RWA-1

Building a Manufacuring Robotic Software System

March 2, 2022

Students:
Pulkit Mehta (UID: 117551693)

Darshan Jain (UID: 117454208)

Jeffin Johny Kachappilly (UID: 118293929)

Instructors:
Z. Kootbally; C. Schlenoff

Group:
5

Semester:
Spring 2022

Course code:
ENPM663

Contents

1 Introduction

The document focuses on developing a coherent structure for order processing for the Agile Robotics for Industrial Automation Competition(ARIAC). There are mainly two tasks involved, kit building or kitting and assembly based on the received orders. There are two kinds robots in the given environment namely, kitting(UR10 arm) and the gantry robot. There are also Automated Guided Vehicles(AGV) in the world that are used to ship the products from kitting station to assembly station. The data constituting the details of the orders needs to be retrieved and stored in appropriate data structures so as to initiate the relevant processes. The scope of this document revolves around the architecture used for processing and providing a holistic view of entire sequence of processes involved.

2 Architecture and Design

The software architecture needs to cope up with dynamic orders and various other bottlenecks or environmental failures in order to optimize the order processing time. Preferably, a hybrid architecture may be able to plan and optimize the tasks along with handling the dire situations of sensor failures or any other complication arising from the change in environmental conditions.

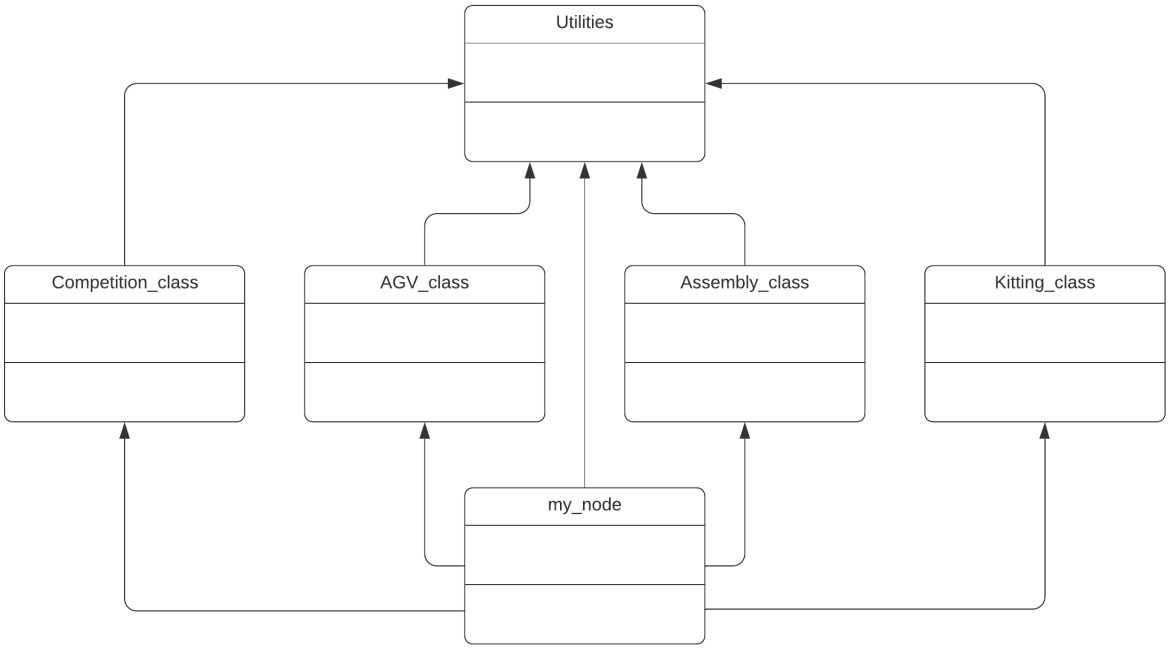

Figure 1: UML class diagram

The figure 1 depicts the division the main structure implemented. It constitutes the following:

- Classes
  - Competition class: It can be considered as the main class of the entire program. It checks

various topics for the data published on the topic and stores the data in appropriate variables for future need.

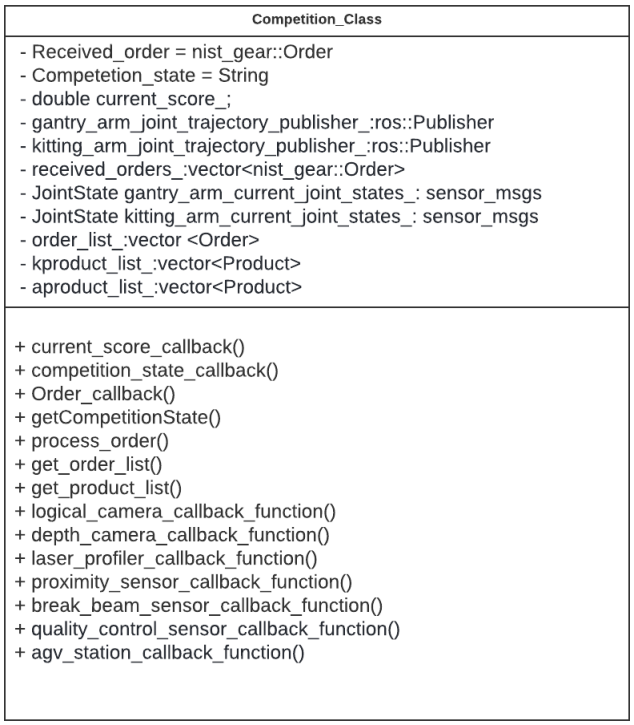

Figure 2: Competition class

It consists of following sub functions, as shown in Fig 2:

* Order_callback: The following function listens to the "/ariac/orders" topic on which the orders are published and stores the data in the appropriate variables which can be accessed in the entire program using various getter functions.
* Competition_state_callback: The function listens to the "/ariac/competition_state" topic on which the state of the competition is published and sets the value in a variable which is accessed later. The following are the three different state values:
  - init - Before the competition starts.
  - go - During the competiton.
  - done - After the competition has ended.
* Sensor callback_functions: There are various sensor call back functions in the competition class which listen to the data published on various sensor topics to give the program the details about the parts in the environment. Some functions also return the type and the location of the part in the environment.
* process_order: Stores the data of the products from the incoming orders in two different lists for the kitting and assembly process.
* Accessors: There are various accessors in the class that help to access the private data members of the class in the main function. They are as follows:
  - get_Competition_state: Returns the current state of the competition.
  - get_order_list: Returns the vector that contains the order stored in a data variable.

- get_product_list: Returns the list of products that have to be delivered to complete the shipment.

* AGV class: This is the class that controls the part of the program pertaining to the AGVs. It contains four service clients one for each AGV It also contains the following methods, as in Fig 3:

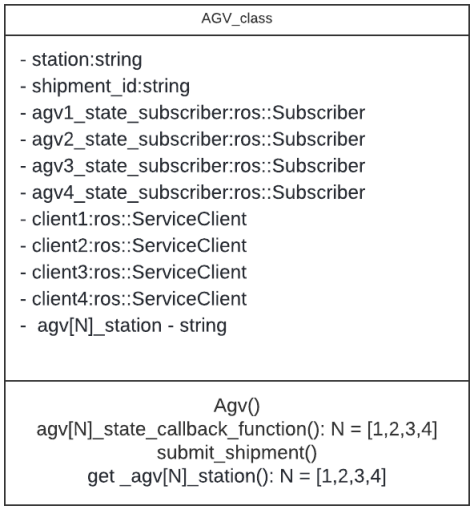

Figure 3: AGV class

- Agv[N] state callback function: The following functions subscribe to the ariac/agv[N]/state topic where N = [1,2,3,4] which publishes the state of AGV at the time of callback.
- Agv[N] station callback function: The following functions subscribe to the ariac/agv[N]/station topic where N = [1,2,3,4] which publishes the current station the AGV is at the time of callback.
- submit shipment: The function is used to submit the AGV to the Assembly station based on the agv_id mentioned in the order.

- Functions
  - Utilities function: Fig4 shows the structs created as per the format of the order messages, the detailed structure for which is depicted in Fig 6. It also contains all the libraries including the various message types required for subscribers and services.

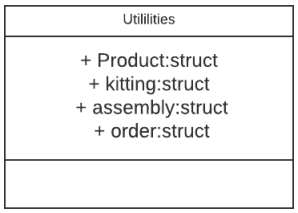

Figure 4: Utils

  - My Node Function: This function can be considered as the main entry point to the program. Fig5 shows the methods in the function.

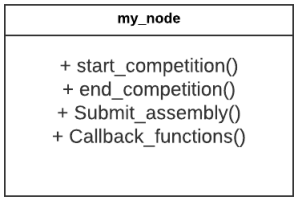

Figure 5: My node

* It initializes the ROS handle, object for the competition and the AGV class.
* It starts the competition.
* It initializes various subscribers and calls the callback functions when something is published on the topic.
* Currently it also submits the order to the assembly station.
* It ends the competition once all the orders are submitted.

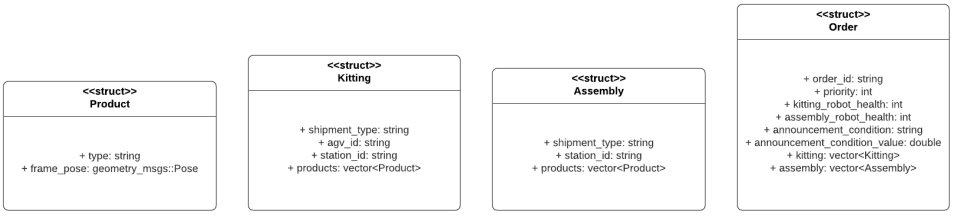

Figure 6: Structs created in order to store information from order message.

3 Process Flow

3.1 Kitting Process

The process of segregating the products according to certain criteria is called kit building or kitting. The Fig7 depicts the sequence diagram for kitting. The following steps were considered to be completed before starting the kitting process for this diagram:

- The competition has already started.
- The conveyor belt has started moving.
- All the parts required for making a kitting shipment are available.
- All the AGVs are at their respective kitting stations at start of each order.

The detailed steps for kitting being:

- Step (1): When a part crosses the proximity sensor , the function gets activated and the kitting robot gets notified.
- Step (1.1): The arm will pick the part from conveyor belt and place it sequentially(based on bin number) in bins . For this step the pose of the part is not considered while placing it in bins. The bins are designed to hold a maximum of 4 parts, so when a bin reaches its limit, the arm will place the next part in higher order bin.

- Step (1.2): In this step, the logical camera determines the type, pose and colour of the parts available in bins.
- Optional: This is a if-then block . When the logical camera outputs that enough number of parts are available for particular shipment, it gets execute.
- Step(1.3): The arm performs the picking of necessary parts from the bins and goes to the AGV provided in order. And it places the parts in correct position and orientation on the AGV tray.
- Step (1.4): This function gets executed only when a faulty part exists in that AGV tray.
- LOOP : This looping block gets executed , only if [Step (1.4)] detects a faulty component. And the loop can only be exited, if all the parts present inside the tray are not faulty.
- Step (1.5): In this step, the arm picks the faulty component from the AGV tray and places it in the faulty collector. This makes the order incomplete for shipment.
- Step (1.6): The logical camera checks for the missing component inside bins.
- Step (1.7): The arm picks the missing part from bin and places it inside the corresponding AGV tray.
- Step (1.8): Quality sensor checks again, whether the newly placed part is faulty or not. If it evaluates to faulty, the loop executes again.
- Step (1.9): This function deals with the shipment of AGV to the assembly station mentioned in the order.
- Step (1.10) : This marks the end of the kitting shipments.

Below are some of the scenarios which are not considered for this kitting and are part of future scope of work, they are:

- How to act when a high priority order comes in.
- Use of Gantry robot for picking operations from belts, so that parts doesn't fall off from the belt (as only limited parts are available).
- Steps to be taken when the gripper of kitting arm is faulty.

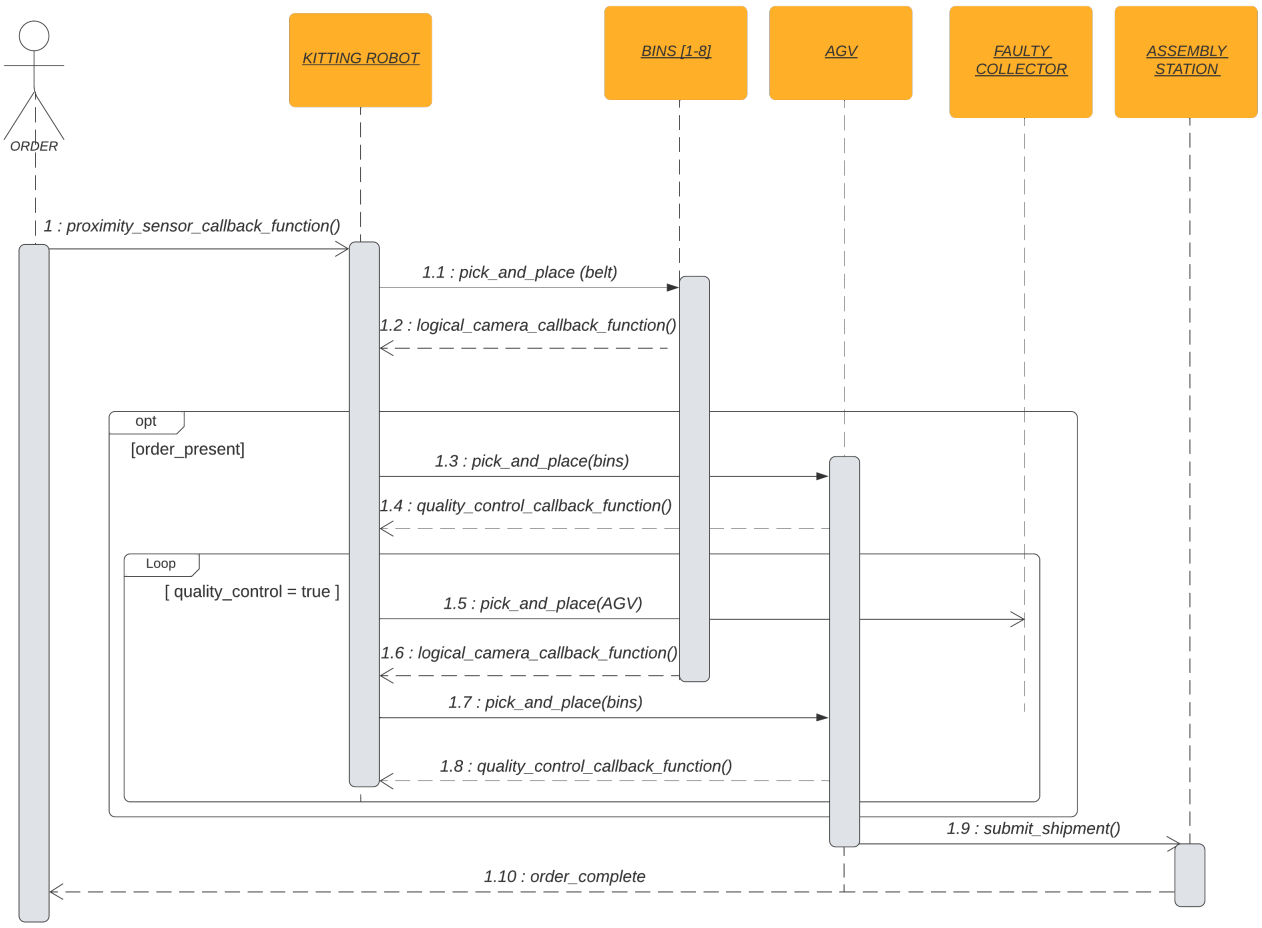

Figure 7: Kitting Sequence flow

3.2 Assembly Process

The assembly process involves placing all the parts in their specified location in the briefcase. The steps involved for assembly shipment are considerably less compared to kitting process. The 'objects' interacting for this diagram are 'Assembly robot' and 'Briefcase'. Assumptions considered for this shipment diagram are:

- At the beginning of the order, all the AGVs are at the required assembly stations.
- Only one assembly shipment is considered for this order.

The steps for the assembly process are depicted in Fig8 and the details are explained below,

- Step (1): This function gets the current station where the AGV will be residing.
- Optional : This is a if-then block. This block gets executed when station position obtained from [Step (1)] is same as station_id given in order details.

- Step (1.1): Here the gantry robot moves to the station location and picks the part from AGV tray. After this the arm will place it in correct pose inside briefcase.
- Step (1.2): After all the parts of a particular shipment has been placed, the assembly is submitted.
- Step (1.1.1): This marks the fulfillment of that particular order.

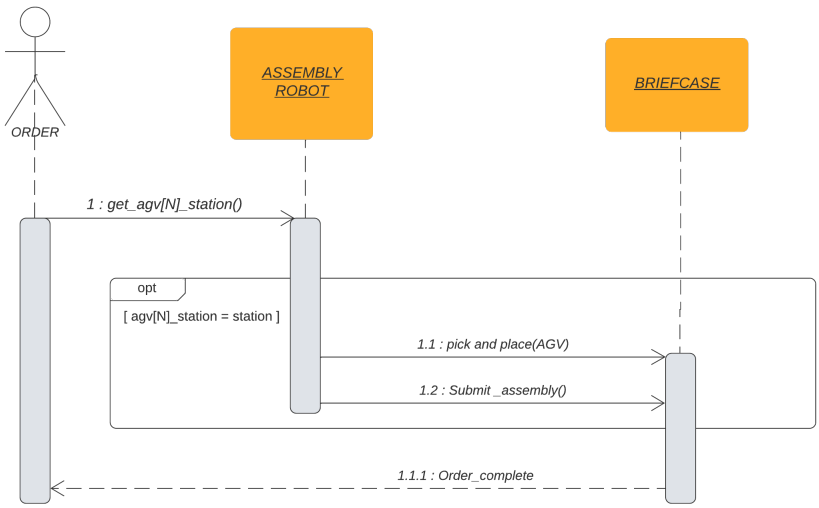

Figure 8: Assembly Sequence flow

4 Challenges

- The processing for multiple orders after retrieving them from the order message sequentially becomes quite a challenge if the data is not stored correctly in the defined data structure. This process needs to be optimized for it to happen efficiently.
- The interfacing of several different classes and calling the service clients after the previous one performs the service successfully. The services for submitting the shipments for kitting and assembly were called by checking some flags indicating whether the service was called before or not. This approach needs to be further refined or a different feedback needs to be added to the process flow.