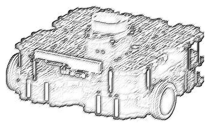


Kiran Ajith
Spring 2023
UMD
College Park, MD

Final Project - Group 14

ENPM809E: Python Applications for Robotics

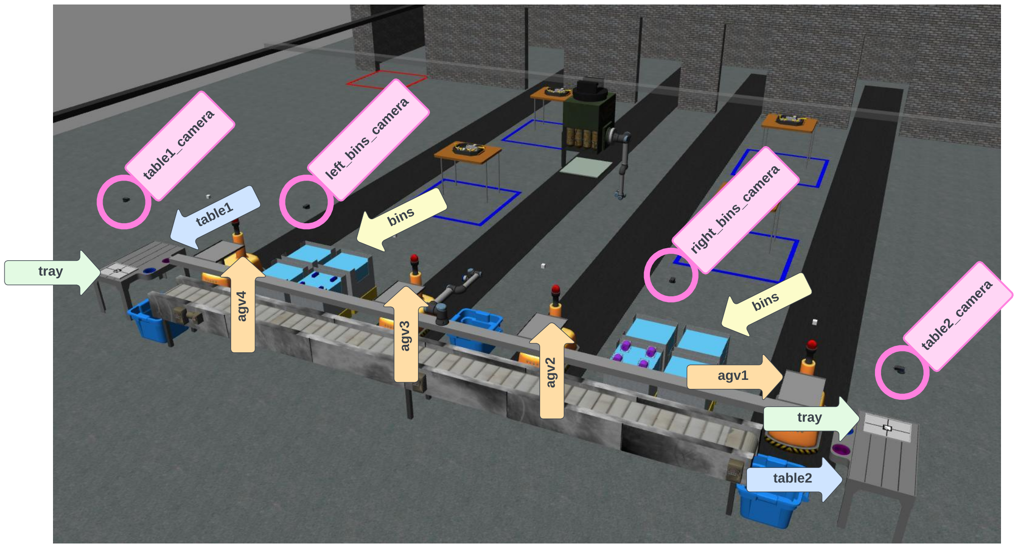
Contents

1 Introduction

The Agile Robotics for Industrial Automation Competition (ARIAC) is a robotics competition designed to showcase and evaluate the capabilities of robotic systems in the context of industrial automation. It serves as a platform for researchers, developers, and robotics enthusiasts to test and demonstrate their robotic systems in a simulated manufacturing environment.ARIAC provides a realistic simulation environment that emulates a complex manufacturing setup with various production tasks and challenges. The environment includes robotic arms, conveyor belts, sensors, cameras, and workstations, simulating a real-world factory floor. figure 1.1 is the environment used in the ARIAC 2023 competition.

1.1 Overview of the competition

Figure 1.1: The Competition Environment



The Environment consist of two robots, a floor robot and a Ceiling robot, which manipulate the action of pick-and-place of parts and trays. The trays are placed on tables on either side of the conveyor belt, while the parts are placed in different bins. Above the bins are two cameras to detect the presence and the location of the parts. There are two tool-changing stations by the tables, where the robots can pick from a choice of two gripper types. There is a specific gripper to pick up the trays and the parts.

Three tasks constitute the competition: Kitting, Assembly and combined tasks. The floor robot is mounted on the linear rail, and the robot can perform only kitting. The ceiling robot is a gantry robot mounted on rails attached to the ceiling. The ceiling robot can perform both kitting and assembly. Only the floor robot is used in this project.

Following are the processes and terms involved in the ARIAC competition:

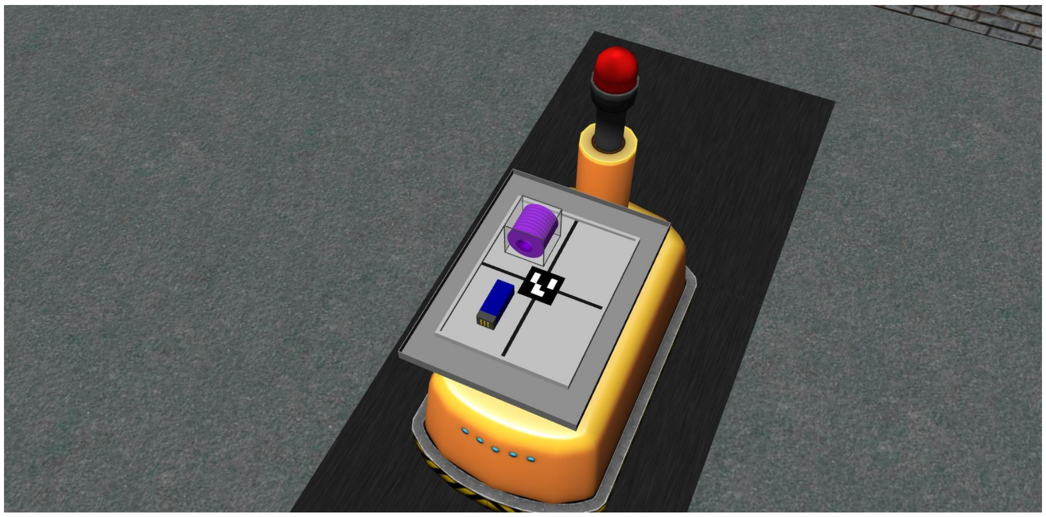
1. Kitting Task
   A kit consists of parts placed on top of a tray which is placed on the AGV. figure 1.2 shows a kit .The AGV transfers the kit to the warehouse for inspection in this task. The steps involved in the process are

   - Place a kit tray onto the AGV
   - Place parts from the bin on the tray located on the AGV
   - Send the AGV to the warehouse

2. Assembly Task
   In this task, participants must use the parts found on the (AGV) and assemble these components at any of the four available assembly stations.

3. Combined Task
   Combined task consist of both the kitting task and the assembly task.

Figure 1.2: A Kit



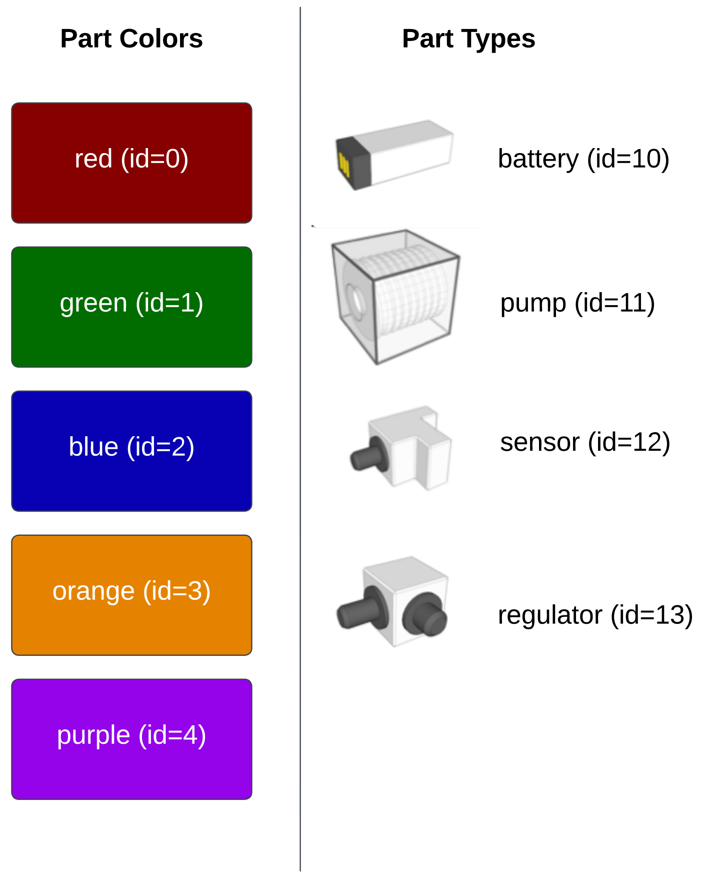
1.2 Terminology

- Order: An order contains information such as the Order id, what task is to be done at that point and how to make the kit. An order also contains the types of parts and their colour to be acquired.

- Part: Four types of parts are available for use (battery, pump, regulator, and sensor), and they can be of five colours (red, green, blue, orange, and purple). The specification of the part to be used will be mentioned in each order.figure 1.3 depicts the types of parts and the colors available. The part types and colors are referenced by id's and are used in the messages to identify it.

- Automated Guided Vehicle (AGV): An AGV is a mobile robot used to transfer parts from one location to another. For the kitting application, the AGV is used to transfer the parts to the warehouse. figure 1.2 shows an AGV.

Figure 1.3: Part colors and types.



1.3 Task

The aim of the project is to perform the kitting process of the ARIAC competition using the floor robot. When the competition starts, four orders are announced, and among them, only one order has to be processed. The floor robot should then be able to make the kit, and the kit has to be send to the warehouse.

1.4 Requirement

The system requirements to run the ARIAC 2023 Competition are as follows:

- Operating System : Ubuntu 20.04 Focal Fossa
- ROS Distribution : ROS2 Galactic
- Gazbeo Version : 11.11.0

2 Structure

2.1 Package Structure

Figure 2.1: Workspace structure

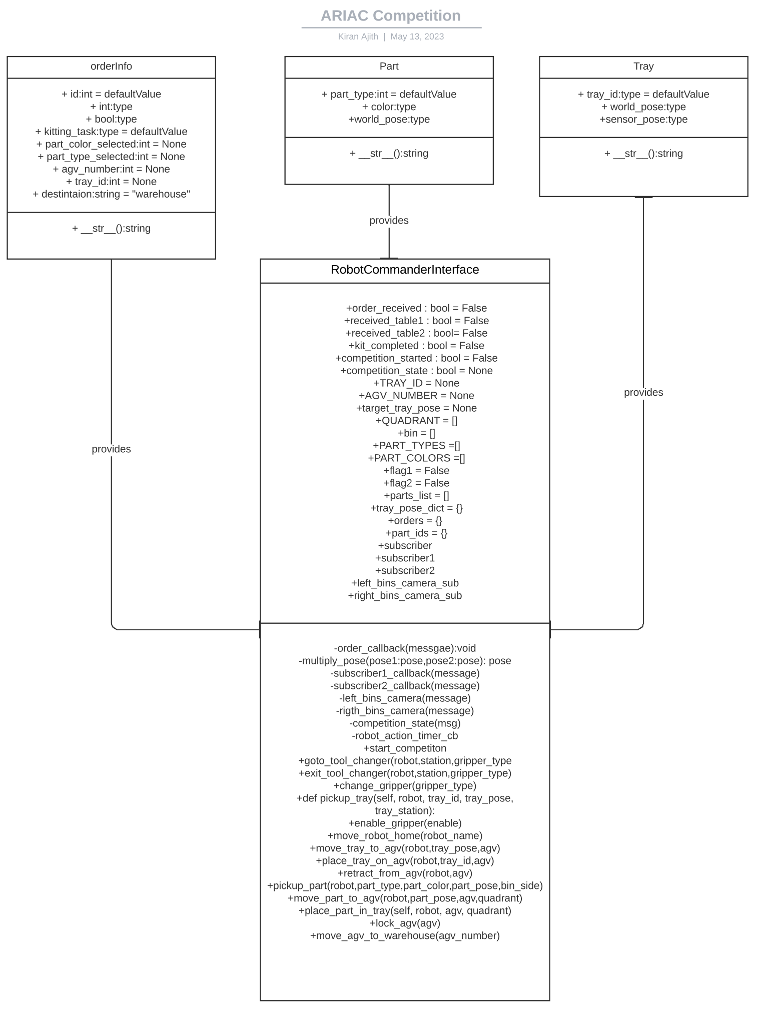
```
final_809e_ws
└─src
   └─enpm809e_final_spring2023
      ├─ARIAC
      └─final_group14
```

2.2 Conventions

Following are the conventions followed in the report.
n represents a Node
t represents a Topic
f represents a Frame

2.3 Class Diagram

Figure 2.2: UML of the classes


ARIAC Competition
Kiran Ajith | May 13, 2023
orderInfo
+ id:int = defaultValue
+ int:type
+ bool:type
+ kitting_task:type = defaultValue
+ part_color_selected:int = None
+ part_type_selected:int = None
+ agv_number:int = None
+ tray_id:int = None
+ destintaion:string = "warehouse"
+ __str__():string
Part
+ part_type:int = defaultValue
+ color:type
+world_pose:type
+ __str__():string
Tray
+ tray_id:type = defaultValue
+ world_pose:type
+sensor_pose:type
+ __str__():string
provides
provides
provides
RobotCommanderInterface
+order_received : bool = False
+received_table1 : bool = False
+received_table2 : bool= False
+kit_completed : bool = False
+competition_started : bool = False
+competition_state : bool = None
+TRAY_ID = None
+AGV_NUMBER = None
+target_tray_pose = None
+QUADRANT = []
+bin = []
+PART_TYPES =[]
+PART_COLORS =[]
+flag1 = False
+flag2 = False
+parts_list = []
+tray_pose_dict = {}
+orders = {}
+part_ids = {}
+subscriber
+subscriber1
+subscriber2
+left_bins_camera_sub
+right_bins_camera_sub
-order_callback(messgae):void
-multiply_pose(pose1:pose,pose2:pose): pose
-subscriber1_callback(message)
-subscriber2_callback(message)
-left_bins_camera(message)
-rigth_bins_camera(message)
-competition_state(msg)
-robot_action_timer_cb
+start_competiton
+goto_tool_changer(robot,station,gripper_type
+exit_tool_changer(robot,station,gripper_type)
+change_gripper(gripper_type)
+def pickup_tray(self, robot, tray_id, tray_pose, tray_station):
+enable_gripper(enable)
+move_robot_home(robot_name)
+move_tray_to_agv(robot,tray_pose,agv)
+place_tray_on_agv(robot,tray_id,agv)
+retract_from_agv(robot,agv)
+pickup_part(robot,part_type,part_color,part_pose,bin_side)
+move_part_to_agv(robot,part_pose,agv,quadrant)
+place_part_in_tray(self, robot, agv, quadrant)
+lock_agv(agv)
+move_agv_to_warehouse(agv_number)

The n commander contains four classes: RobotCommanderInterface, orderInfo, Tray, and Part. The functionality and relationship between the classes are discussed in this section.

1. orderInfo

   - It represents an order which is announced after the competition starts.
   - The attributes of the class store information about the order, such as order id, AGV num-

ber, type of tasks to be done, and the parts needed and which quadrant to place the parts.
- It overrides the __str__ method to provide a string representation of the order's details.

2. Tray
   - It represents a tray detected by the table cameras
   - The attributes of this class store information about the tray's identification and poses in different coordinate frames
   - It overrides the __str__ method to provide a string representation of the tray's details.

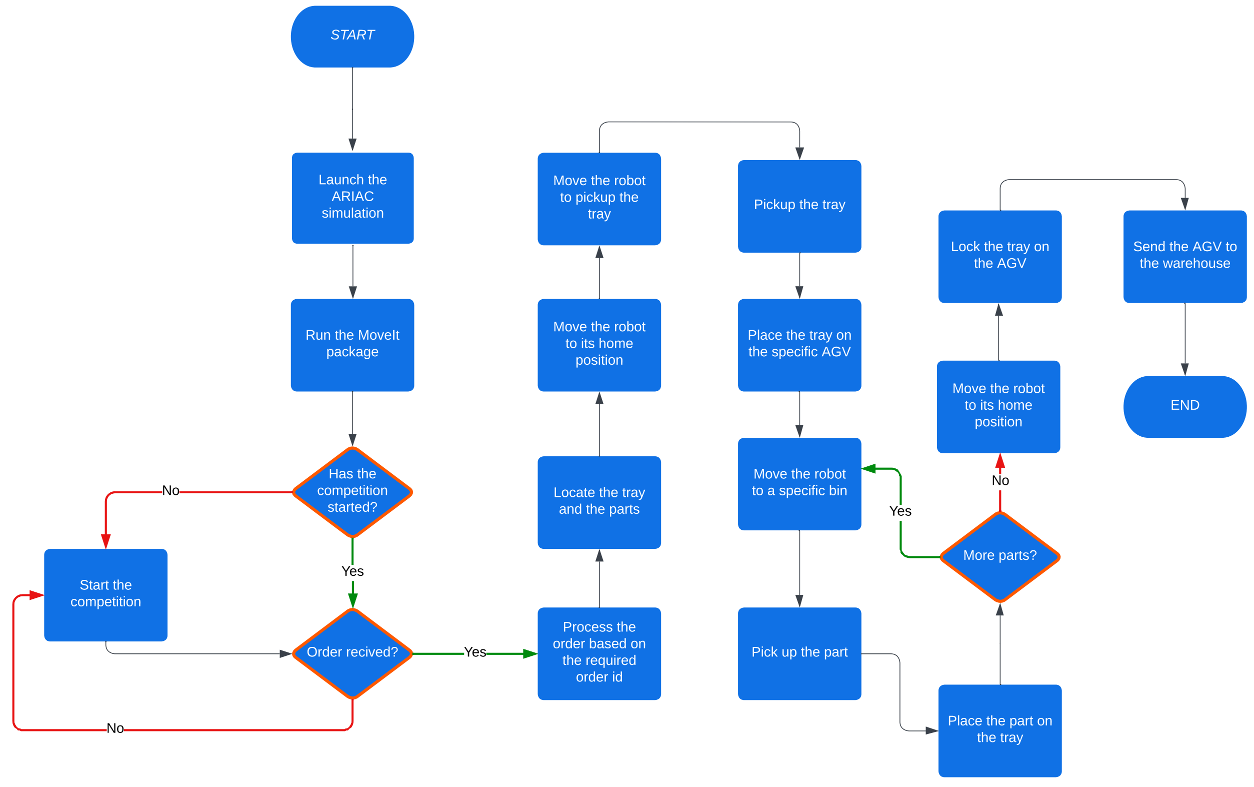
3. Part
   - It represents a part detected by the bin cameras
   - The attributes of this class store information about the part's type, color, and pose in the world coordinate frame.
   - It overrides the __str__ method to provide a string representation of the part's details.

4. RobotCommanderInterface
   - The RobotCommanderInterface class represents an interface for commanding the robot in the ARIAC system. It interacts with the other classes and services to control the robot's actions based on received orders and data from the cameras.
   - It declares and uses various service clients to send requests to the robot, such as moving to home configuration, entering/exiting the tool changer, picking up trays/parts, placing trays/parts on AGVs, etc.
   - The RobotCommanderInterface class subscribes to topics to retrieve order information, data from the cameras to detect the trays and parts, and competition state to receive updates and trigger appropriate actions.

3 Approach

This section discusses the approach of the project. The implementation of classes discussed in the previous section is extended here. To better understand the sequence of operations performed by the program, a flowchart detailing the workflow is provided below in figure 3.1. The subsequent text provides a more comprehensive explanation of each step. The project utilizes several object-oriented programming (OOP) concepts to design and implement an efficient and modular approach for the ARIAC competition.

Figure 3.1: Flowchart


START
Launch the ARIAC simulation
Run the MoveIt package
Has the competition started?
No
Yes
Start the competition
Order recived?
No
Yes
Process the order based on the required order id
Locate the tray and the parts
Move the robot to its home position
Move the robot to pickup the tray
Pickup the tray
Place the tray on the specific AGV
Move the robot to a specific bin
Pick up the part
Place the part on the tray
More parts?
Yes
No
Move the robot to its home position
Lock the tray on the AGV
Send the AGV to the warehouse
END

The simulation begins by running the launch file to run the Gazebo simulation with the ARIAC environment. The MoveIt package is run, which establishes motion planning, and the package sends commands to MoveIt. These two steps are required to run the node to perform the various tasks in the project.

As mentioned in the previous section, the classes used in the n commander do the entire kitting process. This node consists of various subscribers and service calls, the details of which will be discussed in this section. The first and foremost task is to start the competition. This is done by calling the service client $\_start\_competition\_client$. Once the competition is started, Orders are announced, cameras publish data, and robots can be controlled.

The orders are published to the t ariac/orders topic. The n commander has a subscriber to this topic. A callback function is defined, which receives the messages, i.e. the orders published to this topic. But only one among the published orders needs to be processed. The $orederInfo$ class of the node stores the required order. The order that should be processed is stored as a parameter in the params.yaml inside the config and it is then passed to the node during runtime.

The order published consists of order id, which AGV and tray to use, the type and colour of the parts needed to build a kit, and where to place the parts on the tray. Having identified the requirements of the kit, the next task is to find the location of the tray and the parts needed for the kit. The cameras placed above the table1 and table2 provide the locations of the trays placed on the table. The output from the cameras above the tables can be obtained by subscribing to the t /ariac/sensors/table1_camera/image and t /ariac/sensors/table2_camera/image respectively. The messages received from these topics contain the id of the detected tray, the pose of the tray and the camera in the f world.
The Tray class stores the details of the trays. The next task is to locate the parts needed to build the kit. The parts are placed in different bins referenced by "left bin" and "right bin ". There are cameras above the bins which are used to detect the parts in the bins.The cameras placed above the bins provide the type,color and locations of the parts in each bin. The output from these cameras are obtained by subscribing to the t /ariac/sensors/left_bins_camera/image and t /ariac/sensors/right_bins_camera/image, respectively. The Part class stores the details of the parts.
The task at hand is to identify the order to process, then task the robot to complete the kit and send the AGV to the warehouse. The RobotCommanderInterface class parses the order from the orderInfo class, receives the position of the tray to be picked up from the Tray class and gets the part information from the Part class. This class also contains the definitions of the client calls needed to move the floor robot to perform the process of kitting.
With the orders and the locations of the trays and parts in place, the floor robot initially moves to its home position. Then it traverses to one of the tool-changing stations where the gripper type of robot can be changed. Initially, the robot is equipped with a gripper which is used to pick up parts(blue colored gripper). This has to be changed to the one used to pick up a tray(purple colored). Then the robot picks up a tray and places it on the specified AGV. Following this, the robot moves towards a bin,picks up a part, and places it on the specific quadrant of the tray kept on the AGV.This step is repeated until all the required parts are placed on the AGV. Listing 3.1 provide the pseudo code of the service call function, which places a part in the tray.
After all the parts are placed in the tray, the robot returns to its home position, the tray on the AGV is locked so that it does not fall over when the AGV moves and then the AGV is sent to the warehouse.

Listing 3.1: Service call function for placing a part in the tray

```
Function Name: place_part_in_tray

Parameters: robot, agv, quadrant
Pseudocode:
Initialize PlacePartInTray request
```

```
If robot is "floor_robot", set request robot to FLOOR_ROBOT
Else, raise an error "Invalid robot name"
Set request agv and quadrant
Call place_part_in_tray_client service with request
If future result is not None, deactivate gripper
Else, error message "service call failed"
```

4 Challenges Faced

- Working on the project alone: As the sole developer of the project, I faced the challenge of handling all aspects of the project on my own. This included designing the system architecture, implementing the functionality, and resolving any issues that arose. Working alone required me to take on multiple roles and ensure effective time management and organization to meet project deadlines.

- Difficulty in starting the competition and determining callback groups: The ARIAC competition environment presented challenges when it came to starting the competition and properly organizing the callback groups. Determining the correct placement of callback groups was crucial to ensure the proper execution of tasks and prevent conflicts between different components of the system.

- Deadlock resulting in freezing of the competition: One of the significant challenges I encountered was a deadlock situation that caused the competition to freeze. Identifying and resolving the deadlock required thorough debugging and analyzing the code to identify the points of contention and implement appropriate logic to prevent deadlock scenarios.

- Figuring out how to implement the classes: Implementing the classes required careful consideration of the project requirements and the desired functionality. Designing the class hierarchy, defining the attributes and methods, and ensuring proper encapsulation and abstraction were crucial steps in the process. The details of the order,tray and parts were stored in the different classes.

- Slow simulation with a low real-time factor: During the development and testing phase, I faced the challenge of dealing with slow simulation speeds in Gazebo, with a real-time factor of around 0.40. The slow simulation affected the overall performance of the system and the responsiveness of the robot's actions. Also, the simulation was averaging about 5-10 frames per second. The 3D acceleration for the virtual machine could not be enabled and is a possible reason for poor performance.

5 Contribution

I took on the responsibility of developing the project alone. Hence, my contribution to the project as a sole developer involved the entire lifecycle of the project, from initial design to implementation, testing, and documentation.

6 Conclusion and Future Work

The aim of this project was to simulate a simplified version of the ARIAC 2023 competition. Working on this project has significantly enhanced my skills in working with ROS, Gazebo, and Python. The project served as a practical test of the knowledge gained in class, showcasing the value of the theoretical principles learned. Despite being a challenging experience, it was immensely rewarding, focusing on the importance of the taught concepts and their applicability in real-world scenarios. Navigating through this project necessitated a thorough understanding and application of ROS concepts, which initially seemed intimidating but progressively became more approachable.

Working on this project also provided the opportunity to refine my Python programming skills. Python was the primary language for implementing the ROS nodes, which required a robust understanding of its syntax, libraries, and best practices. The project served as a comprehensive Python programming exercise, from creating Python classes and methods to implementing algorithms and handling exceptions.

The scope of this project was primarily focused on the kitting phase of the process, which involved tasks related to the collection and organization of parts for assembly. It represents an important initial step in the larger cycle of a typical industrial production process. However, the potential for expanding this project to include subsequent phases is considerable, thus providing a more holistic solution.

An extension of this project would be to incorporate the assembly phase. This would involve programming the robots to not only collect parts but also assemble them according to specific guidelines. This could involve more intricate manipulation tasks and require additional sensor integration for finer control and precision. The robots might also need to interact with each other more closely during this phase, necessitating advanced coordination algorithms.

In addition, the project could be extended to handle combined tasks. This would involve robots performing both kitting and assembly tasks interchangeably based on real-time demands and scheduling constraints. This would require the development of more sophisticated task allocation and scheduling algorithms to ensure that the entire system operates efficiently.

7 Feedback

The course ENPM809E has been a highly enriching experience. It offered a well-balanced blend of foundational concepts and application-oriented tasks, which significantly contributed to the overall learning process. The transition from the basics of Python and Object-Oriented Programming to ROS2 Galactic was smooth and logically structured, making it easier to grasp and apply the complex concepts.

The assignments posed an appropriate level of challenge, which stimulated critical thinking and problem-solving skills. Each assignment was intricately designed with a strong focus on practical application, which further enhanced the understanding of the theoretical concepts. This hands-on approach facilitated active learning and made the coursework more engaging.

However, the workload was considerable, especially when factoring in the group assignments and the final project. In this case, it was compounded due to working alone on these tasks. While this allowed for a more in-depth understanding of the topics, it also posed additional challenges in terms of time management and work distribution.

The overall pacing of the course was well-balanced. It allowed for a comprehensive understanding of the topics covered while maintaining a steady progression through the syllabus. The course was well-structured, with each topic building on the last, which allowed for a more profound understanding of the intricacies of ROS and Python programming.

In conclusion, the course provided a solid foundation in Python and ROS2 Galactic and presented an optimal mix of theoretical knowledge and practical application. Overall, it was an excellent course that significantly contributed to my learning journey in robotics.

8 References

- Official ›ARIAC documentation
- Final project documentation by Prof.Zeid Kootbally
- Real world application 4 documentation by Prof.Zeid Kootbally