University of Tehran

College of Engineering

School of Electrical and Computer Engineering (ECE)

School of Mechanical Engineering (ME)

Mechatronics & Robotics

Project : Room-Service Core (RSC)

Teaching Assistants:
Elnaz Balazadeh

Deadline: 3 Khordad 1404, 23:59

Table Of Contents

Part 1: Project Overview 3

Part 2: Step-by-step instructions 5

Part 3: Grading 7

Part 1: Project Overview

Project goal — Why we are doing this

Very soon our lab will deploy a mobile robot that carries coffee, mail, and snacks inside a smart apartment. Before wheels touch the ground, we need the *backend software* that can:

1. accept delivery requests.
2. **queue**, **track**, and **cancel** them.
3. instruct a *mock* base to drive between rooms.
4. stream live status updates to the user.

You will build that backend using **pure ROS2 tools**. No Gazebo, no 3-D graphics—just a clean node–topic–service graph that runs on any laptop.

Learning outcomes

Skill you will gain	**ROS2 concept exercised**
Define and compile custom `.msg` / `.srv` files	Interface design
Write publishers & subscribers in `rclpy`	Topics
Offer costume services for cancellation	Services
Run and inspect a multi-node graph	Nodes & CLI
Debug with the `ros2` CLI (`topic`, `service`, `node`, . . .)	CLI tools

Target system at a glance

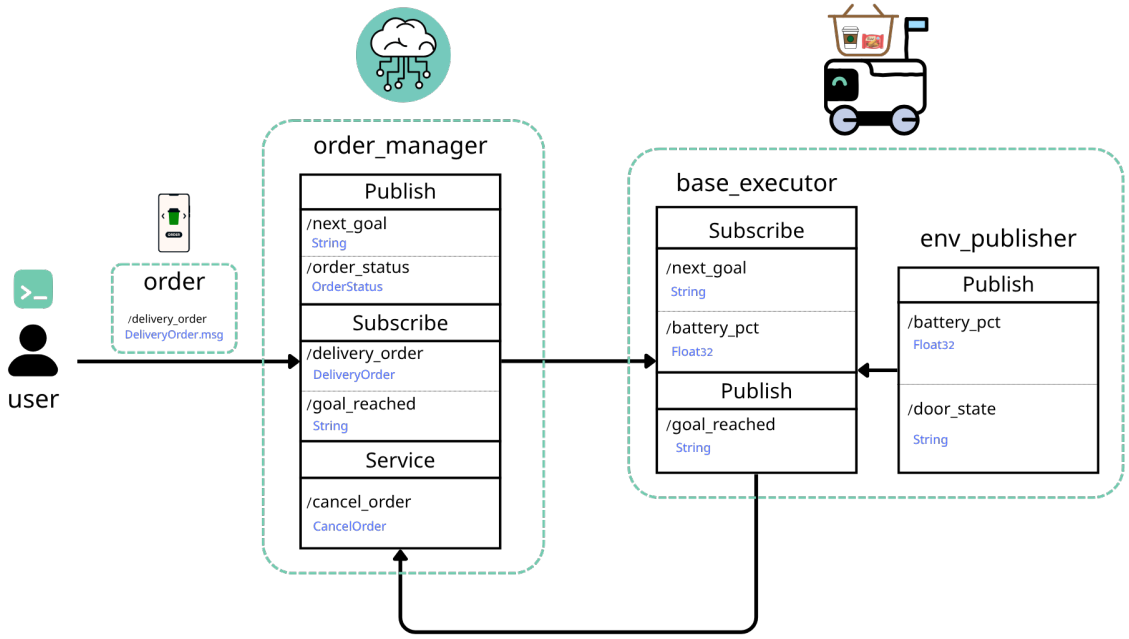

Figure 1: System overview of the Room-Service Core project.

File layout you must create

```
ros2_ws/src/
|- delivery_interfaces/
|   |- msg/
|   |   |- DeliveryOrder.msg
|   |   \- OrderStatus.msg
|   |- srv/
|       \- CancelOrder.srv
|
\- delivery_bringup/
    |- delivery_bringup/
    |   |- order_manager.py
    |   |- base_executor.py
    |   \- env_publisher.py
```

Part 2: Step-by-step instructions

Task 1 — Interface package

1. Create `delivery_interfaces` with `ament_cmake`.
2. Add the three files shown above.
3. Build and verify:

```
$colcon build
$source install/setup.bash
$ros2 interface show delivery_interfaces/msg/DeliveryOrder
```

Task 2 — order_manager

1. Subscribe to `/delivery_order`; store orders in a FIFO(First-in-First-out) queue.
2. Offer `/cancel_order` service; remove or mark orders.
3. Publish `/order_status` whenever a state changes.
4. When idle, publish `/next_goal` as `"A01,kitchen->study"` (String).

Task 3 — base_executor

1. Subscribe to `/next_goal`; parse the string.
2. Simulate pickup & drop-off with two `sleep(3)` calls.
3. Publish the order ID on `/goal_reached`.
4. Use `/battery_pct` for keeping robot alive and working. Navigate to charging station whenever the battery is under 10% and cancel all reaming orders.

Task 4 — close the loop

Make `order_manager` listen to `/goal_reached`; mark the order `DONE` and dispatch the next one.

I will test your work as follows

1. Build your workspace from scratch.
2. Start `order_manager` and `base_executor` in separate terminals:

```
$ros2 run delivery_bringup order_manager
$ros2 run delivery_bringup base_executor
```

3. From the CLI I will:
   - publish *several* orders,
   - cancel some of them before completion,
   - echo `/order_status` to verify state changes.
4. Code will be inspected for clarity and ROS2 best practice.

Part 3: Grading

Submission checklist

1. Compress both packages in a zip file.
2. A complete report as a documentation for your project.
3. 2–3 min demo video showing two orders, one cancellation, live status and battery percentage.

Grading rubric (100 pts)

Criterion	Points
Builds; nodes visible via `ros2 node list`	20
Custom interfaces compile & used correctly	20
Topics/services named & handled correctly	15
Live demo clarity (CLI only)	15
Order state-machine behaves as expected	20
Code quality & documentation	10
Extra credit: Any creative solution	+10

Starter CLI cheat-sheet

```
# Send an order
$ros2 topic pub /delivery_order delivery_interfaces/msg/DeliveryOrder \
  "{order_id: 'A01', item: 'coffee', pickup_room: 'kitchen', dropoff_room: 'study'}"

# Cancel it
$ros2 service call /cancel_order delivery_interfaces/srv/CancelOrder \
  "{order_id: 'A01'}"

# Watch state changes
$ros2 topic echo /order_status
```

Need help?

- Email me — fastest for general questions.
- ROS2 documentation — beginner tutorials on publishers, subscribers, and services.

Homework Guidelines and Instructions

- The deadlines are fixed and cannot be changed. If you need extra time, you can use your grace period (16 days) and upload your answers up to 5 days after the deadline.
- If you write your report of this homework in LaTeX, you will be rewarded 5
-
- The implementation must be in Python programming language and your codes must be executable and uploaded along with the report.
- This project is done by one person.
- If any similarity is observed in the work report or implementation codes, this will be considered as fraud for the parties.
- Using ready-made codes without mentioning the source and without changing them will constitute cheating and your practice score will be considered zero.
- If you do not follow the format of the work report, you will not be awarded the grade of the report.
- All pictures and tables used in the work report must have captions and numbers.
- A large part of your grade is related to the work report and problem solving process.
- Please upload the report, code files, videos and other required attachments in the following format in the system: `R2P_[Lastname]_[StudentNumber].zip`
  For example, the: `R2P_Balazadeh_12345678.zip`
- You can ask your questions or doubts either via Telegram group or sending an email directly to the teaching assistants of this project through the following e-mail with the subject **R2P_Q**. Stay in touch educationally:
  - ROS Project: `balazadeh.elnaz@gmail.com` (Elnaz Balazadeh)
- Be happy and healthy.