ENPM605: Python Applications for Robotics

RWA #3

v1.0

Lecturer: Z. Kootbally

School: University of Maryland

Semester/Year: Spring/2025

MARYLAND APPLIED GRADUATE ENGINEERING

Table of Contents

Changelog

- ***v1.0***: Original version.

Conventions

bad practice	
best practice	
code syntax	
example	
exercise	
file	
folder	
guideline	
note	
question	
task	
terminology	
warning	
web link	link
package	

ROS node	n /node
ROS topic	t /topic
ROS message	m /message
ROS parameter	p /parameter
ROS frame	f /frame

Guidelines

This assignment must be completed ***individually***. Ensure compliance with all specified guidelines, as failure to follow any part of these guidelines will lead to a grade of ***zero*** for the assignment.

- Do not reuse a package obtained from peers.
- Keep your work confidential and refrain from sharing it with peers.
  - If you use github, you have to make your repository private.
- While discussing assignment challenges is encouraged, refrain from exchanging solutions with peers.
- Do not use code generated by AI tools.
  - The potential risks outweigh the benefits.

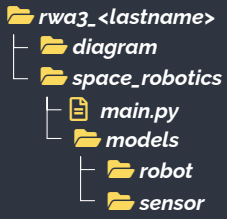
Objectives & Skills Practiced

The goal of this assignment is to apply Object-Oriented Programming (OOP) principles to design and implement a Python program that models space robots used for planetary exploration and space station maintenance. The assignment must include:

- ***Inheritance***: Define a base class and extend it for specialized space robots.
- ***Composition***: Use objects within other objects to model robot components.
- ***Aggregation***: Reference objects without ownership.
- ***Polymorphism***: Implement method overriding to enable different robotic behaviors.

Package

Each student uses one package with the following structure (package can be found on Github):



- ***main.py***: The entry point that imports and demonstrates polymorphism.
- ***diagram*** contains a UML class diagram (PDF format).
- ***robot*** contains the following classes: `SpaceRobot`, `PlanetaryRover`, `OrbitalRobot`, and `RoboticFleet`.
- ***sensor*** contains the following classes: `Sensor`, `LiDAR`, `Camera`, and `SensorSystem`.
- Ensure that each class is written in a separate file.
- Create additional files and folders as necessary.

Deliverable

Compress **rwa3_<lastname>** into ***a single file*** named **rwa3_<lastname>.<extension>**, where **<extension>** can be ***zip***, ***tar***, ***tar.gz***, etc., and **<last name>** is the student's last name. Submit it on Canvas by the due date.

Late submissions will incur a penalty according to the guidelines specified in the syllabus, with exceptions for any valid reason.

- Valid reasons include a doctor's note, proof of travel, or a note from a professor/MAGE.

Students with special circumstances may submit their assignments late without incurring any penalties. However, it is required that these students inform me in advance of their intention to submit their work past the deadline.

Scenario

Space agencies deploy robotic systems for tasks such as planetary exploration, satellite repair, and space station maintenance. These robots share common functionalities but differ in their capabilities. In this assignment, you will implement a Space Robotic Fleet using OOP principles in Python. Your application must emphasize aggregation and composition alongside inheritance, encapsulation, and polymorphism.

Overview

Your task is to design a Python program that models a fleet of space robots, incorporating:

- ***Encapsulation***: Protected attributes with controlled access.
- ***Aggregation***: A robotic fleet aggregates robots, which can exist independently of the robotic fleet.
- ***Composition***:
  - Each sensor system composes sensors, which cannot exist independently of the sensor system.
  - Each robot composes one sensor system, which cannot exist independently of the robot.
- ***Inheritance***:
  - A hierarchy of robot types.
  - A hierarchy of sensor types.
- ***Polymorphism***: Task execution varying by robot type.

`Sensor` Class (Abstract Base Class)

- Purpose: Defines the common structure and behavior for all sensor types.
- Attributes:
  - `_model` (`str`): The sensor's model name (e.g., `'X-500'`)
  - `_range` (`float`): The sensor's operational range in meters (e.g., `100.0`).
- Methods:
  - `get_info()`: An `@abstractmethod` that subclasses must override to return a string describing the sensor.
  - `operate()`: An `@abstractmethod` that subclasses must override to describe the sensor's specific operation (e.g., scanning, capturing).

LiDAR Class (Subclass of Sensor)

- Purpose: Represents a LiDAR sensor for distance measurement and mapping.
- Attributes:
  - Inherited: `_model` and `_range`.
  - `_angular_resolution` (`float`): Angular resolution in degrees (e.g., 0.5°), affecting scan precision.
- Methods:
  - `get_info()`: Returns a string with sensor information.
    - Example Output: `LiDAR (model: X-500, range: 100m, angular resolution: 0.5°)`
  - `operate()`: Returns a string describing the sensor operation.
    - Example Output: `X-500 (LiDAR): Scanning terrain with 0.5° resolution`

Camera Class (Subclass of Sensor)

- Purpose: Represents a camera sensor for imaging.
- Attributes:
  - Inherited: `_model` and `_range`.
  - `pixel_resolution` (`int`): Resolution in megapixels (e.g., 12 MP).
- Methods:
  - `get_info()`: Returns a string with sensor information.
    - Example Output: `Camera (model: CamPro, range: 50m, pixel resolution: 12MP))`
  - `operate()`: Returns a string describing the sensor operation.
    - Example Output: `CamPro (Camera): Capturing image at 12MP resolution`

`SensorSystem` Class

- Purpose: Manages a collection of `Sensor` subclass instances.
- Attributes:
  - `_sensors` (`list[Sensor]`): List of `Sensor` subclass instances.
  - `_status` (`str`): Current state (`"active"` or `"inactive"`).
    - This attribute is always set to `"inactive"` when a `SensorSystem` object is created.
- Methods:
  - `activate()`: Sets status to `"active"` and returns a string listing all sensors' `get_info()`.
  - `operate_sensors()`: Returns a string combining all sensors' `operate()` outputs.

`SpaceRobot` Class (Abstract Base Class)

- Purpose: Defines the common structure and behavior for all space robot types.
- Attributes:
  - `_name` (`str`): Identifier (e.g., `"MarsExplorer"`, `"OrbitFixer"`).
  - `_sensor_system` (`SensorSystem`): Composed instance managing `Sensor` subclass instances.
  - `_mobility` (`str`): Mobility mechanism (e.g., `"wheels"`, `"thrusters"`, `"tracks"`, , `"hoppers"`).
- Methods:
  - `activate()`: Calls `_sensor_system.activate()`.
  - `perform_task(task)`: An `@abstractmethod` that subclasses must override to return a string describing the task performed by the robot.

`PlanetaryRover` Class (Subclass of `SpaceRobot`)

- Purpose: Represents a robotic rover designed for planetary surface exploration.
- Attributes:
  - Inherited: `_name`, `_sensor_system`, and `_mobility`.
  - `_terrain_type` (`str`): Type of terrain the rover operates on (e.g., `"rocky"`, `"sandy"`, `"snowy"`).
- Methods:
  - `move()`: Returns the string: `Moving: Using <mobility> across <terrain_type> terrain`
    - Example Output: `Moving: Using wheels across rocky terrain`
  - `perform_task(task: str)`: Returns a string of the performed task. `PlanetaryRover` robots can perform only two tasks:
    - `"mapping"` displays the robot's model, calls `move()`, and calls `operate_sensors()` on the `_sensor_system` attribute. Example Output:

```
-Robot: MarsExplorer
    Moving: Using wheels across rocky terrain
    X-500 (LiDAR): Scanning terrain with 0.5° resolution
    CamPro (Camera): Capturing image at 12MP resolution
```

    - `"collection"` displays the robot's model and calls `move()`. Example Output:

```
-Robot: MarsExplorer
    Moving: Collecting sample on rocky terrain
```

`OrbitalRobot` Class (Subclass of `SpaceRobot`)

- Purpose: Represents a robotic system for servicing space stations and satellites in orbit.
- Attributes:
  - Inherited: `_name`, `_sensor_system`, and `_mobility`.
  - `_orbit_altitude` (`float`): Operational altitude in kilometers (e.g., `400.5`, `500.0`).
- Methods:
  - `move()`: Returns the string: `Orbiting: Using <mobility> at <orbit altitude> km altitude`
    - Example Output: `Orbiting: Using thrusters at 400.5 km altitude`
  - `perform_task(task: str)`: Returns a string of the performed task. `OrbitalRobot` robots can perform only two tasks:
    - `"repair"` displays the robot's model, calls `move()`, and displays `Repairing: Satellite`. Example Output:

```
-Robot: OrbitFixer
    Orbiting: Using thrusters at 400.5 km altitude
    Repairing: Satellite
```

    - `"maintenance"` displays the robot's model, calls `move()`, and displays `Inspecting: Station hull`. Example Output:

```
-Robot: OrbitFixer
    Orbiting: Using thrusters at 400.5 km altitude
    Inspecting: Station hull
```

RoboticFleet Class

- Purpose: Represents a fleet of space robots.
- Attributes:
  - `_robots` (`list[SpaceRobot]`): Collection of `SpaceRobot` objects.
- Methods:
  - `add_robot(robot: SpaceRobot)`: Adds a robot to the fleet and returns a confirmation string.
    - Example Output: `Added MarsExplorer to fleet`
  - `remove_robot(robot: SpaceRobot)`: Removes a robot from the fleet and returns a confirmation string.
    - Example Output: `Removed MarsExplorer from fleet`
  - `deploy_mission(task: str)`: Executes `perform_task(task)` on all robots in the fleet, returning a string of combined results.
  - Example Output: For `"mapping"` with a fleet containing a `PlanetaryRover` and an `OrbitalRobot`:

```
-Robot: MarsExplorer
    Moving: Using wheels across rocky terrain
    X-500 (LiDAR): Scanning terrain with 0.5° resolution
    CamPro (Camera): Capturing image at 12MP resolution
-Robot: OrbitFixer
    Task 'mapping' not supported
```

  - `report_status( )`: Returns a string summarizing the status of all robots in the fleet (`_name` and `_mobility`).
    Example Output:

```
-Robot: MarsExplorer, Mobility: wheels
-Robot: OrbitFixer, Mobility: thrusters
```

Class Diagram

Create a class diagram that captures class attributes and methods, aggregation, inheritance, and composition. Convert this class to PDF and place it in the folder 📂 ***diagram***.

Demonstration

Demonstrate different aspects of your program by following the instructions from
main.py

Module Imports

The second lecture on OOP demonstrates how to import modules located in different directories. Refer to the in-class examples to guide you in importing the necessary modules for this assignment.

Grading Rubric (1/2) 20 pts

- ***Module Imports and Package Structure (3 pts)***
  - ***2 pts***: Correctly import classes from 🗁 ***robot*** and 🗁 ***sensor*** subdirectories in 🗎 ***main.py***, following the second OOP lecture examples.
  - ***1 pt***: Adhere to the package structure (🗁 ***rwa3_<lastname>***) with required subfolders (🗁 ***space_robotics***, 🗁 ***diagram***, 🗁 ***models/robot***, 🗁 ***models/sensor***).
- ***Sensor Hierarchy and Composition (5 pts)***
  - ***1 pt***: Implement `Sensor` as an abstract base class with `_model` and `_range` attributes and abstract methods `get_info()` and `operate()`.
  - ***1 pt***: Correctly define `LiDAR` with `_angular_resolution` and required method overrides.
  - ***1 pt***: Correctly define `Camera` with `pixel_resolution` and required method overrides.
  - ***2 pts***: Implement `SensorSystem` to compose `Sensor` instances, with `_sensors` and `_status` (initially `"inactive"`).
- ***Robot Hierarchy and Composition (2 pts)***
  - ***1 pt***: Implement `SpaceRobot` as an abstract base class with `_name`, `_sensor_system`, and `_mobility`.
  - ***1 pt***: Correctly compose `SensorSystem` within `SpaceRobot` during initialization.

Grading Rubric (2/2) 20 pts

- ***Robot Hierarchy and Composition (cont.) (3 pts)***
  - ***1 pt***: Define `PlanetaryRover` with `_terrain_type` and implement `move()` and `perform_task()` for `"mapping"` and `"collection"`.
  - ***1 pt***: Define `OrbitalRobot` with `_orbit_altitude` and implement `move()` and `perform_task()` for `"repair"` and `"maintenance"`.
  - ***1 pt***: Ensure encapsulation with protected attributes and controlled access (e.g., via `@property`).
- ***Robotic Fleet and Aggregation (3 pts)***
  - ***1 pt***: Implement `RoboticFleet` with `_robots` as a list aggregating `SpaceRobot` instances.
  - ***1 pt***: Correctly implement `add_robot()` and `remove_robot()` with appropriate return strings.
  - ***1 pt***: Implement `deploy_mission()` and `report_status()`.
- ***Polymorphism Demonstration (2 pts)***
  - ***2 pts***: Demonstrate polymorphic behavior in ***main.py*** by deploying missions (`"mapping"`, `"collection"`, `"repair"`, `"maintenance"`) with varying outputs for `PlanetaryRover` and `OrbitalRobot`.
- ***Deliverables and Program Execution (2 pts)***
  - ***1 pt***: Submit a compressed file (***rwa3_<lastname>.<extension>***) with all required components and a UML class diagram in ***diagram***.
  - ***1 pt***: Ensure ***main.py*** executes without errors, producing clear output matching design examples.

Deductions

- Late submission: ***As per syllabus guidelines***.
- Use of AI-generated code: Assignment grade of ***zero***.
- Code sharing/plagiarism: Assignment grade of ***zero***.
- Non-private GitHub repository: Assignment grade of ***zero***.
- Classes lack adequate documentation (missing docstrings) or insufficient inline comments: ***-2 pts***.
- If getters and setters are used in your program, ensure you do so with the `@property` decorator. If any other approaches are used, you will incur a penalty of ***-2 pts***.
- If even a single attribute is not prefixed with an underscore: ***-2 pts***.