



**Z. Kootbally & C. Schlenoff**

Spring 2023

UMD

College Park, MD

**RWA2 (v0.0)**

---

# ENPM663: Building a Manufacturing Robot Software System


---

Due date: **Wednesday, March 15, 2023, 4 pm**

# Contents

<b>1</b>	<b>Updates</b>	<b>3</b>
<b>2</b>	<b>Conventions</b>	<b>3</b>
<b>3</b>	<b>Prerequisites</b>	<b>3</b>
<b>4</b>	<b>Assignment Tasks</b>	<b>4</b>
4.1	Retrieve Orders . . . . .	5
4.2	Locate Parts . . . . .	6
4.2.1	Kitting Task and Combined Task . . . . .	6
4.2.2	Assembly Task . . . . .	8
4.3	Identify Challenges . . . . .	8
4.4	Task-level Planning . . . . .	9
4.4.1	Which robot(s) should work on a task? . . . . .	9
4.4.2	Where to write the methods? . . . . .	11
4.5	Running the Trial . . . . .	11
<b>5</b>	<b>Submission</b>	<b>13</b>
<b>6</b>	<b>Approach</b>	<b>14</b>
<b>7</b>	<b>Grading Rubric</b>	<b>14</b>









# 1 Updates

This section describes updates added to this document since its first release. Updates include addition to the document and fixed typos. The version number seen on the cover page will be updated accordingly.  There may be some typos even after proofreading the document. If this is the case, please let me know.






- **v0.0**: Original release of the document.

# 2 Conventions

In this document, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

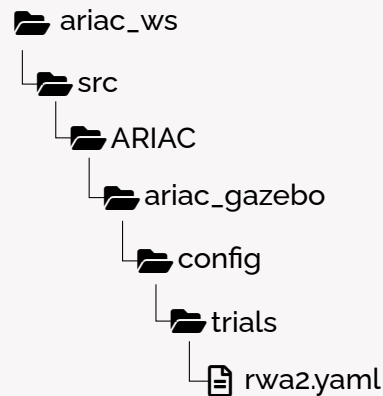
- This is a  file.txt
- This is a  folder
- This is a  package
- This is an important note 
- To do 
- This is a warning 
- This is a [link](#)
- This is a  topic
- This is a  service

# 3 Prerequisites


- This assignment consists of the insufficient parts challenge. Read the [documentation](#) to learn more about this challenge.
- Retrieve the file  rwa2.yaml from Canvas and place it in the  config folder as shown in [Figure 3.1](#). This file is a trial configuration file and consists of 3 orders that will be announced at different time intervals. The first order is a kitting task and is announced when the competition is started. The second order is an assembly task and will be announced 20 s after the competition is started. The third order is a combined task and will be announced 50 s after the competition is started.  Do not modify the time announcement of each Order in this assignment.
- The following Messages are required in this assignment:
  -  ariac\_msgs/msg/Order
  -  ariac\_msgs/msg/CompetitionState

- `m ariac_msgs/msg/BinParts`
- `m ariac_msgs/msg/ConveyorParts`
- The following Service is required in this assignment:
  - `s ariac_msgs/srv/SubmitOrder`
- To start the environment with `rwa2.yaml`
  - `> ros2 launch ariac_gazebo ariac.launch.py trial_name:=rwa2`

Figure 3.1: Trial file location.



## 4 Assignment Tasks

-  This is a group assignment.
- Resources needed for the assignment can be found in the official [documentation](#).
  - The latest documentation is under *competition\_release\_staging*.

This assignment requires that you augment the package you created in RWA1, which allowed you to 1) start the competition, 2) store orders published by the ARIAC manager, 3) submit orders, and 4) end the competition.

In this assignment, you need to:

1. Start the competition (RWA1).
2. Retrieve Orders (RWA1).
3. Locate parts required in the Orders.
4. Identify challenges.
5. Call functions to do kitting and assembly.
6. Submit Orders (RWA1).
7. End the competition (RWA1).

## 4.1 Retrieve Orders



The discussion in this section should help you identify a task from a published Order and how to store the task internally in your program. If you did not properly do these steps in RWA1, now is a good time to fix this.

Each order has one and only one task (kitting, assembly, or combined). When an order is published on `t /ariac/orders`, the fields for each task are provided. For instance [Listing 4.1](#) shows an order published on `t /ariac/orders`. One can see the the information for *kitting\_task* (line 5), *assembly\_task* (line 14), and *combined\_task* (line 18). To know which one of the three tasks is in the Order, you need to compare the value of the field *type* (line 3) with the constants *KITTING*, *ASSEMBLY*, and *COMBINED* from `Order.msg` (see [Listing 4.2](#)). In this case, the value of *type* is 0, meaning that the task within this Order is a kitting task. Once the task is identified, you need to properly create a C++ or Python object from your class representing a kitting task and then store the object in a data structure.

Listing 4.1: Published order.

```
1  ---
2  id: KIT10H56
3  type: 0
4  priority: false
5  kitting_task:
6    agv_number: 4
7    tray_id: 3
8    destination: 3
9    parts:
10     - part:
11         color: 4
12         type: 11
13         quadrant: 1
14  assembly_task:
15    agv_numbers: []
16    station: 0
17    parts: []
18  combined_task:
19    station: 0
20    parts: []
21  ---
```

Listing 4.2: Definition of Order.msg.


```

uint8 KITTING=0
uint8 ASSEMBLY=1
uint8 COMBINED=2

string id
uint8 type # KITTING, ASSEMBLY, or COMBINED
bool priority
ariac_msgs/KittingTask kitting_task
ariac_msgs/AssemblyTask assembly_task
ariac_msgs/CombinedTask combined_task

```



- Retrieve the type of the task from the published Orders.
- Once the task is identified, create a C++ or Python object and store the task internally in a data structure. The Message definition for each task can be retrieved with:
  - `ros2 interface show ariac_msgs/msg/KittingTask`
  - `ros2 interface show ariac_msgs/msg/AssemblyTask`
  - `ros2 interface show ariac_msgs/msg/CombinedTask`
-  Your Python/C++ classes can be written to replicate the structure of `KittingTask.msg`, `AssemblyTask.msg`, and `CombinedTask.msg`.

## 4.2 Locate Parts

Once you read the Orders in, you have to locate parts needed for the task within each Order.

- For a kitting task, parts can be found in bins and/or on the conveyor belt.
- For an assembly task, parts are already placed on one or two AGVs.
- For a combined task, parts can be found in bins and/or on the conveyor belt.

### 4.2.1 Kitting Task and Combined Task



- Read about [kitting task](#).
- Read about [combined task](#).

The [Topics](#) `/ariac/bin_parts` and `/ariac/conveyor_parts` provide information on part locations and can be used for kitting and combined tasks. `/ariac/bin_parts` provides the part color, part type, and part quantity for each bin. The pose of parts in bins is not provided on this Topic and should be retrieved using cameras (covered

in RWA3). Listing 4.3 shows only one Message published on `t /ariac/bin_parts` for `rwa2.yaml`. The information published on this Topic is not updated, that is, if parts are removed from bins, the published Messages stay the same as they do not reflect the changes. In other words, Messages published on this Topic provide initial bin information.

The Topic `t /ariac/conveyor_parts` is similar to `t /ariac/bin_parts`, except it is applied to the conveyor belt. Similarly to `t /ariac/bin_parts`, parts removed from the conveyor belt are not updated in the published Messages. One Message published on `t /ariac/conveyor_parts` for `rwa2.yaml` is shown in Listing 4.4.

Listing 4.3: Topic /ariac/bin\_parts

```
---
bins:
- bin_number: 2
  parts:
  - part:
    color: 4
    type: 11
    quantity: 4
- bin_number: 6
  parts:
  - part:
    color: 2
    type: 11
    quantity: 2
  - part:
    color: 2
    type: 12
    quantity: 2
  - part:
    color: 3
    type: 13
    quantity: 2
  - part:
    color: 3
    type: 10
    quantity: 2
---
```

Listing 4.4: Topic /ariac/conveyor\_parts

```
---
parts:
- part:
  color: 1
  type: 13
---
```

```
quantity: 2
- part:
  color: 0
  type: 12
  quantity: 2
---
```



Use one or multiple data structures to store information published on the Topics `t /ariac/bin_parts` and `t /ariac/conveyor_parts`. In the callback functions for these Topics, retrieve only one Message from each Topic and store this Message. There is no need to store the same information each time a Message is published as the Messages published on each of these Topics are unchanged.

### 4.2.2 Assembly Task



Read about [assembly task](#).

For an assembly task, parts needed for assembly are already located on one or two AGVs. You only need to move the AGV(s) to the correct assembly station and then do assembly.

## 4.3 Identify Challenges

Some challenges can be identified when the competition starts and other challenges can only be identified when your robots start doing pick-and-place. The [insufficient parts challenge](#) occurs when there is not enough parts in the workcell to perform a kitting task. This challenge can be identified once an Order is announced. This challenge does not apply to assembly and combined tasks.

`rwa2.yaml` was designed as to not have enough parts to perform the kitting task (a blue battery is missing). When a part is missing, you have to submit the Order as incomplete.



- After reading a kitting task from `t /ariac/orders`, use `t /ariac/bin_parts` and `t /ariac_conveyor_parts` to identify if there is an *insufficient parts* challenge in the trial. The field *quantity* can be used to check if there are enough parts to complete a kitting task.
- It can also happen that the information you get from these Topics may not be enough to know if all the parts are available for a kitting task. Imagine you have



a trial with two kitting tasks. The first task is announced and there are enough parts in the workcell for the first task. The second task is announced much later but some parts needed for the second task have been used for the first task and there are not enough parts for the second task. There is a need to keep track of parts you have used in each task and then decrease your part storage each time a part is used in a task. Although this situation is not present in the trial for this assignment, you should implement it as well.

---

## 4.4 Task-level Planning

Task-level planning consists of high-level actions to complete tasks. These actions will eventually be translated to low-level commands (motion planning), the latter will be covered later.

For each one of the three tasks in ARIAC, you need to write a set of methods to complete the task. Before writing these methods, you have to figure out two things:

1. Which robot(s) should work on a given task?
2. Where to write the methods?



### 4.4.1 Which robot(s) should work on a task?

- Only the ceiling robot can perform the assembly task. There is not much choice as to which robot to choose for this task.
- Both robots can perform a kitting task. The possibilities are:
  - Only the floor robot performs the kitting task.
  - Both the floor and ceiling robots perform the kitting task. If both robots work on a kitting task, you need to decide which robot picks up parts from bins and which one picks up parts from the conveyor belt.
- Both robots can perform a combined task. The possibilities are:
  - Only the ceiling robot performs a combined task (kitting + assembly).
  - The floor robot performs kitting and the ceiling robot performs assembly.
  - Both the floor and ceiling robots perform kitting and the ceiling robot performs assembly.



---

1. Make a copy of this [activity diagram](#).

-  Do not directly work on this diagram as you will modify it for everybody.
- Once a copy is made, you can edit your version with the Google drive plugin [diagrams.net](#).
- Complete the activities for CombinedTask, AssemblyTask, and KittingTask (fill out the three rightmost columns in the diagram).
- When you are done with the activities for the three tasks, create the folder  `document` in your package and save the diagram as a jpg file in this folder.

---

The activity diagram for each task must be carefully implemented as these diagrams are used in your next task to implement the functionalities. Here is a reminder/hint of what you *at least* need to have in your diagrams (in no specific order).


- Identify whether or not there is an insufficient parts challenge for the trial.
- KittingTask:
  - From the Order, identify parts needed for kitting.
  - From the Order, identify the tray needed for kitting.
  - Use cameras/sensors to locate parts needed for kitting (RWA3).
  - Use cameras/sensors to locate the tray needed for kitting (RWA3).
  - Pick up the tray (RWA4).
  - Place the tray on AGV (RWA4).
  - Pick up parts from bins and/or from the conveyor belt (RWA4).
  - Place parts in tray (RWA4).
  - Change gripper (RWA4).
  - Lock AGV (RWA4).
  - Move AGV to warehouse (RWA4).
  - Submit the Order.
- AssemblyTask:
  - From the Order, identify the AGV(s) with parts needed for assembly.
  - From the Order, identify the station where assembly needs to be performed.
  - Lock AGV(s) (RWA4).
  - Move AGV(s) to station (RWA4).
  - Use cameras/sensors to locate parts on AGV(s) (RWA3).
  - Pick parts from AGV(s) (RWA4).
  - Place parts in insert (RWA4).
  - Submit the Order.
- CombinedTask:
  - From the Order, identify the station where assembly needs to be performed.
  - From the Order, identify parts needed for assembly.

- Use cameras/sensors to locate parts needed for assembly (RWA3).
- Use cameras/sensors to locate tray for kitting (RWA3).
- Pick up a tray (RWA4).
- Place the tray on AGV (RWA4).
- Pick up parts from bins and/or from the conveyor belt (RWA4).
- Place parts in tray (RWA4).
- Change gripper (RWA4).
- Lock AGV (RWA4).
- Move AGV to station (RWA4).
- Locate parts on AGV (RWA3).
- Pick parts from AGV (RWA4).
- Place parts in insert (RWA4).
- Submit the Order.

#### 4.4.2 Where to write the methods?

Your next task for this assignment is to write methods to complete each ARIAC task. One of the requirements for this course is to have programming skills. This task is a good opportunity to put these skills to practice. For this assignment you are only required to print a message in the terminal whenever these methods are executed. In future assignments you will augment the current packages with sensor/camera information and with motion planning.


There are different ways to structure your program to complete tasks. Three propositions are given below but there are other ways to structure your package. Feel free to structure your package as you see fit.

- Proposition#1: Create the class *ARIACTask*. This proposition is similar to the one [we implemented in ARIAC](#) for the package  `test_competitor`
- Proposition#2: Create a base class *Robot* and two derived classes *CeilingRobot* and *FloorRobot*.
- Proposition#3: Create two base classes, *CeilingRobot* and *FloorRobot*.



1. Using your activity diagrams, augment your package with classes and methods to complete tasks.
2. In each method, write one single line of code which prints a message in the terminal, showing what is happening when the method is executed.

## 4.5 Running the Trial

Your final task for this assignment is to run your Node(s) to complete each Order in  `rwa2.yaml`. [Listing 4.5](#) shows an example of what your program should output for

the kitting task.

Listing 4.5: Terminal outputs for KittingTask

```
1 -----
2 starting trial
3 -----
4 received order KIT10H56: KittingTask
5 -----
6 insufficient parts challenge: Missing blue battery
7 -----
8 [FloorRobot] pick green regulator from belt
9 -----
10 [FloorRobot] place green regulator in bin 8 - slot 1
11 -----
12 [FloorRobot] pick red sensor from conveyor belt
13 -----
14 [FloorRobot] place red sensor in bin 8 - slot 2
15 -----
16 [FloorRobot] change to tray gripper
17 -----
18 [FloorRobot] pick tray id 3
19 -----
20 [FloorRobot] place tray on agv 4
21 -----
22 [CeilingRobot] pick green regulator from bin 8
23 -----
24 [CeilingRobot] place green regulator in quadrant 2
25 -----
26 [FloorRobot] change to part gripper
27 -----
28 [FloorRobot] pick purple pump from bin 2
29 -----
30 [FloorRobot] place purple pump in quadrant 1
31 -----
32 [FloorRobot] pick red sensor from bin 8
33 -----
34 [FloorRobot] place red sensor in quadrant 4
35 -----
36 lock agv 4
37 -----
38 move agv 4 to warehouse
39 -----
```

```

40 submit order KIT10H56
41 -----

```

Some comments on [Listing 4.5](#):




- Line 6: The *insufficient parts* challenge is detected at the beginning of the competition and a message about which parts are missing is displayed. When we grade your assignment, the insufficient parts challenge may include more than one parts. On the other hand, we may not include an insufficient parts challenge at all in the trial used during grading. Make sure you include these cases.
- Lines 8–14: Parts are picked up from the conveyor belt and placed in an empty bin. Once parts on the belt reach the end of the belt, these parts disappear and do not come back. One good practice is to grab those parts as soon as possible and temporarily store them in an empty bin for later. You know which bins are empty from `t /ariac/bin_parts`. There will always be at least two bins that are empty in a given trial. As for the slots within a bin, you are free to hardcode them.
- Line 16: Each robot starts with a part gripper, which can be used to pick up parts only. To pick up trays, your robots need a tray gripper.
- Lines 22 and 24: In this example, the ceiling robot is working with the floor robot on the kitting task. This is just an example of two robots working together. You may have decided in your activity diagram to only use the floor robot for kitting and the ceiling robot only for assembly, and this is fine.
- Lines 26–34: The floor robot changes gripper and place the remaining parts in the tray.
- Line 36: Locking the AGV prevents parts and tray from moving while the AGV is moving. To lock the AGV, a service is used. This service will be used in RWA3 or RWA4.
- Line 38: A service is used to move an AGV to the warehouse. This service will be used in RWA3 or RWA4.
- Line 40: Call the service to submit an Order. This service was implemented and used in RWA1.



1. Use [Listing 4.5](#) as an inspiration for what to output on the screen for the kitting, the assembly, and the combined tasks.



## 5 Submission

You can either compress and submit your package on Canvas or invite us (anikk94 and zeidk) as collaborators to your private Github repository. If you submit your package,

make sure you only submit your package and not the whole workspace. There is no deadline extension for this assignment and late submissions will be penalized. We often have situations where students forget to include a bug fix in the submission and they ask permission to submit again. We do not grant permission to resubmit the assignment passed the due date and time. Before you submit the assignment, delete  build,  log, and  install, rebuild your package, and run your Node(s) again to make sure everything works.



---

In the folder  document, which should already contain a jpg file of your diagram, create the file  instructions.txt. In this file, add the instructions on how to run your Node(s). Make sure the instructions are correct to avoid any delay in grading the assignment.

---

## 6 Approach

The difficulty level of this assignment has increased since RWA1. This assignment is very important as it brings you closer to the final structure of your package. You are given 2 weeks to complete the assignment, which is enough time if you work as a team. If one team member does not contribute, this will greatly impact the end result. Everyone has to contribute.

## 7 Grading Rubric

- This assignment is worth **30 pts**.
  - **15 pts** will be awarded for the activity diagram. The instructors will personally grade the diagram.
  - **15 pts** for the outputs from your package. Remember that we will use a different trial file during grading and the outputs should match the tasks in the trial.