pulley part and move to the pulley part location
Pickup the pulley part that and move to the agv1/2 location
if pulley part needs to be flipped
Yes
No
Move to the flipped configuration
Activate the right gripper
Deactivate the left gripper
Place the part on the agv using the right gripper
Process the part and place it on the agv in its current orientation
End

**Figure 17: Flowchart of the Flip part implementation**

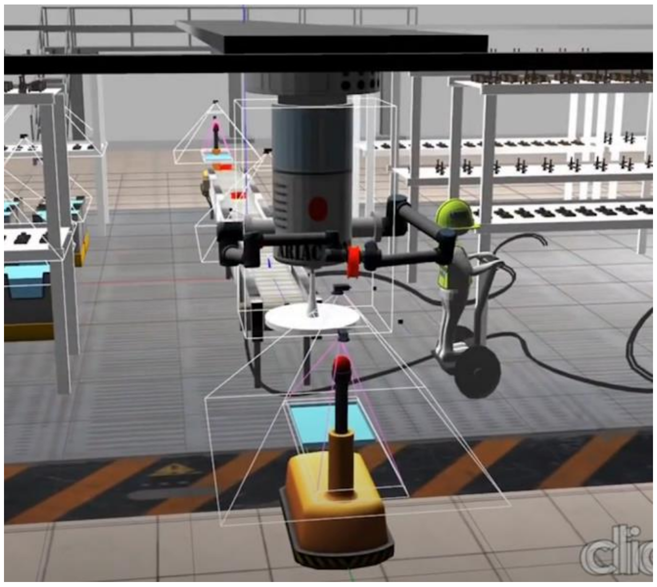


**Figure 18: Flip part configuration**

## Dynamic Preset Locations

Dynamic Preset locations is an important milestone we achieved in this project. The need for making the preset locations dynamic arises when the parts are placed far apart from one another and one preset configuration is not enough to make the gantry reach all the parts. Hence, in dynamic preset location, after reaching the preset configuration, we move towards the coordinates of the part detected by the logical camera. In this way, the gantry picks the part easily.

## Contribution

We have divided the work equally for every assignment consisting of tackling one agility challenge at a time. The following are the contributions made by the team members on agility challenges and other important milestones:

1. Faulty Part:
    a. Eashwar Sathyamurthy
    b. Akwasi A Obeng
    c. Sri Sai Kaushik Mushty
2. Conveyor:
    a. Eashwar Sathyamurthy
    b. Sri Sai Kaushik Mushty
    c. Shantam Bajpai
3. Faulty Gripper
    a. Eashwar Sathyamurthy
    b. Sri Sai Kaushik Mushty
    c. Shantam Bajpai

4. Flip Part
    a. Eashwar Sathyamurthy
    b. Sri Sai Kaushik Mushty
    c. Shantam Bajpai
5. High Priority Order
    a. Akwasi A Obeng
    b. Sri Sai Kaushik Mushty
6. Obstacle Avoidance
    a. Akwasi A Obeng
7. Sensor Blackout
    a. Akwasi A Obeng
8. Preset Locations:
    a. Eashwar Sathyamurthy
    b. Sri Sai Kaushik Mushty
9. Dynamic Preset locations
    a. Eashwar Sathyamurthy
    b. Sri Sai Kaushik Mushty
10. Structure of Program
    a. Akwasi A Obeng
11. Testing and Debugging
    a. Eashwar Sathyamurthy
    b. Akwasi A Obeng
    c. Sri Sai Kaushik Mushty
    d. Shantam Bajpai
12. Report and Presentation
    a. Eashwar Sathyamurthy
    b. Akwasi A Obeng
    c. Sri Sai Kaushik Mushty

## Future Works

The following are the milestones we plan to accomplish in the future:

1. We plan to make preset locations more robust for all the agility challenges.
2. We are planning to implement the conveyor functionality in a more robust manner by moving the gantry with the same speed as the conveyor to pick up the part.

## References

1. Usnistgov. “Usnistgov/ARIAC.” *GitHub*, github.com/usnistgov/ARIAC.
2. “Highlight Videos from the 2020 ARIAC Competition.” *NIST*, 1 June 2020, www.nist.gov/video/highlight-videos-2020-ariac-competition.