Spring'22

Zeid Kootbally
Craig Schlenoff
University of Maryland
College Park, MD

RWA-1 (v0.1.2)

ENPM663: Building a Manufacturing Robotic Software System

Due date: **Wednesday, March 02, 2022, 4 pm**
2 weeks

Contents

1 Updates

This section describes updates added to this document.

- v0.1.2: Added information on how to call Services.
- v0.1.1: Modified the objectives to allow submissions in C++, Python, or both.
- v0.1.0: Rewrote the document for clarity.
- v0.0.1: First draft.

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.
   - Once you have ARIAC installed, make sure you can run it with:

   -
     ```
     > roslaunch nist_gear sample_environment.launch
     ```
   - Ctrl + c afterwards.
2. General knowledge on ARIAC. This was one of the requirements that was posted on Canvas when the course was published.
3. You have attended the two ROS lectures and the Architectures lecture in this course.
4. You have completed the exercise from the second ROS lecture.
5. You must be part of a group to complete the assignment.

4 Setup

Two files are provided for this assignment. `[rwa1_trial.yaml]` consists of the trial used for this assignment. This file includes information on orders, parts to be spawned, and challenges for the trial. `[rwa1_sensor.yaml]` consists of sensor types and their locations in the environment. For now, you do not need to understand the content of these files. You need to place these files in some specific folders.

- Create the folder `rwa` in `nist_gear/config/trial_config` and place the file [rwa1_trial.yaml] in this folder.
- Create the folder `rwa` in `nist_gear/config/user_config` and place the file [rwa1_sensor.yaml] in this folder.
- Modify lines 8 and 9 as shown below in [sample_environment.launch], which is located in `nist_gear/launch`:

```
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 nist_gear)/config/trial_config/rwa/rwa1_trial.yaml
9         $(find nist_gear)/config/user_config/rwa/rwa1_sensor.yaml
10        " required="true" output="screen" />
```

- Run `> roslaunch nist_gear sample_environment.launch` to make sure the Gazebo simulation environment starts. Ctrl + c afterwards.

5 Objectives

Using ROS lectures and the Architectures lecture given so far in this course, your objectives for this assignment consist of creating a catkin competitor package to get you ready for future assignments and the final project. In other words, you will need to build the skeleton for your competitor package. The package can be written in C++, Python, or both but we need only one package per group.

Note: For this assignment, you can copy/paste some of the code found in the folder `~/ariac_ws/src/ARIAC/ariac_example/src`. Also note that the code found in this folder is not up-to-date with ARIAC 2021, but you can still adapt it.

6 Tasks

The following tasks must be performed for this assignment.

6.1 Catkin Package

Create a catkin package in `~/ariac_ws/src`. If you followed the installation instructions, this workspace already contains two packages: ARIAC and ariac-gazebo_ros_pkgs. Create a third package <groupNumber>_rwa1 (e.g., group1_rwa1) with the following dependencies:

- nist_gear to be able to use Services and Messages from this package.
- roscpp (or rospy or both)
- sensor_msgs to be able to subscribe to sensor and camera Topics.
- std_srvs to be able to call Services provided by ROS.
- trajectory_msgs to be able to control the arms. This package not be used for this assignment but it will be needed for future assignments.

In your package, create a `doc` folder and create a file `[instructions.txt]` in this folder. This file will provide instructions on how to run your Node(s) for this assignment. The `doc` folder will also be used to place the report due for this assignment.

6.2 Architecture and Design

You are about to start a project to which you will add new functionalities through future assignments. Before writing any software, you have to clearly describe the system on paper and then implement it using programming. UML, which stands for Unified Modeling Language, is a way to visually represent the architecture, design, and implementation of complex software systems. The second step in this assignment is to use UML to describe the architecture for your competitor software. As a UMD student, you you have free access to https://www.lucidchart.com to create UML diagrams.

- Class diagram: Create one class diagram to represent the components you think are needed in ARIAC. You should have read about ARIAC on the wiki and by now you should have a good idea about these components. For instance, some components of ARIAC are AGVs, robots, bins, parts, cameras, sensors, and assembly stations. Note that this is not an exhaustive list.
- Sequence diagrams: Create one sequence diagram for kitting and one sequence diagram for assembly. You may forget some steps in these sequence diagrams but try your best to be as accurate as possible. Information on what happens in kitting and assembly can be found on the wiki.

6.3 Software Implementation

Once you have a class diagram, translate it to C++ or Python code to build the structure of your software. Translating a class diagram into a program consists of creating object-oriented classes. Follow the sub sections below to understand what is needed in your implementation. All the Topics and Services described in these sub-section can be found on the wiki wiki.

6.3.1 Programming Style Guide

You will be working as a team and it is important everyone follows the same programming guideline and conventions, otherwise you will end up with a mess with everyone writing code in his own style. All programs written in Python have to follow the PEP 8 Style Guide. All programs written in C++ have to follow this C++ Core Guidelines.

WARNING! Groups who do not follow these guidelines will incur a 5 pts penalty.

6.3.2 Process Order Information

Subscribe to the Topic /ariac/orders and process orders published to this Topic. Processing an order consists of using a callback function to parse Messages published to this Topic and to store order Messages in your program. The trial used in this assignment consists of only one order with two tasks: kitting and assembly. You need to create appropriate data structures to store each task from this order. Depending on how you designed your class diagram you may have an Order class or maybe one Kitting class and one Assembly class.

6.3.3 Process Camera/Sensor Information

Using the ARIAC documentation on sensors and cameras, Create a Subscriber for each camera and for each sensor located in the environment.
These cameras and sensors are described in `[rwa1_sensor.yaml]`. Topic names for cameras and sensors are dynamically built from cameras and sensors' names. For instance, the break beam sensor in `[rwa1_sensor.yaml]` is named **breakbeam_0** and the Topic publishing data for this sensor is /ariac/breakbeam_0. Besides Subscribers for cameras and sensors described in `[rwa1_sensor.yaml]`, you need 4 more Subscribers for the quality control sensors located above AGVs. Topic names for these quality control sensors are /ariac/quality_control_sensor_{N}, where N is in the range [1,4].
For this assignment, you can hard code Topic names in your Subscribers. For now, all callback functions consist of a string message that you will print on the screen only when Messages are published to Topics. You need to use the documentation to find out what sensor/camera Topics look like when Messages are published Vs. when Messages are not published. Here is an example of printing a string message on the screen when the break beam sensor is triggered:

- `rospy.loginfo("Callback triggered for Topic /ariac/breakbeam_0")` (Python).
- `ROS_INFO_STREAM("Callback triggered for Topic /ariac/breakbeam_0")` (C++).

Note: Create a Subscriber/callback function for the depth camera but do not process any incoming Messages and do not output anything on the screen. The body of the callback should be empty (you will get a warning during compilation but this is fine).

Note: The break beam sensor publishes on 2 Topics. You need to subscribe to both Topics.
Note: You do not store sensor/camera Messages in your program in this assignment, this will be your task in the next assignment. You only need to print a message on the screen when Messages are published to sensor/camera Topics.

6.3.4 Handling Automated Guided Vehicles (AGVs)

Competitors do not perform path planning for AGVs but they need to submit AGVs to the correct assembly stations once kitting is done. In other words, to compute the score for kitting you need to build the kit on the correct AGV and submit the AGV to correct assembly station.
To-do: Create Subscribers for the AGV Topics below and appropriately store Message information in your program (you should have an AGV OOP class).

- /ariac/agv{N}/state. N is in the range [1,4], so you need 4 Subscribers/callback functions.
- /ariac/agv{N}/station. N is in the range [1,4], so you need 4 Subscribers/callback functions.

To-do: Create ROS Service clients to programmatically submit AGVs using the Service below. We did not see ROS Services in class but Service clients are easy to implement. Here is the tutorial for C++ and here is the tutorial for Python.

- /ariac/agv{N}/submit_shipment (you need 4 Service clients since we have 4 AGVs).

6.3.5 Starting and Ending the Competition

You need to programmatically start and end the competition through Service calls. Specific Topics provide information on the state of the competition.
To-do: Create a Subscriber for the following Topics and store information published to this Topic in your program:

- /ariac/competition_state

To-do: Create Service clients to start and end the competition for the following Services:

- /ariac/start_competition
- /ariac/end_competition

6.3.6 Submit Assembly

To submit assembly for scoring you need to call a specific Service.
To-do: Create 4 Service clients for the following Services:

- /ariac/as{N}/submit_shipment

6.3.7 Main Function

Later in the course you will do pick-and-place. This assignment focuses on the structure of your package, on the correct implementation of Subscribers, and on calling the correct Service clients. When the ARIAC environment starts, you will see parts located on an AGV and parts also already assembled at an assembly station.
To-do: Write the main function for your Node(s) which will do the following steps:

- Check the state of the competition (Topic /ariac/competition_state) and start the competition (Service /ariac/start_competition) if it has not started yet. Note: When the competition is not started, the state of the competition published on /ariac/competition_state is "init". Once you start the competition, its state will change to "go". You should update your program to reflect the change in the competition state. Once the competition starts you will see parts moving on the conveyor belt. We should also see more outputs in the terminal since each sensor/camera callback function outputs a string message on the screen. You should also see a message from the quality control sensor above AGV1 since there is a faulty part on this AGV.
- Wait about 1 minute in your program and submit the AGV to the assembly station (Service /ariac/agv{N}/submit_shipment). You need to replace N with an AGV number. Information on which AGV to submit and where to submit this AGV is provided on the Topic /ariac/orders for the kitting task. The shipment type needed to call this service should also be retrieved from the Topic /ariac/orders. Next, submit assembly (Service /ariac/as{N}/submit_shipment). Again, which assembly station and which assembly shipment to submit should be retrieved from /ariac/orders for the assembly task.
- End the competition (Service /ariac/end_competition). Make sure the state of the competition is not "done" before you end the competition.
- Check the competition has ended (the competition state is "done") and shut down your Node(s). In class we saw that Ctrl + c shuts down a Node. For this assignment, you will need to shut down your Node(s) programmatically with `ros::shutdown()` in C++ or with `rospy.on_shutdown()` in Python.

7 Program Execution

To test your program, you will perform the following:

- Start ARIAC: `> roslaunch nist_gear sample_environment.launch`
- Start your Node(s), which will execute the main function described in Section 6.3.7. If your package consists of multiple Nodes and you need to start all of them at

once, then use a launch file (create a `launch` folder in your package). We saw launch files in our first ROS lecture with the turtlebot.

- Note: We will use a different trial file to grade this assignment. To make sure you did not hard code data that should be retrieved dynamically we will change the AGV and the assembly station used in the kitting order. We will also change the assembly station used in the assembly order. Everything else stays the same.

8 Deliverables

Your group's project submission consists of two deliverables:

10 pts A report (10 pages maximum) describing the architecture and the design you chose to use. Do not just dump figures in the report without any explanation. Describe your choice for the architecture and your choice for classes. Describe the different steps of your sequence diagrams. Place this report in the `doc` folder in your package.

20 pts ROS package with Node(s), launch files, and other contents that you deem necessary to run your Node(s).

- You must use OOP.
- Make sure your code follows the conventions and guidelines.
- Get rid of unused code. Do not leave big chunks of commented code in your program.
- Comment and document your code. Use doxygen for C++ and docstring for Python. See this VSC extension for doxygen and this VSC extension for docstring.
- As mentioned earlier, create `[instructions.txt]` in `doc` and write the instructions in this file on how to run your program.
- Compress/zip your package and upload it on Canvas. Only one zipped file per group.

9 Getting Help

You are given 2 weeks for this assignment. Although this is the first assignment, it requires an important amount of work. Start working on this assignment as soon as possible. If some parts of ARIAC are unclear, contact me via email. We can have zoom meetings if lengthty explanations are needed.