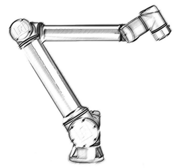


Spring 2023

Final Report (Group 6)

ENPM663: Building a Manufacturing Robot Software System

Rishikesh Jadhav - 119256534
Vishaal Sivakumar - 117764314
Abhinav Garg - 117553000
Saketh Banagiri - 118548814
Nishant Pandey - 119247556

Contents

1 Introduction

the ARIAC competition has been created to handle pick-and-place issues in the industrial setting, ARIAC is a simulation-based competition that aims to increase the agility of robotic systems. The National Institute of Standards and Technology (NIST) launched the Agile Robotics for Industrial Automation Competition (ARIAC) in 2017 and began paying cash prizes to the top 3 winners every year later in an effort to improve the agility of industrial robots. The tournament, which has been organized for six years running, focuses on evaluating the agility of industrial robots using pick-and-place tasks.

The Open Source Robotics Foundation (OSRF)'s Gazebo open-source 3D simulator is used in the simulation-based competition known as ARIAC. The OSRF-designed and NIST-maintained Gazebo Environment for Agile Robotics (GEAR) software serves as an interface between competitor packages and the competition environment. Participants in the competition must create a system in ROS to complete a set of shipments. Participants must choose which sensors to deploy in the environment for one or more given robots. These options include break beam, proximity sensor, laser scanner, cameras, and more.

1.1 Challenges

- **Faulty Parts**: Faulty parts are parts that are not in good condition. They are not suitable for use in the competition. If an order is submitted with faulty parts, these parts are not considered for scoring. Faulty parts can be identified using ARIAC services. The goal of this challenge is to test the ability of the CCS to correctly use the quality checker service to detect faulty parts and replace them with new parts.
- **High-priority Orders**: The high-priority orders challenge simulates an order that must be completed before a low-priority order. The high-priority order must be completed and submitted before the low-priority order. The goal of this challenge is to test the ability of the CCS to prioritize high-priority orders over low-priority orders. This requires the CCS to be able to detect when a high-priority order is announced and to switch tasks.
- **Insufficient Parts**: The insufficient parts challenge simulates a situation where the work cell does not contain enough parts to complete one or multiple orders. The goal of this challenge is to test whether or not the CCS is capable of identifying insufficient parts to complete one or multiple orders. When an insufficient parts challenge takes place, the CCS must use an alternate part or submit an incomplete order.

2 Architecture

The figures below show the flow of the architecture and execution of all the tasks:

- **Kitting**: In Figure 1, the robot detects the parts on a conveyor belt using the Advance Logocal Camera. These parts are picked up using the floor robot and placed in bins(empty slots). The gripper is changed to a tray gripper to pick up the empty tray. The trays are placed on the AGVs. The parts from the bins are then placed on the tray based on the kitting requirements and the AGV is sent to the destination for assembly. Throughout the process, the agility challenges like High priority orders, and faulty parts are taken into account. If there is a high-priority order, then, the ongoing kitting process is stopped and the execution of high priority order is started. If there are issues with the quality of the part then that part is dropped into one of the blue bins on the warehouse floor.

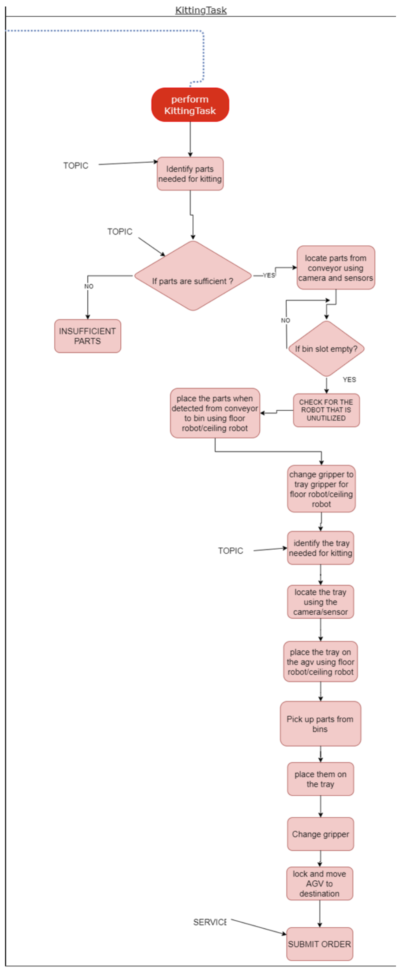

KittingTask
perform KittingTask
TOPIC
Identify parts needed for kitting
TOPIC
If parts are sufficient ?
NO
YES
INSUFFICIENT PARTS
locate parts from conveyor using camera and sensors
NO
If bin slot empty?
YES
place the parts when detected from conveyor to bin using floor robot/ceiling robot
CHECK FOR THE ROBOT THAT IS UNUTILIZED
change gripper to tray gripper for floor robot/ceiling robot
TOPIC
identify the tray needed for kitting
locate the tray using the camera/sensor
place the tray on the agv using floor robot/ceiling robot
Pick up parts from bins
place them on the tray
Change gripper
lock and move AGV to destination
SERVICE
SUBMIT ORDER

Figure 1: Architecture Diagram-Kitting

- **Assembly**: In Figure 2, the assembly process is done using the ceiling robot. The parts from the tray are picked up and inserted into their respective slots.

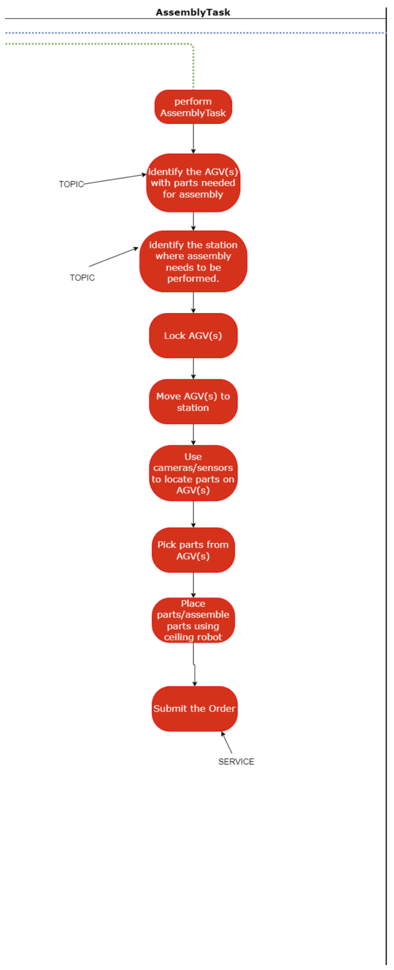

AssemblyTask
perform AssemblyTask
identify the AGV(s) with parts needed for assembly
TOPIC
identify the station where assembly needs to be performed.
TOPIC
Lock AGV(s)
Move AGV(s) to station
Use cameras/sensors to locate parts on AGV(s)
Pick parts from AGV(s)
Place parts/assemble parts using ceiling robot
Submit the Order
SERVICE

Figure 2: Architecture Diagram-Assembly

- **Combined**: Figure 3 represents a combined task, which is a combination of both kitting and assembly tasks. Kitting task is done using the floor robot and assembly task is executed using the ceiling robot.

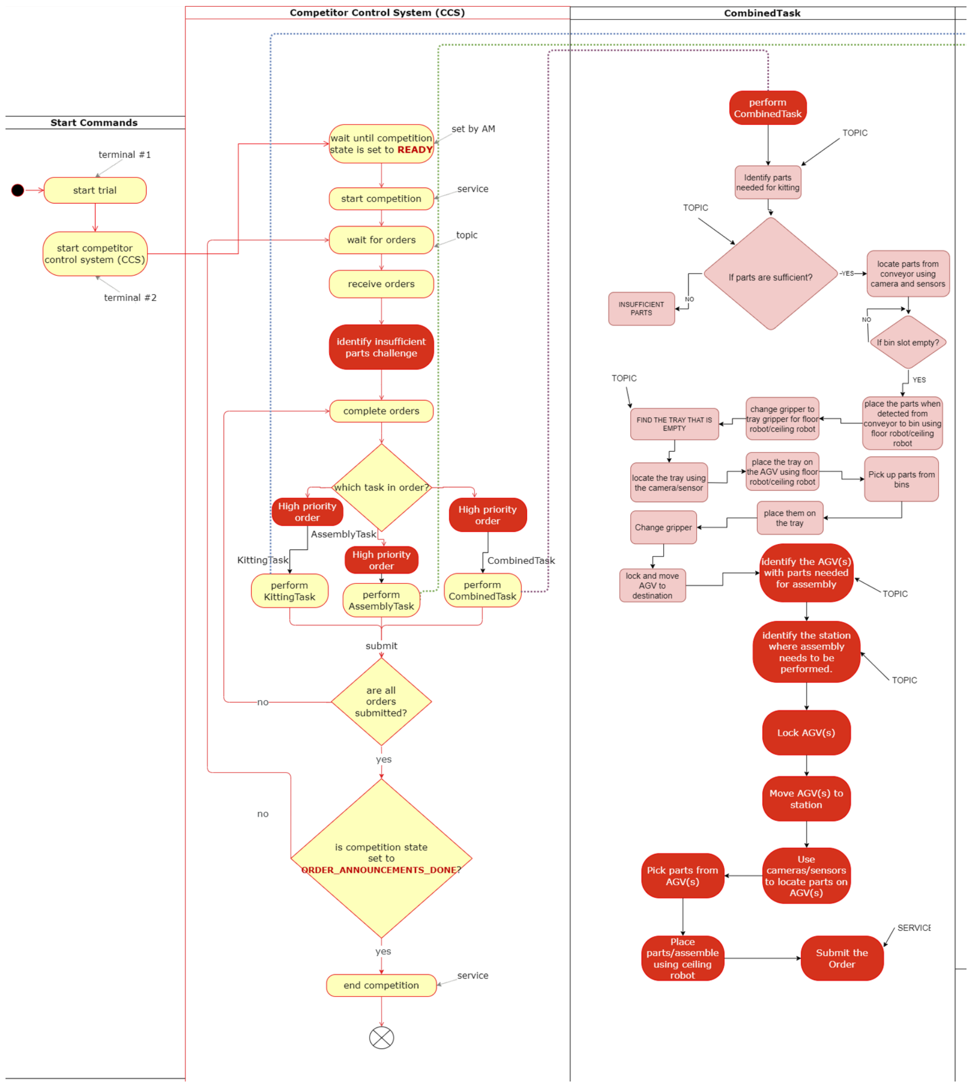

Start Commands
terminal #1
start trial
start competitor control system (CCS)
terminal #2
Competitor Control System (CCS)
wait until competition state is set to READY
set by AM
start competition
service
wait for orders
topic
receive orders
identify insufficient parts challenge
complete orders
which task in order?
High priority order
AssemblyTask
KittingTask
High priority order
High priority order
CombinedTask
perform KittingTask
perform AssemblyTask
perform CombinedTask
submit
are all orders submitted?
no
yes
no
is competition state set to ORDER_ANNOUNCEMENTS_DONE?
yes
end competition
service
CombinedTask
perform CombinedTask
TOPIC
Identify parts needed for kitting
TOPIC
If parts are sufficient?
YES
NO
INSUFFICIENT PARTS
locate parts from conveyor using camera and sensors
NO
If bin slot empty?
YES
place the parts when detected from conveyor to bin using floor robot/ceiling robot
TOPIC
FIND THE TRAY THAT IS EMPTY
change gripper to tray gripper for floor robot/ceiling robot
locate the tray using the camera/sensor
place the tray on the AGV using floor robot/ceiling robot
Pick up parts from bins
Change gripper
place them on the tray
lock and move AGV to destination
identify the AGV(s) with parts needed for assembly
TOPIC
identify the station where assembly needs to be performed.
TOPIC
Lock AGV(s)
Move AGV(s) to station
Use cameras/sensors to locate parts on AGV(s)
Pick parts from AGV(s)
Place parts/assemble using ceiling robot
Submit the Order
SERVICE

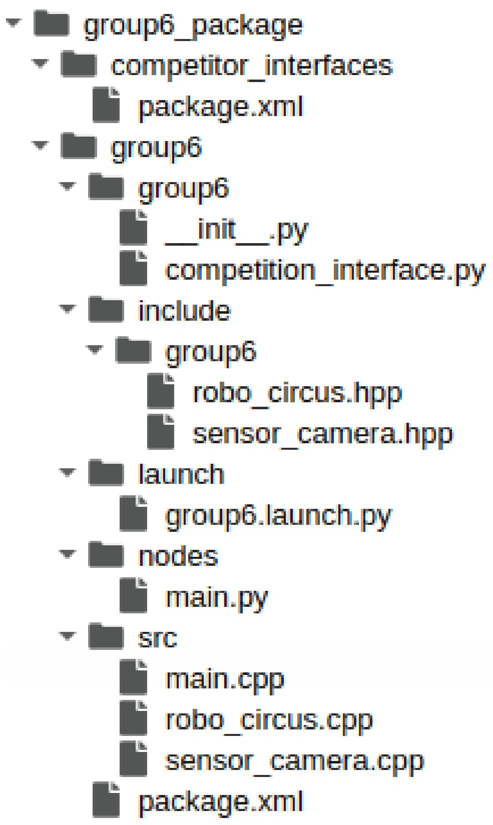
Figure 3: Architecture Diagram-Combined

3 Package Structure

The folder structure used by group 6 to perform all the challenges given as part of ARIAC is deptiicted in Figure 4. The folder contains two packages - the competitor_-interfaces and group6. The competitor_interfaces package defines the messages that would be used to communicate between group6 package nodes. The two main messages used for communication are,

1. RobotTask message, which contains information about different overall tasks (Kitting, Assembly and Combined) that the CSS would need to complete before ending the competition, and
2. CompletedOrder message, which has the order id so that the order could be sub-

mitted once it is completed.



```
group6_package
  competitor_interfaces
    package.xml
  group6
    group6
      __init__.py
      competition_interface.py
    include
      group6
        robo_circus.hpp
        sensor_camera.hpp
    launch
      group6.launch.py
    nodes
      main.py
    src
      main.cpp
      robo_circus.cpp
      sensor_camera.cpp
    package.xml
```

Figure 4: Package Structure

The group6 package has the files which cover the nodes that perform tasks during the ARIAC competition. There is a python node and a C++ node which communicate with each other and complete the required tasks.

4 Class Diagram

As discussed in the previous section, the group6 package has a python node - competitor_interfaces and a C++ node RoboCircus, shown in Figure 4. The competitor_interfaces node is responsible for starting the competition using a service and it subscribe to the orders topic and publishes the tasks in two topics - one for high priority task and the other for low priorty tasks. This node also submits the orders which it receives through the completed orders topic and finally ends the competition once the RoboCircus node has completed all the tasks.

The Robocircus node parses the tasks from the two orders topics, checks and processes the orders based on priority and type of task. It is responsible for calling services for

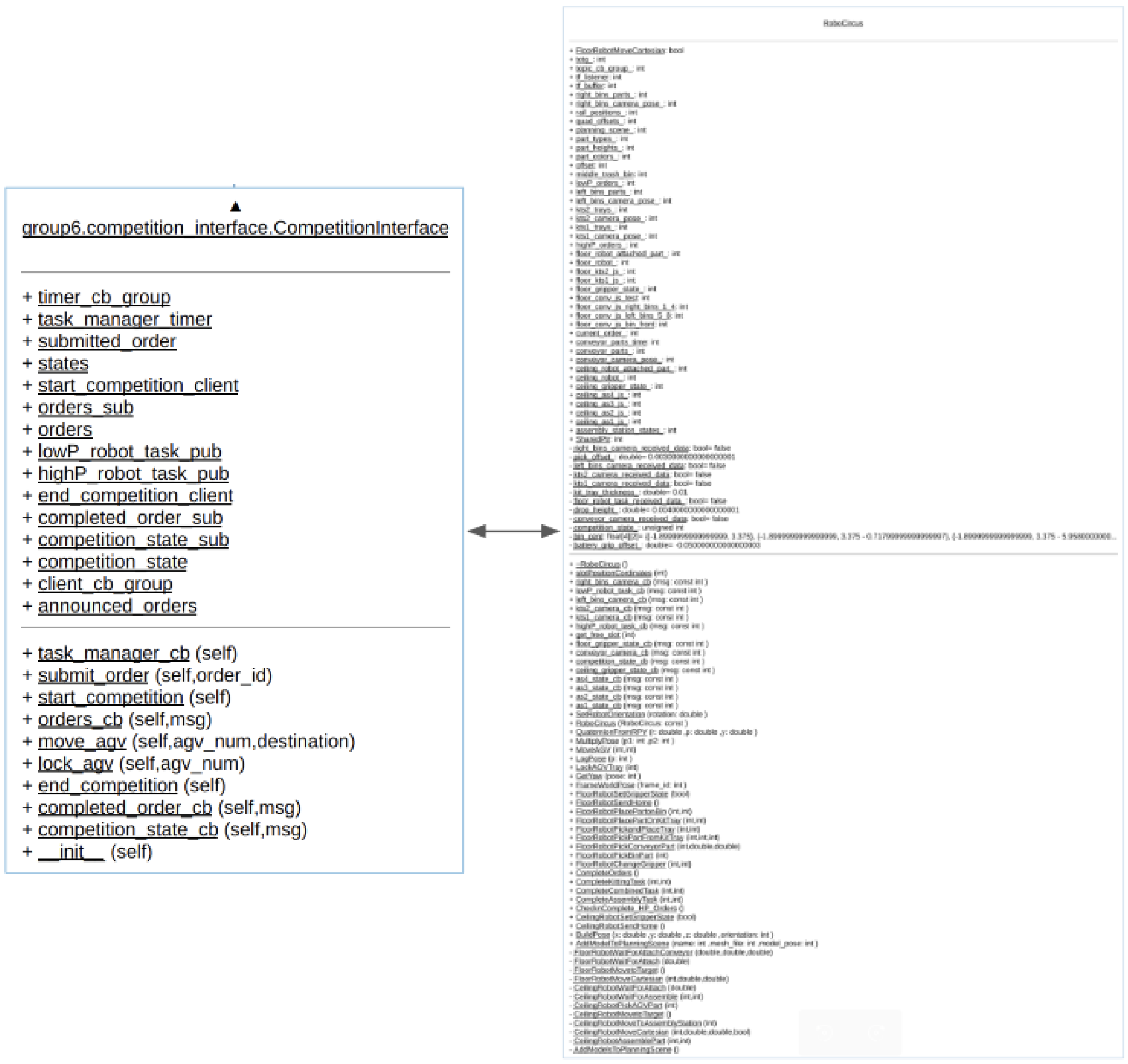


Figure 5: Class Diagram

moving the AGVs and sending joint level control commands to the floor robot and ceiling robot. There are preset joint positions and local planning involved for different tasks such as placing parts/trays, picking up parts from conveyors/bins, and placing the parts on trays over AGVs. The node checks for different conditions and plans the path for the manipulators based on the preset positions. Once the task is complete, the RoboCircus node publishes the order id so that it can be submitted by the competitor interfaces node.

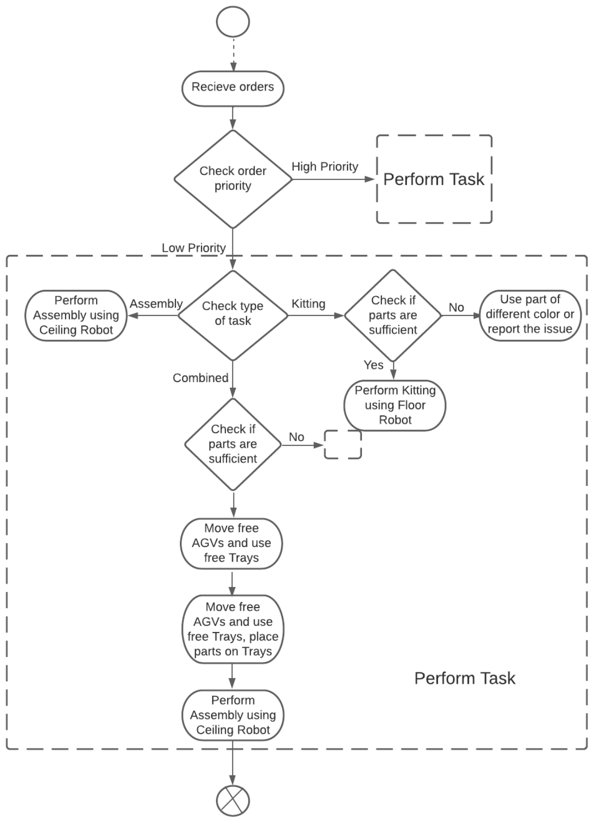

Recieve orders
Check order priority
High Priority
Perform Task
Low Priority
Perform Assembly using Ceiling Robot
Assembly
Check type of task
Kitting
Check if parts are sufficient
No
Use part of different color or report the issue
Yes
Perform Kitting using Floor Robot
Combined
Check if parts are sufficient
No
Move free AGVs and use free Trays
Move free AGVs and use free Trays, place parts on Trays
Perform Task
Perform Assembly using Ceiling Robot

Figure 6: Flow chart depicting how orders are processed

5 Order Parsing and Processing

Once orders are received in the competitor_interfaces node, they are published into the low and high priority orders topics as discussed before. The RoboCircus node subscribes to these topics and stores them in vectors of type RobotTask message, which is created as part of the competitor_interfaces package. The orders are again stored in two vectors based on the task priority and contain the order ids, the priority, the type of task and the task details. This is done to prioritize the high priority orders when they are received, the details are discussed in later section on how this is performed.

The orders get appended as they are received at different time conditions and processed as soon as the previous task is completed. If the current order is a Kitting or Combine task, the sufficiency of parts is checked first. The required parts for the order are compared with the available parts in bins and moved from the bin to the trays. If the parts are not sufficient to complete an order, a part of the same type and different color if found, is used to complete the order. Else the order is submitted with the incomplete number of parts.

In case of a combined order, the AGV and tray numbers are decided based on the availability. The tray is kept on the AGV using the floor robot and the parts are moved in the sequence given in the order. Once the parts are placed the AGVs are locked and moved to the given destination and submitted in the case of a Kitting task. For the Assembly and combined task, the AGVs are moved to the given assembly station and the pre-assembly poses of parts are obtained using a service. The ceiling robot is then moved to the assembly station. The RoboCircus node provides joint control commands to insert the parts based on the given poses and assembly direction, and the order is submitted.

6 Sensors and Cameras

In this environment, sensors and cameras are utilized to detect the part type, color, and location. At first, Advanced Logical Cameras were utilized successfully to perform the kitting and assembly tasks. However, they were later replaced with RGB cameras and Basic Logical Cameras to perform the same tasks.

The cameras are placed in pairs on the bins, kit trays, and conveyor belt. The RGB camera is used to determine the color and type of the part, while the Basic Logical Camera provides information about the part's position and orientation. Figure 7 depicts the competition arena with all the sensors.

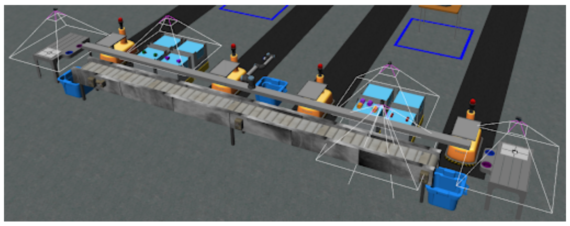


Figure 7: Camera positions in the system

To determine the color of the part, HSV masking is utilized, which allows the system to isolate the part's color and ignore any background or other colors in the image. The part's type is determined by a logic which is the combination of the area of the mask obtained and aspect ratio of the mask, which enables the system to differentiate between different types of parts based on their physical characteristics. Figure 8 consists of a flow chart interpreting the pipepline followed to distinguish the parts.

On the conveyor belt, only a specific part of the belt is considered to detect the type of the part, as the area of the part would vary depending on its orientation relative to the

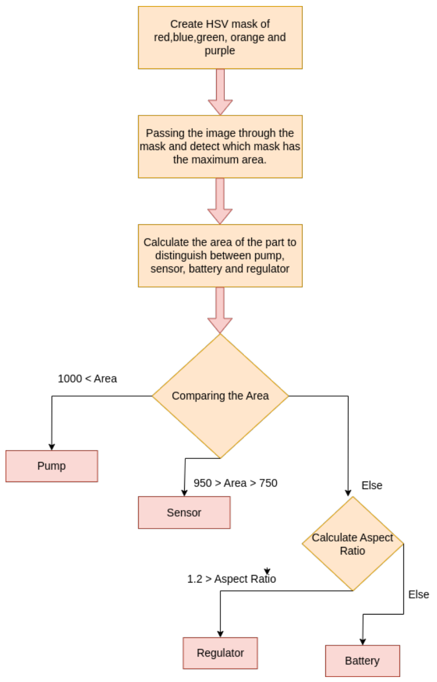

Create HSV mask of red,blue,green, orange and purple
Passing the image through the mask and detect which mask has the maximum area.
Calculate the area of the part to distinguish between pump, sensor, battery and regulator
Comparing the Area
1000 < Area
Pump
950 > Area > 750
Sensor
Else
Calculate Aspect Ratio
1.2 > Aspect Ratio
Regulator
Else
Battery

Figure 8: Perception Pipeline

camera. By focusing on a specific part of the belt, the system can accurately identify the type of the part, regardless of its orientation. Figure 9 and Figure 10 show the sample output from conveyor camera and sample masked image respectively. Overall, this sensor-based approach provides an efficient, reliable, and computationally cost-effective way to identify parts.

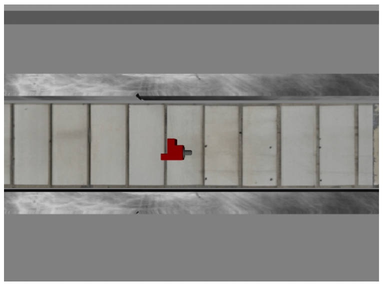


Figure 9: Sample conveyor camera output

7 Pick and Place from Conveyor Belt

In order to determine the pickup zone on a moving conveyor, an approach was employed that relied on the relative velocity between the floor robot and the conveyor part's velocity. By taking into account the speed of the conveyor (0.2 m/s) and the maximum

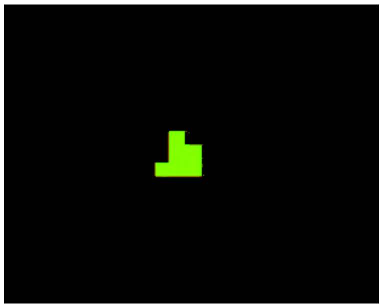


Figure 10: Sample masked image

speed at which the floor robot could travel on the conveyor (1 m/s), two distinct scenarios were considered. The first scenario involved the floor robot trailing behind the moving part, while the second scenario entailed the floor robot leading the part.

To ensure that the floor robot didn't solely track the object without picking it up, a stopping condition was incorporated into the algorithm. Additionally, an additional distance parameter was introduced. This parameter served the purpose of allowing the floor robot to reach the correct location and have enough time to plan its movement to the pickup configuration as the part approached.

By leveraging the relative velocities and considering the speed constraints, the objective was to accurately estimate the pickup zone for efficient retrieval of parts from the conveyor. This approach aimed to optimize coordination between the floor robot and the moving conveyor, facilitating smooth functioning of the system.

Also, the end effector is placed in a somewhat slanted position while picking up the part so that there is more contact area between the part and the end effector and the part doesn't get dropped in the middle of the pick and place operation.

Each bin is divided into a 3x3 grid and the part is then dropped inside the slot in the closest bin which is free.

8 Agility Challenges

8.1 Insufficient Parts Challenge

8.1.1 Approach

The team overcame the insufficient part challenge by creating an unordered map in C++ language. The key to this unordered map is a pair<int, int>, the first int corresponds to the respective color of the part and the second int corresponds to the part type. The value of the unordered map was again divided into two unordered maps. The map representation is shown in Figure 11.

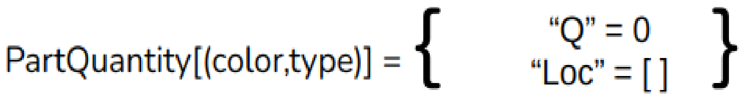


Figure 11: Key-Value Pair to track quantity

The 'Q' signifies the total quantity of that particular part type and color, and 'Loc' stores the location of where that part type can be found. In the case of an insufficient part, the quantity would be 0. Example PartQuantity[('Red', 'pump')]['q'] = 0. Hence if the quantity of the part mentioned in the order is equal to 0 then it needs to be replaced with a part type with different color, it then proceeds with for loop on the array storing different color values (Red, Purple, Orange, Blue, Green). If the quantity of the part type of any different color is greater than zero that particular part is used.

8.1.2 Pseudo Code

```
function partCheck( PartQuantity, partType, color) -> int:
    colorList = {Red:1, Purple:2, Orange:3, Blue:4, Green:5]
    if ParQuantity[(color,type)] = 0:
        print("Insufficient Part)
        for key in colorList:
            if ParQuantity[(key,type)] > 0
                return colorList[key] # returns the int value of color
    else:
        print("Part is present")
        return 0                      # returns 0 if the part is present
```

8.2 High Priority Challenge

8.2.1 Approach

As per the requirement of high priority challenge in ARIAC competition, the teams were required to immediately implement a high-priority order as soon as it is received, even if robots are in the middle of executing a low-priority order. Group 6 team tackled this challenge by creating two vectors "High-Priority-Orders" and "Low-Priority-Orders" and a boolean flag. The idea is that whenever an order is received it will be appended to the two vectors based on their priority, and if a high-priority task is currently being implemented then the flag would be changed to 1 but to 0 if it's a low-priority task.

Following this, all the kitting, assembly, and combined tasks were discretized, in such a way that whenever the robot is not moving (moveit not being used) then it continuously checks the size of the hight priority vector and the status of the flag, if the size of the vector is 0 the function returns from the function. If the size is greater than zero it then proceeds with the higher priority task and assigns the flag to 1. The flag prevents the situation where the robot is executing a high-priority task and another high-priority task is received. Once the high-priority task is completed and the vector size is equal to 0 the function comes out of the recursive call and starts implementing the low-priority task from where it left off.

8.2.2 Pseudo Code

```
function HighPriorityOrderCheck() -> bool :
    if HighPriorityOrder.length > 0 and self.flag == 0:
        print( Received High Priority)
        self.flag = 1
        while HighPriorityOrder != 0:
            CompleteOrder();
        self.flag = 0
        return true
    else:
        return false
```

8.3 Faulty Parts Challenge

8.3.1 Approach

The approach to tackle the Faulty parts challenge in the kitting process involved the following steps:

1. **Identification and Pickup**:
    - The first step was to identify the parts required for kitting and initiate the picking process.
    - The Floor robot carefully picked up the first part.
    - It then moved towards the designated quadrant on the tray, positioning itself in the drop orientation.

2. **Quality Checking**:
    - Prior to dropping the part on the desired quadrant on the desired AGV, the Floor robot initiated a quality check to ensure its integrity.
    - By leveraging the quality checker service using the camera on the AGVs, the Floor robot verified whether the current part was faulty or not.
    - The quality checker service provided a reliable assessment of the part's condition without the need for visual inspection.
3. **Faulty Part Handling**:
    - In the event that the quality check determined the part to be faulty, a specific course of action was taken.
    - The Floor robot promptly dropped the faulty part into the designated trash bin for disposal.
    - Following the disposal, the Floor robot began the search for a suitable replacement to ensure a seamless kitting process.
4. **Replacement Strategy**:
    - When an exact replacement part, identical in color and type, was available in the bins, it was promptly selected.
    - The Floor robot carefully retrieved the replacement part and replaced the faulty part on the tray.
    - This strategy maintained the original specifications of the kit, ensuring a high level of quality control.
5. **Alternative Replacement**:
    - In cases where an exact replacement part was not available, an alternative approach was implemented.
    - The Floor robot opted to replace the faulty part with a part of the same type but in a different color.
    - This alternative ensured that the kit remained complete, even if there was a slight aesthetic deviation.
6. **Part Unavailability Handling**:
    - In the rare scenario where neither an exact replacement nor an alternative part was available, a contingency plan was in place.
    - The Floor robot proceeded with the kitting process for the remaining parts, excluding the unavailable part.

- This ensured that the rest of the kit assembly was not delayed, prioritizing the completion of the overall task.

7. **Comprehensive Quality Control**:
    - The entire kitting process was executed for all parts, following the aforementioned steps.
    - Emphasis was placed on ensuring that no faulty part was submitted in the final kit.
    - This comprehensive quality control approach aimed to deliver kits with high standards of accuracy and reliability.

8.3.2 Problems Encountered

The implementation of the parts kitting process encountered several challenges, which were addressed as follows:

1. **Lack of Camera for Exact Part Location on AGV**:
    - One major challenge was the absence of a camera on the AGV to precisely determine the part's location (pose and orientation).
    - To overcome this, we used the saved drop poses with a downward offset (in the z-direction) to place the part from the tray for replacement.
2. **Collisions Detected by Moveit**:
    - However, this approach resulted in collisions being detected by Moveit since the part was still present in the planning scene.
    - To resolve this issue, we had to remove the part from the planning scene, although this caused slight inconsistencies when picking the pump from the bins.
    - Through fine-tuning of the values and adjustments, we were able to make the process more robust and got rid of any disruptions caused by these inconsistencies.
3. **Quality Checker Service Limitations**:
    - Another challenge arose when the quality checker service failed to function properly for detecting faulty parts in quadrants 2, 3, and 4.
    - The solution to this was to call the service twice, with a 3-second delay, to ensure accurate and reliable detection of faulty parts, thus mitigating any potential flaws.

4. **Insufficient and Faulty Parts Scenario**:
    - In certain cases, kitting tasks involved identical parts in two or more quadrants, and there were insufficient quantities of those parts.
    - To handle this challenge, a replacement strategy has been introduced as mentioned earlier, ensuring that the kit remained complete.
    - However, a special case occurred when both insufficient and faulty parts presented a challenge simultaneously.
    - In this scenario, we tackled the situation by replacing the faulty part with a part of the same type and color already present on the tray and replaced this part using the replacement strategy mentioned above.
    - The kit was then submitted without the faulty part, guaranteeing that no faulty part would pass the quality control process.

9 Difficulties Encountered

Throughout the course, various challenges were encountered that posed difficulties in carrying out the required tasks effectively. The main difficulties that were faced are: Before starting the assignments, getting to know each other and setting up the goals and expectations was crucial. This step was undertaken to avoid any problems in submissions and ensure a smooth collaboration among team members.

RWA 1

- Selecting a data structure and correctly storing orders by creating classes and using Object oriented programming to do so was a new challenge for us especially coming from different backgrounds

RWA 2

- Orders were submitted with insufficient parts due to CCS not reading conveyor parts through the topic.

RWA 3

- While picking up the pump from the front, it often resulted in toppling while placing it.
- The dynamic pick and place task presented multiple issues, which was tackled as follows:
    - The floor robot was following the part, but we struggled to find an optimal stop condition (offset in the y-axis) that needed to be finely tuned.

  - To address the offset problem, we devised a solution by extracting the x-coordinate and adding it to the end effector position, along with a z-offset (specific to each part), obtained from the conveyor pickup configuration.
  - Implementing dynamic pick and place involved tracking parts based on their velocity and distance traveled in the y-direction. This approach allowed us to avoid fixed pickup zones and minimized the risk of missing parts, making the process more time-efficient.
  - To ensure consistent tracking, we relied on the first instance of the part encountered on the conveyor and tracked it throughout the conveyor belt.

RWA 4

- Integrating the ceiling robot functions with the floor robot functions proved challenging. We had to create a parent class and two subclasses, initially having only the FloorRobot class. The addition of the CeilingRobot class required us to modify the existing structure, with the parent class named Robocircus. This naming choice was inspired by the acrobatics-like behavior of the robots in Gazebo, reminiscent of an actual circus.
- Another issue arose during the combined task after completing the high priority kitting task. The ceiling robot failed to submit the order after completing the assembly part of the order due to an order ID mismatch. We resolved this by ensuring that the correct order ID was submitted, while maintaining the task tracking from the previous steps, thereby successfully completing high priority orders.

Final Demo

- In the final demo, we encountered several challenges and scenarios that were assigned to us by the professor. Our team worked diligently in the final week to tackle these obstacles and ensure a successful demonstration.
- One of the challenges we faced was dealing with faulty parts. In the environment, there were multiple faulty parts that needed to be identified and handled appropriately. Our system was designed to detect these faulty parts and take necessary actions, such as rejecting or replacing them, to ensure the quality of the final product.
- Another challenge was the availability of insufficient parts. We were faced with situations where there were not enough parts to complete the required tasks. Our system incorporated intelligent resource management techniques to optimize the utilization of available parts and prioritize high-priority orders, ensuring efficient production despite limited resources.

- Additionally, we had to address the issue of high-priority orders. These orders demanded immediate attention and had to be given precedence over other tasks. Our system was equipped to identify and prioritize high-priority orders, ensuring their timely completion and delivery.
- To enhance the robustness of our code for the final demo, we considered various scenarios.
  - Multiple low-priority kitting tasks were announced at the beginning of the simulation, and we successfully managed their execution.
  - We also handled situations where kitting or combined tasks were announced after another order was submitted, and tasks were announced when a part was already placed on the AGV of another task. Our system efficiently managed these scenarios, ensuring smooth task execution without conflicts.
  - Furthermore, we successfully dealt with the announcement of multiple combined tasks at a specific time. Our system promptly processed these tasks, maintaining order and efficiency in the assembly process.
  - One specific scenario involved a low-priority kitting task announced at the beginning, followed by a high-priority kitting task when specific conditions were met. We ensured that there were enough parts to complete both kits before submitting them to the warehouse, avoiding any potential conflicts. We also incorporated checks to ensure that no high-priority orders were received before locking and sending the AGV to the warehouse.
- During the final demo, each team was given one try, with a second try provided only in cases where the CCS (Central Control System) or simulation encountered issues.
- Scoring for the live demo was based on the points obtained when submitting orders during the ARIAC trial. The maximum points achieved in the trial translated to a full score for the live demo. In the demo our package was agile enough to tackle all the challenges and complete the orders given by the task manager seamlessly and we got a full score of 30/30 at the end of the demo.
- Throughout the final demo, our team demonstrated strong problem-solving skills, effective coordination, and efficient utilization of available resources to tackle the challenges and scenarios presented. We were confident in the reliability and performance of our system and it paid dividends when in mattered.

10 Contributions

We mostly worked together on all the assignments and followed the concept of 'pair programming'.

Table 1: Contributions in Each RWA Assignment

RWA 1	
Start Competition	Rishikesh, Saketh, Nishant
Storing the Orders	Vishaal, Saketh, Abhinav
Submitting the Orders	Rishikesh, Vishaal, Saketh, Nishant
End Competition	Rishikesh, Vishaal, Abhinav
RWA 2	
Locating parts on the bins and conveyor	Rishikesh, Vishaal, Saketh
Challenges - Insufficient and High Priority	Vishaal, Saketh, Abhinav
Kitting and Assembly Functions	Abhinav, Saketh, Nishant
RWA 3	
Locating parts and trays using cameras	Vishaal, Rishikesh, Abhinav, Nishant
Picking parts from Conveyor and Placing them on the bins	Vishaal, Rishikesh, Saketh
Integrating parts from Bins to Tray	Vishaal, Abhinav, Saketh
RWA 4	
Ceiling Robot Integration	Abhinav, Vishaal, Rishikesh, Nishant, Saketh
Faulty Part	Vishaal, Rishikesh
High Priority	Abhinav, Saketh, Nishant

11 Conclusion and Acknowledgments

In conclusion, the project encountered numerous challenges, which served as valuable learning experiences for the team. Although it was not possible to mention all the difficulties faced due to the extensive nature of the project, we have highlighted the most significant ones. It is important to note that these challenges did not deter our progress, as we were able to overcome them and successfully complete the project.

The project taught us valuable lessons and provided us with practical insights that we can apply to future projects. Throughout the process, we demonstrated our ability to collaborate effectively as a team, filling in each other's gaps and leveraging our collective strengths. This collaborative effort played a crucial role in the development of our package, shaping it into what it is today.

We extend our heartfelt gratitude to Prof.Zeid Kootbally, Prof.Craig Schlenoff, and TA Anirudh Krishnan Komaralingam for their unwavering guidance and support throughout the project. Their expertise and encouragement were instrumental in our success. Additionally, we would like to express our appreciation to each member of our group for their hard work, dedication, and commitment to the project's objectives.

Lastly, we hope that this final report serves as a valuable resource for other students and by sharing our experiences, we aim to provide insights into the possibilities and opportunities that this course offers. May our journey inspire and guide future students as they embark on their own educational endeavors.

References

[1] https://ariac.readthedocs.io/en/latest/getting_started/installation.html

[2] https://moveit.picknik.ai/humble/index.html