Spring'22

Zeid Kootbally
Craig Schlenoff
University of Maryland
College Park, MD

RWA-2 (v0.0.1)

ENPM663: Building a Manufacturing Robotic Software System

Due date: **Wednesday, March 12, 2022, 12 am**

Contents

1 Updates

This section describes updates added to this document.

2 Disclaimer

To the best of my knowledge I have included all the information needed to complete this assignment. If some parts are confusing or missing, then let me know and I will update this document and re-upload it on Canvas. The version number will also be updated.

3 Prerequisites

1. ARIAC 2021 must be installed on your system (Ubuntu 18.04 + ROS Melodic). Instructions to install ARIAC 2021 can be found here.
2. You must have completed Lecture 6.
3. You must reuse the package created for RWA-1.
4. You must be part of a group.

4 Setup

From now on you will start ARIAC from your own package. This requires the creation of a `launch` folder (if you do not have one yet) and the creation of a `config` folder in your package.
Note: In this document group 1 is used as an example. Replace each instance of **group1** with your group number.

1. First, rename the package you created for RWA-1. For instance, if your package name is group1_rwa1, rename it to group1_rwa2. After you rename your package, you need to change the package name in `[package.xml]`:

   ```
   <name>group1_rwa2</name>
   ```

   and in `[CMakeLists.txt]` (change it even if you are not using C++):

   ```
   project(group1_rwa2)
   ```

2. Next, move your package to the same catkin workspace as ARIAC packages (if it is not already there). You should have the structure below.

```
ariac_ws
├── ARIAC
├── ariac-gazebo_ros_pkgs
└── group1_rwa2
```

3. Create a `launch` folder (if it does not already exist), a `script` folder, and a `config` folder in your package. In `config`, create `trial_config` and `user_config`.

```
group1_rwa2
├── launch
├── config
│   ├── trial_config
│   └── user_config
└── script
```

4. The next step consists of copying some files in your package as to have the following file structure.

```
group1_rwa2
├── launch
│   └── [ariac.launch]
├── config
│   ├── trial_config
│   │   └── [rwa2_trial.yaml]
│   └── user_config
│       └── [group1_config.yaml]
└── script
    └── [part_spawner.sh]
```

   (a) Copy the file `[sample_environment.launch]` (from `nist_gear/launch` in your `launch` folder and rename it to `[ariac.launch]`.

(b) Copy [rwa1_sensor.yaml] in user_config and rename it to [group1_config.yaml].
(c) Copy [rwa1_trial.yaml] in trial_config and rename it to [rwa2_trial.yaml].
(d) Get [part_spawner.sh] from Canvas and place it in the script folder.

5. Modify [ariac.launch] to use your YAML files (modify lines 8 and 9 in this file).

```
1  <node name="ariac_sim" pkg="nist_gear" type="gear.py"
2       args="
3         $(arg verbose_args)
4         $(arg state_logging_args)
5         $(arg gui_args)
6         $(arg load_moveit_args)
7         $(arg fill_demo_shipment_args)
8         -f $(find group1_rwa2)/config/trial_config/rwa2_trial.yaml
9         $(find group1_rwa2)/config/user_config/group1_config.yaml
10        " required="true" output="screen" />
```

6. Check that you can start the simulation with:

```
> catkin build
> source ~/ariac_ws/devel/setup.bash
> roslaunch group1_rwa2 ariac.launch
```

5 Objectives

This assignment consists of writing/editing YAML files and modifying the catkin package created in RWA-1 to detect agility challenges. The faulty gripper challenge and the dropped part challenge will not be addressed in this assignment. The robots should be operational for these two challenges, which we will cover in next lecture.

6 Tasks

The following tasks must be performed for this assignment.

6.1 Sensor/Camera Setup

Note: The estimated time to complete this task is 30-40 min (that is if you are not watching TV at the same time).

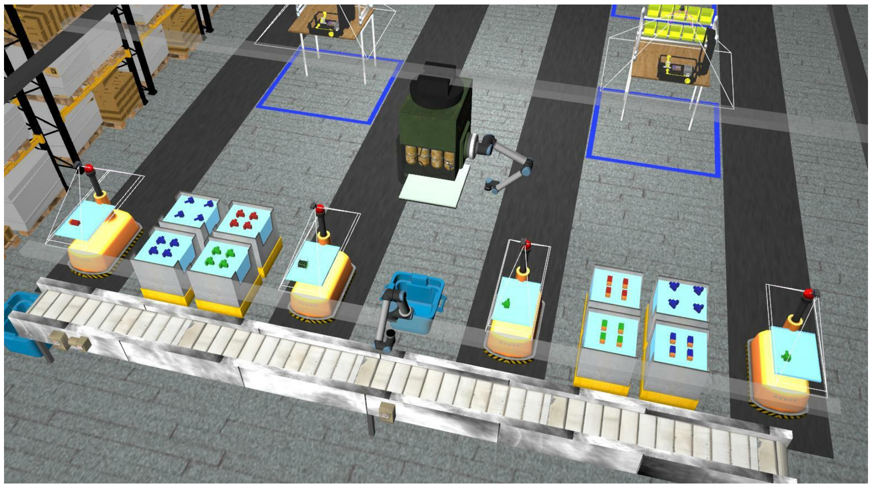


Figure 1: Part locations in bins and on AGVs.

The user config file provided in class has some locations of cameras and sensors. However, this file was provided only as an example and does not contain all the needed cameras and sensors to complete ARIAC. Your mission, should you choose to accept it, is to add logical cameras or modify the location of existing ones in [group1_config.yaml] so that these cameras cover all the areas where parts can be placed in ARIAC.

1. Make sure you have the option `--visualize-sensor-views` in your launch file (see line 8 in the snippet below) so you can see the cameras/sensors fields of view in Gazebo.

```
1  <node name="ariac_sim" pkg="nist_gear" type="gear.py"
2       args="
3         $(arg verbose_args)
4         $(arg state_logging_args)
5         $(arg gui_args)
6         $(arg load_moveit_args)
7         $(arg fill_demo_shipment_args)
8         --visualize-sensor-views
9         -f $(find group1_rwa2)/config/trial_config/rwa2_trial.yaml
10        $(find group1_rwa2)/config/user_config/group1_config.yaml
11        " required="true" output="screen" />
```

2. Modify [rwa2_trial.yaml] to spawn 4 parts in each bin.
   - For reference, here is how to spawn 4 blue batteries in bin1:

```yaml
models_over_bins:
   bin1:
     models:
       assembly_battery_blue:
         xyz_start: [0.2, 0.2, 0.0]
         xyz_end: [0.4, 0.4, 0.0]
         rpy: [0, 0, 0]
         num_models_x: 2
         num_models_y: 2
```

- Parts to spawn in bins are provided in Table 1.

bin1	blue battery x 4
bin2	green battery x 4
bin3	red battery x 4
bin4	blue sensor x 4
bin5	blue sensor x 4
bin6	green sensor x 4
bin7	red sensor x 4
bin8	blue regulator x 4

Table 1: Parts in bins.

3. Modify [rwa2_trial.yaml] to spawn 1 part on each AGV.
   - For reference, here is how to spawn 1 green regulator on agv1:

```yaml
agv_infos:
  agv1:
    location: ks1
    products:
      part_0:
        type: assembly_regulator_green
        pose:
          xyz: [0.1, 0.1, 0]
          rpy: [0, 0, 0]
```

   - Parts to spawn on AGVs are provided in Table 2.

agv1	green regulator x 1
agv2	green regulator x 1
agv3	green pump x 1
agv4	red battery x 1

Table 2: Parts on AGVs.

4. Modify [rwa2_trial.yaml] to spawn 2 parts in each briefcase.
   - For reference, here is how to spawn 1 green battery and 1 red pump in briefcase1:

```
models_over_stations:
  as1:
    models:
      assembly_battery_green:
        xyz: [-0.032465, 0.174845, 0.15]
        rpy: [0, 0, 'pi']
      assembly_pump_red:
        xyz: [0.032085, -0.152835, 0.15]
        rpy: [0, 0, 0]
```

   - Parts to spawn in briefcases are provided in Table 3.

briefcase1	1 blue battery + 1 blue pump
briefcase2	1 blue battery + 1 blue pump
briefcase3	1 blue battery + 1 blue pump
briefcase4	1 blue battery + 1 blue pump

Table 3: Parts in briefcases.

5. Place logical cameras above bins. It is suggested to have 1 logical camera for bin[1,2,3,4] and another logical camera for bin[5,6,7,8]. Use rqt to make sure each camera sees 16 parts (16 frames should be created).
   - Start `> rqt` in the terminal. Then in the menu select **Plugins→Visualization→TF Tree**.
   - You may need to clear the buffer after you move a camera (top left icon in rqt) so it can properly refresh the TF tree.

   When you are done, store the pose of these 4 cameras in [group1_config.yaml]. From here Ctrl + c and restart the simulation to ensure your cameras are properly positioned.
6. Place a logical camera above each AGV. Those cameras are needed to check that the parts you have placed in a tray match the parts described in a shipment. Also ensure the cameras placed above the AGVs do not overlap with the cameras placed above the bins. Again, use rqt to check that each camera above the AGVs sees only 1 part. When you are done, store the pose of each camera in [group1_config.yaml]. Ctrl + c and restart the simulation to ensure your cameras are properly positioned.
7. Place a camera above each assembly station so each camera can report 2 parts (located in the briefcase). Record the pose of these cameras in [group1_config.yaml].

Ctrl + c and restart the simulation to ensure your cameras are properly positioned.

8. Edit [rwa2_trial.yaml] so that when the simulation starts, agv1 and agv2 are at as1, and agv3 and agv4 are at as3 (see Figure 2).
9. Place a camera above each AGV. You need these cameras to find parts on AGVs for assembly. After you ensure each camera sees a part on the AGV, store the pose of each camera in [group1_config.yaml]. Ctrl + c and restart the simulation to ensure your cameras are properly positioned.
10. Edit [rwa2_trial.yaml] one more time so when the simulation starts agv1 and agv2 are at as2, and agv3 and agv4 are at as4 (see Figure 3). Repeat the same previous process: Place a camera above each AGV, ensure each camera sees the part on the AGV, record the pose of the camera in [group1_config.yaml], and finally Ctrl + c.
11. Edit [rwa2_trial.yaml] one last time.
    - Start the AGVs at their kitting station (ks1, ks2, ks3, and ks4).
    - Make all the bins empty.
    - Do the needed steps so that when the competition starts, 15 red pumps will spawn on the conveyor belt at an interval of 10 simulation seconds.
    - Start the simulation and then start the competition (through the service call).
    - Place a logical camera above the belt so it can see parts on the belt. Record the pose of this camera in [group1_config.yaml].
    - Place a breakbeam sensor by the belt so it can detect when an object breaks the beam. Record the pose of this sensor in [group1_config.yaml]. Note: The user config file provided in class already has a breakbeam sensor which seems to be well positioned. You can copy-paste the pose of this sensor in [group1_config.yaml].
12. Make sure your environment has 18 logical cameras:
    - 2 for the bins.
    - 4 for the AGVs at ks1, ks2, ks3, and ks4.
    - 4 for the AGVs at as1 and as3.
    - 4 for the AGVs at as2 and as4.
    - 4 for the briefcases.

6.2 Agility Challenges

For this task you will need to modify [rwa2_trial.yaml] to trigger agility challenges. You will then modify the source code of your package to detect these agility challenges. Agility challenges will be triggered with the execution of [part_spawner.sh].

- It was mentioned in class to wait 20 simulation seconds to see if parts will show up on the belt. You can get the simulation time in your program by subscribing

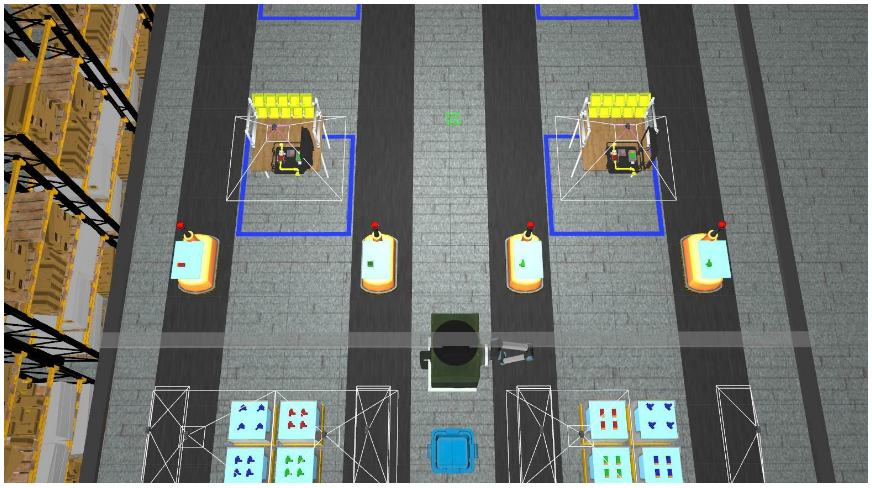


Figure 2: agv1 and agv2 at as1. agv3 and agv4 at as3.

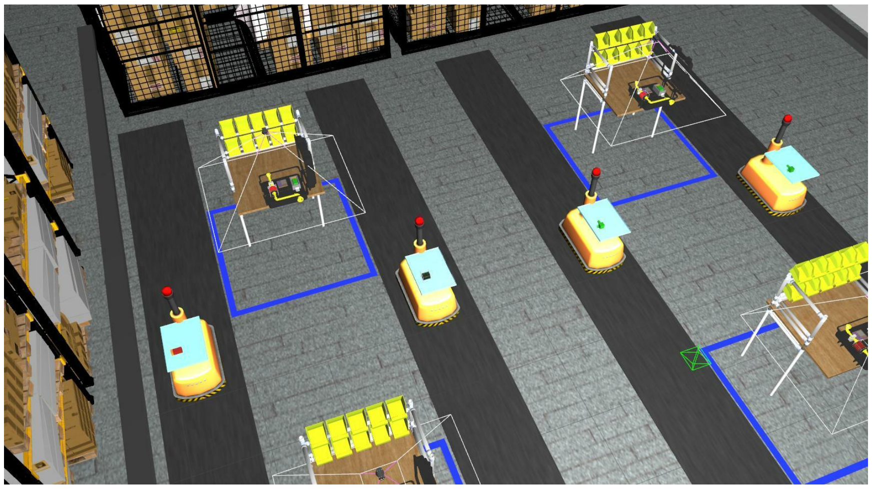
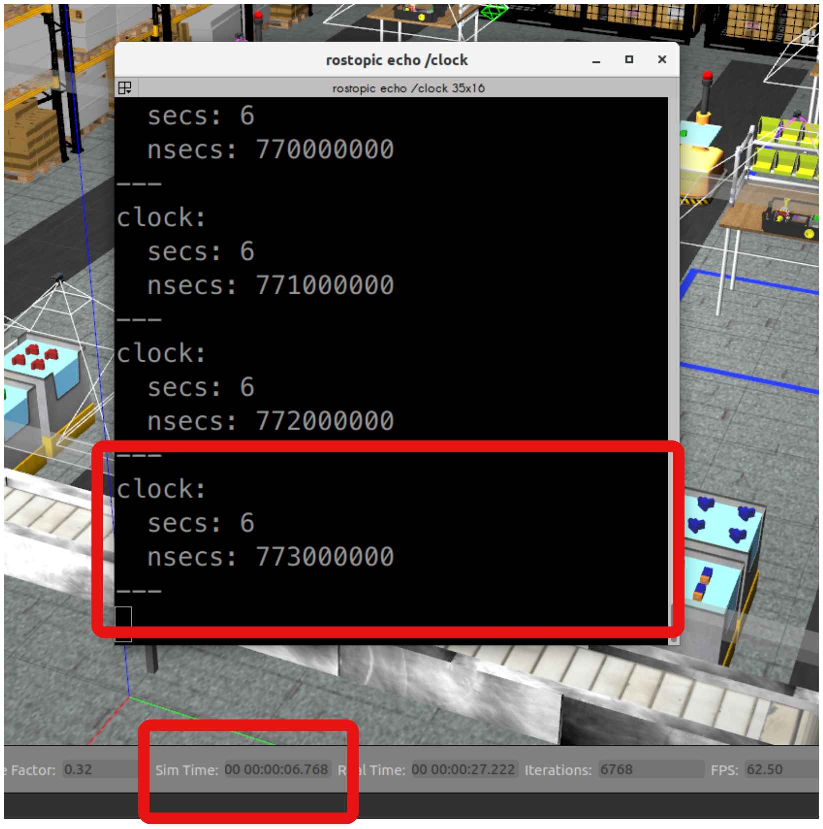


Figure 3: agv1 and agv2 at as2. agv3 and agv4 at as4.



```
  secs: 6
  nsecs: 770000000
---
clock:
  secs: 6
  nsecs: 771000000
---
clock:
  secs: 6
  nsecs: 772000000
---
clock:
  secs: 6
  nsecs: 773000000
---
```

```
Factor: 0.32   Sim Time: 00 00:00:06.768   R  l Time: 00 00:00:27.222   Iterations: 6768   FPS: 62.50
```

Figure 4: Time on the Topic /clock matches the Simulation Time in Gazebo.

to the Topic /clock. Gazebo publishes the simulation time on the Topic /clock and the time published on this Topic is synchronized with the simulation time in Gazebo (see Figure 4).

6.2.1 Sequence and Class Diagrams

Modify the sequence diagrams (and maybe the class diagram) created in RWA-1 to address agility challenges discussed in class (Lecture 6). Remember that all the challenges apply to kitting while only some challenges apply to assembly. Once you are done with your diagrams update the report you submitted with RWA-1.

6.2.2 Challenges

This section describes what you need to add to `[rwa2_trial.yaml]` to start agility challenges when some events occur in the environment.
Modify `[rwa2_trial.yaml]` to have 4 red pumps in bin1 and 4 blue batteries in bin7. No parts on any AGV, in any briefcase, and on the belt.

- In-process Order Change: Create two orders (order_0 and order_1) with one kitting shipment for each order.
  - The kitting shipment for order_0 must be built on agv1 and submitted to as2. This shipment consists of 2 red pumps.
  - The kitting shipment for order_1 must be built on agv3 and submitted to as3. This shipment consists of 1 blue battery and 1 green sensor. This order is a high-priority order.
  - Write the announcement condition and the announcement value to trigger order_1 when the first red pump is placed on agv1.
- Sensor Blackout: A sensor blackout should be triggered when the number of parts on any AGV is 2. Make the sensor blackout last 10 simulation seconds.
- Faulty Part: Make assembly_battery_blue_5 a faulty part.
- Flipped Part: Mark one of the two red pumps in the shipment of order_0 as a part that needs to be flipped.
- Insufficient Product: This challenge is already taken care of since order_1 needs a green sensor and there is no green sensor in the workcell.

7 Program Execution

[part_spawner.sh] is provided to you and it will spawn parts at different intervals.

```
spawn assembly_pump_red_5 on agv1
sleep 10
spawn assembly_battery_blue_5 on agv3
```

- Start the simulation: `> roslaunch group1_rwa2 ariac.launch`
- Start your program which will start the competition (through the service client).
- Your Node reads the order published on /ariac/orders and looks for parts needed in this order. Your Node should display in the terminal (only once) the pose of parts found by your camera(s) that are needed in the shipment of order_0. In other words, your Node should display the pose (in the /world frame) of 4 red pumps (these parts are in bin1).
- Execute [part_spawner.sh] in a different terminal:
  - assembly_pump_red_5 will spawn on agv1, this should announce order_1. Have your Node display the message "High-priority order is announced" in the terminal.
  - Read order_1 and check if your cameras can find parts needed for this order. Display in the terminal the pose of parts found by your camera(s) that are needed in the shipment in this order. In this case, your Node should only print out the pose (in the /world frame) of 4 blue batteries (located in bin7). order_1 also needs 1 green sensor but there is no green sensor in the bins.

If after 20 s (/clock Topic) your cameras did not detect any green sensor part then have your Node print the message "insufficient parts to complete order_1" in the terminal.

– [part_spawner.sh] will then spawn assembly_battery_blue_5 on agv3. This should trigger a sensor blackout. Have your Node display the message "Sensor blackout" in the terminal. You will need to check that your cameras are not reporting anything to deduce that there is a sensor blackout. The blue battery spawned is a faulty part (you marked it as a faulty part in [rwa2_trial.yaml]). Your Node should print out "Faulty part detected on agv3" in the terminal. Remember this part will be reported as faulty by the quality control sensor above agv3.

8 Deliverables

Your group's project submission consists of two deliverables:

10 pts A report (5 pages maximum) describing the modifications you made to your diagrams from the previous assignment.

20 pts Evaluation of your package. For group 1, the zipped package would be [group1_rwa2.zip]. We will modify [rwa2_trial.yaml] and [part_spawner.sh] to see if your program still gives us the expected outputs.