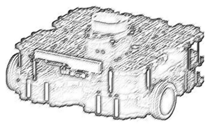

Zeid Kootbally
Spring 2023
UMD
College Park, MD

Final Project (v0.0)

ENPM809E: Python Applications for Robotics

Due date: **Wednesday, May 10, 2023, 4 pm**

Contents

1 Updates

This section describes updates added to this document since its first released. 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 an important note
- This is a warning
- This is a ›link
- This is a set of tasks to do
- This is a n node
- This is a t topic
- This is a f frame
- This is a s service
- This is a p parameter
- This is a m message

3 Prerequisites

3.1 Final Project Stack

- Create a clean workspace and clone the packages needed for the final project:
  - `> cd "My␣Documents"` changes to directory `My Documents`.
  - `cd`
  - `mkdir -p groupX_ws/src`
  - `cd groupX_ws/src`
  - `git clone https://github.com/zeidk/enpm809e_final_spring2023.git -b final`
  - Move the package you created in RWA4 into the src folder of this workspace.
  - `cd ~/groupX_ws`
  - `rosdep install -from-paths src -ignore-src -r`
  - `colcon build`

– `source install/setup.bash`

4 Introduction

This project consists of performing kitting with the use of an industrial robot. This project is split in two parts:

1. Receive orders and locate parts and the tray need for the order. In RWA4, only one Order was published to t /ariac/orders. In this project, four orders are published to this Topic but only one of these four Orders must be completed. You need to modify the package you created for RWA4 to be able to store these four orders.
2. Read a Parameter to know which one of the four Orders to build and send commands to the robot to build the kit.

5 Assignment

This section describes the different tasks that your Nodes have to complete. You can perform everything with one single Node. If you think it is easier to write multiple Nodes then you are free to do so.

The following commands are expected to be used to run your Nodes:

- *terminal 1*: `ros2 launch ariac_gazebo ariac.launch.py trial_name:=final` – This starts the simulation environment. Add the option `start_rviz:=true` if you want to start rviz.
- *terminal 2*: `ros2 launch robot_commander robot_commander.launch.py` – This starts MoveIt for motion planning and the package which sends commands to MoveIt.
- *terminal 3*: `ros2 run robot_commander floor_robot_main.py` – This starts a minimal Node which sends commands to the robot. You will need to replace this command with a different command to run your Nodes.

5.1 Storing Orders

Four Orders are announced at the same time, each Order consists of one single kitting task, and each kitting task requires two parts. These Orders are defined in ariac_gazebo/config/trials/final.yaml

1. Improve the package you created in RWA4 to be able to store all four Orders in your program. Later you will need to parse these Orders to figure out which one the robot has to build.
2. This task should also locate parts and the tray needed for each Order. This should have already been performed in RWA4.
3. If you are starting your Nodes in multiple terminals, group them in one single launch file. This launch file will be important for the next phase of this project.

5.2 Complete the Order

The second and final phase of the project is to task the robot to complete the kit. ARIAC consists of two robots. The floor robot is the robot mounted on the linear rail and 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. For this project, we will only use the floor robot.

Before completing a kit your program needs to figure out which Order out of the four Orders to complete. For this, create a parameter file order.yaml in your package and pass it to your Node from the launch file you created in the previous phase. In class we saw how to pass a parameter file to a Node within a launch file. The content of the parameter file is depicted in Listing 5.1. Make sure to modify line 1 and use the name of your Node instead.

Listing 5.1: Parameter file.

```yaml
1 my_node:
2   ros__parameters:
3     order_id: "2"
```

You need to read the parameter p order_id in your program to know which Order to build. In this example, the order with id "2" is expected to be built. During testing, change the Order id to make sure your program builds the kit for the correct Order.

5.2.1 Python Package for Moving the Robot

The package robot_commander consists of a C++ and a Python package. The communication between the Python package and the C++ package is done with the use of ROS Services.

- The C++ package hosts multiple ROS Service Servers which are just waiting to receive commands from Service clients. Requests received by these servers will execute robot actions. One request performs one robot action.
- The Python package provides a minimal example of how Service clients can send requests to the Service Servers.

The Python class RobotCommanderInterface located in the package robot_commander is a good starting point. Modify your Nodes to replicate the content of this file but do not modify this file. A brief description of this file is provided below.

- The subscriber to t /ariac/competition_state is used to get the state of the competition. The competition goes into different states during its lifetime and its state changes based on some events. The subscriber only stores the state of the competition in the attribute _competition_state.
- The Service client _start_competition_client is used to start the competition. In RWA4, the competition was started automatically for you. In this project we start the competition ourselves. Once the competition is started, Orders are announced, cameras publish data, and robots can be controlled. None of these are possible if the competition is not started. The competition

will be started automatically for you when you run this Node.

- The Service client _move_floor_robot_home_client is used to move the robot to its home location. By default, robots are spawned in the workcell with a default configuration. By moving the robot to its home configuration you can make sure the robot is controllable (among other things).
- The Service client _goto_tool_changer_client is used to move the robot to the tool changer. The tool changer allows you to change the gripper type of the robot. When the simulation environment starts, the robot is equipped with a part gripper, which can only pick up parts. To pick up trays, the gripper type needs to be changed at the tool changing station.
- A timer is used to execute Service clients sequentially. To prevent deadlocks, we use two threads, one of the timer callback and one for the Service clients. _robot_action_timer_callback is the timer callback where Service clients are executed sequentially. Instead of executing Service requests directly, we call class methods which will execute the Service requests. Placing the Service requests in methods provide clean and reusable code.

5.2.2 Services for Kitting

This Section describes all Services that are available to perform kitting. A reminder that to see the service definition you can use `ros2 interfaces show`

1. S /ariac/start_competition
   - Definition: **std_srvs/srv/Trigger**
   - Description: Start the competition.
2. S /competitor/floor_robot/go_home
   - Definition: **std_srvs/srv/Trigger**
   - Description: Move the floor robot to its home configuration.
3. S /competitor/floor_robot/enter_tool_changer
   - Definition: **competitor_interfaces/srv/EnterToolChanger**
   - Description: Move the end effector of the robot in one of the two tool changers at one of the two stations.
4. S /competitor/floor_robot/exit_tool_changer
   - Definition: **competitor_interfaces/srv/ExitToolChanger**
   - Description: Move the end effector out of the tool changer at the specified station.
5. S /competitor/floor_robot/pickup_tray
   - Definition: **competitor_interfaces/srv/PickupTray**
   - Description: Pick up a tray from the tray station.
6. S /competitor/floor_robot/move_tray_to_agv
   - Definition: **competitor_interfaces/srv/MoveTrayToAGV**
   - Description: Move the tray above the AGV without placing it.
7. S /competitor/floor_robot/place_tray_on_agv
   - Definition: **competitor_interfaces/srv/PlaceTrayOnAGV**
   - Description: Place the tray on the AGV.

8. S /competitor/floor_robot/retract_from_agv
   - Definition: **competitor_interfaces/srv/RetractFromAGV**
   - Description: Retract from the AGV after placing an object on the AGV.
9. S /competitor/floor_robot/pickup_part
   - Definition: **competitor_interfaces/srv/PickupPart**
   - Description: Pick up a part from the bin.
10. S /competitor/floor_robot/move_part_to_agv
    - Definition: **competitor_interfaces/srv/MovePartToAGV**
    - Description: Move a part above the AGV without placing the part.
11. S /competitor/floor_robot/place_part_in_tray
    - Definition: **competitor_interfaces/srv/PlacePartInTray**
    - Description: Place part in a specific quadrant in the tray.
12. S /ariac/agvX_lock_tray Replace X with an AGV number (1, 2, 3, or 4).
    - Definition: **std_srvs/srv/Trigger**
    - Description: Lock the tray and parts on the AGV so they do not fall when the AGV is in motion.
13. S /ariac/move_agvX Replace X with an AGV number (1, 2, 3, or 4).
    - Definition: **ariac_msgs/srv/MoveAGV**
    - Description: Move the AGV to a destination.
14. S /ariac/floor_robot_change_gripper
    - Definition: **ariac_msgs/srv/ChangeGripper**
    - Description: Change the type of the gripper.
15. S /ariac/floor_robot_enable_gripper
    - Definition: **ariac_msgs/srv/VacuumGripperControl**
    - Description: Turns the gripper on/off.

5.2.3 Example

Listing 5.2 provides an example of method calls in the timer callback to complete a kit. Many arguments are hardcoded as variables as this is an example. Obviously, your program will need to retrieve these arguments dynamically from your data structures. The most important part of this example is the order in which each method is called. It is expected that your program calls these methods in the same sequence. If these methods are called in the wrong order, the robot will not be able to build the kit. Also note that the services S /ariac/floor_robot_change_gripper and S /ariac/floor_robot_enable_gripper are not visible in Listing 5.2. These Services are called within some of the methods. Listing 5.3 shows how the method place_part_in_tray**()** is implemented (this one is free of charge). Line 32 shows a method call which will deactivate the gripper so the part drops in the tray.

Listing 5.2: Example of method calls.

```python
def _robot_action_timer_callback(self):
    '''
    Callback for the timer that triggers the robot actions
    '''

    if self._kit_completed:
        return

    # Pose of tray in the world frame
    quaternion = quaternion_from_euler(0, 0, 3.141591)
    tray_pose = Pose()
    tray_pose.position.x = -0.870000
    tray_pose.position.y = -5.840000
    tray_pose.position.z = 0.734990
    tray_pose.orientation.x = quaternion[0]
    tray_pose.orientation.y = quaternion[1]
    tray_pose.orientation.z = quaternion[2]
    tray_pose.orientation.w = quaternion[3]

    # Pose of a purple pump in the world frame
    quaternion = quaternion_from_euler(0, 0, 3.141591)
    purple_pump_pose = Pose()
    purple_pump_pose.position.x = -2.079999
    purple_pump_pose.position.y = 2.805001
    purple_pump_pose.position.z = 0.723487
    purple_pump_pose.orientation.x = quaternion[0]
    purple_pump_pose.orientation.y = quaternion[1]
    purple_pump_pose.orientation.z = quaternion[2]
    purple_pump_pose.orientation.w = quaternion[3]

    pump_type = Part.PUMP
    pump_color = Part.PURPLE
    pump_quadrant = 2
    pump_bin = "right_bins"

    # Pose of a blue battery in the world frame
    quaternion = quaternion_from_euler(0, 0, 3.141591)
    blue_battery_pose = Pose()
    blue_battery_pose.position.x = -2.080001
    blue_battery_pose.position.y = -2.445000
    blue_battery_pose.position.z = 0.723434
    blue_battery_pose.orientation.x = quaternion[0]
    blue_battery_pose.orientation.y = quaternion[1]
    blue_battery_pose.orientation.z = quaternion[2]
    blue_battery_pose.orientation.w = quaternion[3]

    battery_type = Part.BATTERY
    battery_color = Part.BLUE
    battery_quadrant = 4
    battery_bin = "left_bins"

    # ID of the tray
    tray_id = 3

    # table where the tray is located
    tray_table = "kts1"

    # AGV
    agv = "agv1"
```

```
 60
 61         # move robot home
 62         self.move_robot_home("floor_robot")
 63
 64         # change gripper type
 65         self.goto_tool_changer("floor_robot", tray_table, "trays")
 66         self.retract_from_tool_changer("floor_robot", tray_table, "trays")
 67
 68         # pick and place tray
 69         self.pickup_tray("floor_robot", tray_id, tray_pose, tray_table)
 70         self.move_tray_to_agv("floor_robot", tray_pose, agv)
 71         self.place_tray("floor_robot", tray_id, agv)
 72         self.retract_from_agv("floor_robot", agv)
 73
 74         # change gripper to pick up parts
 75         self.goto_tool_changer("floor_robot", "kts2", "parts")
 76         self.retract_from_tool_changer("floor_robot", "kts2", "parts")
 77
 78         # pick and place purple pump
 79         self.pickup_part("floor_robot", purple_pump_pose, pump_type, pump_color, "right_bins")
 80         self.move_part_to_agv("floor_robot", purple_pump_pose, agv, pump_quadrant)
 81         self.place_part_in_tray("floor_robot", agv, pump_quadrant)
 82         self.retract_from_agv("floor_robot", agv)
 83
 84         # move robot home
 85         self.move_robot_home("floor_robot")
 86
 87         # pick and place blue battery
 88         self.pickup_part("floor_robot", blue_battery_pose, battery_type, battery_color, battery_bin)
 89         self.move_part_to_agv("floor_robot", blue_battery_pose, agv, battery_quadrant)
 90         self.place_part_in_tray("floor_robot", agv, battery_quadrant)
 91         self.retract_from_agv("floor_robot", agv)
 92
 93         # move robot home
 94         self.move_robot_home("floor_robot")
 95
 96         # move agv to warehouse
 97         self.lock_agv(agv)
 98         self.move_agv_to_warehouse(agv)
 99
100         self._kit_completed = True
```

Explanation:

- Lines 6–7 ensure that the remaining code of this callback is skipped if the kit has been completed.
- Lines 9–59 provide hardcoded variables. You have to reuse the program you wrote for RWA4 to retrieve the value of these variables.
  - Lines 31–32 & 47–48: This example uses m /ariac_msgs/msg/Part.msg to provide the type and the color of the part. In your case, you will get this information from the published Order.
  - Lines 33 & 49: The quadrant where the part is placed in the tray. You should retrieve this information from the published Order.
  - Lines 34 & 50: The Service **competitor_interfaces/srv/PickupPart** has the field **bin_side** which can take one the two values "right_bins" or "left_bins". If the part is detected by the right_bins_camera then the value for this field is "right_bins", otherwise it

is "left_bins".
  - Line 53: Id of the tray needed for the Order. This should be retrieved from the published Order.
  - Line 56: Table where the tray needed for the Order is located. This is "kts1" if the tray was detected by kts1_camera and "kts2" if it was detected by kts2_camera.
  - Line 59: AGV where the kit needs to be built.
- Lines 65–66: Since the robot is changing the gripper to pick up the tray, it makes more sense to change the gripper at the same tool changing station where the tray is located. If you replace tray_table with "kts2", the program will still work but the robot goes to one side of the room to change gripper and then back to the other side to pick up the tray, which is not efficient.
- Line 98: This method calls the Service S /ariac/move_agvX, which has the definition **ariac_-msgs/srv/MoveAGV**. For this project, the value for the field **location** should always be 3 so that the AGV goes to the warehouse.

Listing 5.3: Method to place a part in a tray.

```python
def place_part_in_tray(self, robot, agv, quadrant):
        '''
        Places a part in the tray

        Args:
            robot (str): Robot to use
            agv (str): AGV where the tray is located
            quadrant (int): Quadrant oin the tray to place the part in
        '''
        request = PlacePartInTray.Request()

        if robot == "floor_robot":
            request.robot = Robots.FLOOR_ROBOT
        else:
            raise ValueError('Invalid robot name')

        request.agv = agv
        request.quadrant = quadrant

        future = self._place_part_in_tray_client.call_async(request)

        try:
            rclpy.spin_until_future_complete(self, future)
        except KeyboardInterrupt:
            self.get_logger().warn('Interrupted while waiting for the service. Exiting...')
            return

        if future.result() is not None:
            response = future.result()
            if response:
                self.get_logger().warn('Deactivate gripper')
                self.set_floor_robot_gripper_state(False)
        else:
            self.get_logger().warn(f'Service call failed {future.exception()}')
```

1. Write the remaining Service clients.
2. Write the remaining method definitions which call the Service clients.
3. Call the remaining methods in the timer callback for the Order which needs to be built.

5.3 Robustness

Besides the robustness presented in RWA4, this project consists of one more point to consider. In Listing 5.2, lines 78-85 are for picking and placing the first part and lines 87–94 are for the second part. What if you have an Order with three or four parts? Your program should be able to handle these Orders as well.

6 Grading Rubric

- We start by giving each team full points and then deduct points as follows:
  - -10pts for not providing docstring documentation.
  - -10pts for not using correct naming convention for attributes, classes, methods, etc.
  - -20pts for not properly encapsulating your attributes. A leading underscore is required. The use of the property function or decorator for accessors. The use of setter, e.g., @x.setter for mutators.
  - -10pts if the methods are not called in the correct order.
  - -10pts if the wrong arguments are passed to the methods. The pose of parts and trays must be converted to the world frame and passed to the methods.
  - -10pts for lack of robustness. If we create an order with 4 parts, will your program be able to complete the kit? Is your program capable of completing a kit with multiple parts of the same type?
  - -5pts if your submission does not contain instructions on how to run your package.
  - -5pts for not properly uploading your package. We only want your package and nothing else.