MARYLAND APPLIED
GRADUATE ENGINEERING

Z. Kootbally
Fall 2023
UMD
College Park, MD

RWA2 (v1.3)

ENPM809Y: Introductory Robot Programming

Due date: **Friday, November 10, 2023, 11:59 pm**

Contents

1 Changelog

- **v1.3** (11/10/2023)
  - Section 5.1.2: Changed to `Battery <model_> is charging.` and `Battery <model_> is fully charged.`
  - Sections 5.1.5 and 5.1.4: Changed to "Call `take_off(distance/2)`" and "Call `dive(distance/2)`", respectively.
  - In the class diagram: Modified the type of the attribute `sensors_` (in class `MobileRobot`) to `std::vector<std::unique_ptr<RWA2::Sensor>>`
- **v1.2** (11/5/2023): Fixed typo for the method `jump()` (page 10). It is supposed to read "= 20 cm" instead of "- 20 cm".
- **v1.1** (11/1/2023): Slight modification and addition about the method `rotate(double angle)`. In the base class, only add or subtract angle to the orientation of the robot. In the overriden methods, only print something in the terminal. Deadline extended to Friday (from Thursday).
- **v1.0** (10/28/2023): Original release of the document.

2 Disclaimer

Certain commercial products or company names are identified in this document to describe examples of robots, batteries, and sensors. Such identification is not intended to imply recommendation or endorsement by the University of Maryland, nor is it intended to imply that the products or names identified are necessarily the best available for the purpose.

3 Conventions

In this document, you will find a number of text styles that distinguish between different kinds of information.

- link folder
- Important note.
- List of tasks to do.

4 Overview

- This assignment must be done as a group.
- The workspace structure for this assignment is shown below.

```
RWA2-<Group#>
├── include
│   └── *.h
├── src
│   ├── main.cpp
│   └── *.cpp
└── doc
```

- The folder src contains all your *.cpp files, including main.cpp.
- The folder include contains all your *.h files.
- The folder doc contains HTML Doxygen documentation for your project.

- Once you are done with the assignment, zip the folder RWA2-<Group#> and upload it on Canvas.
- Do not upload your .vscode folder.

5 Assignment

This assignment focuses on OOP and you will need to demonstrate encapsulation, abstraction, inheritance, and polymorphism. This assignment consists of 1) Creating C++ classes based on a UML class diagram and of 2) Instantiating classes. You need to replace anything in `<>` with actual attributes.

If you followed along during classes on Doxygen and OOP, and completed the exercise as asked in Lecture 7, then you have accomplished 60 % of the assignment.

5.1 Class Diagram

The class diagram for this assignment is shown in Figure 1. A better resolution of the diagram is also provided as a standalone file on Canvas.

- The namespace used for all classes is `RWA2`. This can be seen in each class with the notation ***(from RWA2)***.
- Classes with the notation ***{leaf}*** are classes that cannot be derived from. Use the correct C++ specifier for these classes.
- There is an aggregation relationship between `Sensor` and `MobileRobot`.
- There is a composition relationship between `Battery` and `MobileRobot`.
- `AquaticRobot`, `AerialRobot`, `WheeledRobot`, and `LeggedRobot` `public`ly derive `MobileRobot`.

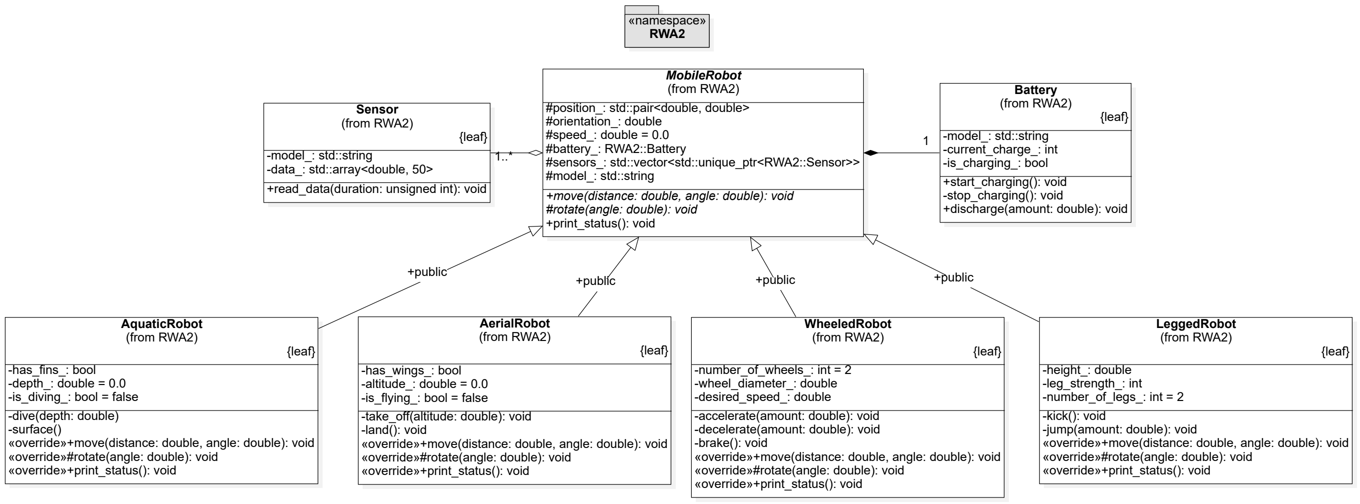
«namespace»
RWA2
MobileRobot
(from RWA2)
#position_: std::pair<double, double>
#orientation_: double
#speed_: double = 0.0
#battery_: RWA2::Battery
#sensors_: std::vector<std::unique_ptr<RWA2::Sensor>>
#model_: std::string
+move(distance: double, angle: double): void
#rotate(angle: double): void
+print_status(): void
Sensor
(from RWA2)
{leaf}
-model_: std::string
-data_: std::array<double, 50>
+read_data(duration: unsigned int): void
1..*
Battery
(from RWA2)
{leaf}
-model_: std::string
-current_charge_: int
-is_charging_: bool
+start_charging(): void
-stop_charging(): void
+discharge(amount: double): void
1
+public
+public
+public
+public
AquaticRobot
(from RWA2)
{leaf}
-has_fins_: bool
-depth_: double = 0.0
-is_diving_: bool = false
-dive(depth: double)
-surface()
«override»+move(distance: double, angle: double): void
«override»#rotate(angle: double): void
«override»+print_status(): void
AerialRobot
(from RWA2)
{leaf}
-has_wings_: bool
-altitude_: double = 0.0
-is_flying_: bool = false
-take_off(altitude: double): void
-land(): void
«override»+move(distance: double, angle: double): void
«override»#rotate(angle: double): void
«override»+print_status(): void
WheeledRobot
(from RWA2)
{leaf}
-number_of_wheels_: int = 2
-wheel_diameter_: double
-desired_speed_: double
-accelerate(amount: double): void
-decelerate(amount: double): void
-brake(): void
«override»+move(distance: double, angle: double): void
«override»#rotate(angle: double): void
«override»+print_status(): void
LeggedRobot
(from RWA2)
{leaf}
-height_: double
-leg_strength_: int
-number_of_legs_: int = 2
-kick(): void
-jump(amount: double): void
«override»+move(distance: double, angle: double): void
«override»#rotate(angle: double): void
«override»+print_status(): void

Figure 1: Class diagram for RWA2.

- The stereotype ***«override»*** means that a method overrides a base class method.
- Constructors, destructors, accessors, and mutators are not represented in class diagrams.
- You are free to add more methods if needed but you cannot modify the method signatures and access specifiers of the methods shown in the diagram.
- You are free to add more attributes if needed but you cannot change the names, types, and access specifiers of the attributes shown in the diagram.

5.1.1 Class Sensor

This class represents the sensor(s) that can be mounted on a mobile robot.

- `model_` – Model of the sensor. In this assignment, we will use the following models:
  - From Ouster: ***OSDome***, ***OS0***, ***OS1***, and ***OS2***.
  - From SICK: ***multiScan100***, ***LMS5xx***, and ***LD-MRS***.
- `data_`: An array of 50 floating point numbers. Each number has a value between 0 and 30. This describes a sensor with 50 beams and each beam can detect an object within a distance of 30 m. This is a simplification.
- `read_data(unsigned int duration)` – Gather data during a number a seconds (`duration`). To emulate sensor data acquisition, make your program sleep for `duration` seconds. After the time is up, fill out the array `data_` with random numbers between 0 and 30. This method also prints:

```
Sensor <model_> gathering data for <duration> seconds.
```

5.1.2 Class Battery

This class represents a battery on a mobile robot.

- `model_` – Model of the battery. In this assignment, the available models are ***Li-ion*** or ***LiFePO4***.
- `current_charge_` – Percentage representing the current charge of the robot. The min current charge is 0% and the max current charge is 100%.
- `is_charging_` – `true` if the battery is charging, `false` otherwise.
- `start_charging()` – Action of charging the battery. This method can only be called if `is_charging_` is `false`. The method should calculate the remaining charge time based on the `current_charge_` attribute. For instance, if current_-charge_ is at 86%, the battery needs to be charged the remaining 14%. This will take 7 seconds for a ***Li-ion*** battery and 3.5 seconds for a ***LiFePO4*** battery. The method should then sleep the program for the respective amount of time, simulating the charging process. This method also prints:
  ```
  Sensor <model_> is charging.
  ```
- `stop_charging()` – This method is called when `current_charge_` is `100`. The attribute `is_charging_` is set to `false`. This method is `private` as it is not meant to be called outside the class. This method also prints:
  ```
  Sensor <model_> is fully charged.
  ```
- `discharge(double amount)` – Each time this method is called, the battery's `current_charge_` decrements by `amount`.

5.1.3 Class MobileRobot

Abstract class which provides functionalities to derived classes. In UML, an abstract class is represented by using italic font for the class name. All the attributes of this class are inherited in derived classes and most of the attributes have been discussed in Lectures 7 and 8.

`move(double distance, double angle)` is pure `virtual`.

- `model_` – This is the robot model. We will use one of the following models for derived classes.
  - `AquaticRobot` – ***HoloOcean*** and ***SoFi***.
  - `AerialRobot` – ***Erle-Plane*** and ***Crazyflie***.
  - `WheeledRobot` – ***Turtlebot*** (2-wheeled), ***Hiwonder*** (4-wheeled), and ***Dagu*** (6-wheeled).
  - `LeggedRobot` – ***Atlas*** (bipedal), ***Spot*** (quadrupedal), and ***Mx-Phoenix*** (hexa-

pod).

- Implement `rotate(double angle)` as follows: Add `angle` to `orientation_` if `angle` is positive or subtract `angle` from `orientation_` if `angle` is negative. This method is marked as `protected` but not overriden in the derived classes. This means that we can call this method in the derived classes but not in the `main()` function.
- Implement `print_status()` to display the position, the orientation, and the speed of a mobile robot. Reuse the work from Lecture 8.

5.1.4 Class AquaticRobot

This is a concrete class for underwater (aquatic) robots.

- `has_fins_` – `true` if the robot has fins, otherwise `false`. The robot ***SoFi*** is the only one with fins.
- `depth_` – Current depth of the robot in meter. Objects created from `AquaticRobot` are initialized with a `depth_` of `0.0`.
- `is_diving_` – `true` if the robot is diving, otherwise `false`. `AquaticRobot` objects are initialized with `is_diving_` set to `false`.
- `dive(double depth)` – The robot dives to reach `depth`. Write the logic for this method as follows:
  - This method can be called only if the robot is not already diving. A robot with fins moves at 2 m/s and a robot with no fins moves at 1 m/s. To emulate the robot diving, make the program sleeps the correct amount of time until `depth` is reached. For instance, if the robot has to reach a depth of 10.0 meters and the robot moves at a speed of 2 m/s, then make the method sleeps 5 s.
- `surface()` – Action of moving the robot to the surface. This method can only be executed if the robot is already diving. Once the robot reaches the surface, its battery must be fully recharged. Make sure to set the correct value for the attributes of this class. Write the logic for this method as follows:
  - To reach the surface, a robot with fins moves at 4 m/s and a robot with no fins moves at 2 m/s. Make the program sleeps the correct amount of time until `depth_` is `0.0`. For instance, if the robot is at a depth of 10.0 meters and the robot moves at a speed of 4 m/s, then this method should sleep for 2.5 s.
- Override `rotate(double)` by just add a line which prints:
  ```
  AquaticRobot::<model_> rotated <angle> degrees.
  ```
- Override `print_status()` as shown in Lecture 8. Print the base class attributes and the `AquaticRobot` class attributes.
- Override `move(double distance, double angle)`. The max `distance` value is 100. Each meter the robot moves, 1% of the battery's charge is consumed. When

this method is called, the following happens:

- Check the battery's charge is enough to dive and surface.
  - If we call `move(25, 45)`, the robot has to reach a depth of 25 meters and then come back to the surface. In total the robot has to swim 50 meters and 50% of the battery's charge is consumed. If the current charge is enough to dive and surface, there is no need to charge the battery, otherwise, charge the battery.
- Once the robot has enough charge, proceed as follows.
  - Call `read_data(5)`
  - Call `rotate(angle)`
  - Call `dive(distance/2)`
  - Call `surface()`
  - Print: <model_> reached a depth of <distance> meters and then surfaced.
  - Call `print_status()`

5.1.5 Class AerialRobot

This is a concrete class for aerial robots.

- `has_wings_` – `true` if the robot has wings, `false` otherwise. Only the model ***Erle-Plane*** has wings.
- `altitude_` – Current altitude of the robot in meter. Objects created from `AerialRobot` are initialized with `altitude_` of `0.0`.
- `is_flying_` – `true` if the robot is flying, otherwise `false`. `AerialRobot` objects are initialized with `is_flying_` set to `false`.
- `take_off(double altitude)` – The robot takes off to reach the altitude. Write the logic for this method as follows:
  - If the robot is not already flying then this method can be called. A robot with wings moves at 3 m/s and a robot with no wings moves at 1.5 m/s. Make the program sleeps the amount needed to emulate the robot reaching the altitude.
- `land()` – Action of landing the robot. Write the logic for this method as follows:
  - This method can be executed only if the robot is flying. A robot with wings lands at a speed of 4 m/s and a robot with no wings lands at a speed of 2 m/s. Make the program sleeps the amount of time needed to emulate the robot has landed.
- Override `rotate(double)` by just add a line which prints:
  AerialRobot::<model_> rotated <angle> degrees.
- Override `print_status()` as shown in Lecture 8. Print the base class attributes and the `AerialRobot` class attributes.
- Override `move(double distance, double angle)`. The max `distance` value is 50.

Each meter the robot moves, 2% of the battery's charge is consumed. When this method is called, the following happens:

- Check the battery's charge is enough to take off and land. If the current charge is enough to reach the altitude and then come back to the ground, then there is no need to charge the battery, otherwise, charge the battery.
- Once the robot has enough charge, proceed as follows.
  - ◇ Call `read_data(5)`
  - ◇ Call `take_off(distance/2)`
  - ◇ Call `rotate(angle)`
  - ◇ Call `land()`
  - ◇ Print: <model_> reached an altitude of <distance> meters and then landed.
  - ◇ Call `print_status()`

5.1.6 Class WheeledRobot

This is a concrete class for wheeled robots.

- `number_of_wheels_` – Number of wheels of the robot. If not provided, this attribute defaults to 2. Possible values are 2, 4, or 6.
- `wheel_diameter_` – Diameter of each wheel in centimeter.
- `desired_speed_` – Desired speed of the robot.
- `accelerate(double amount)` – In a `while` loop, add `amount` to `speed_` until `speed_` is less or equal to `desired_speed_`. Make the method sleep for 0.5 s each time `amount` is added to `speed_`. Once `desired_speed_` is reached, print:
  <model_> has reached the desired speed of <desired_speed_> m/s
- `decelerate(double amount)` – In a `while` loop, decrease `speed_` by `amount` while `speed_` is greater or equal to zero (make sure `speed_` does not become negative). Make the method sleep for 0.5 s each time `speed_` decreases by `amount`.
- `brake()` – The robot stops moving and its `speed_` is zero.
- Override `rotate(double)` by just add a line which prints:
  WheeledRobot::<model_> rotated <angle> degrees.
- Override `print_status()` as shown in Lecture 8. Print the base class attributes and the `WheeledRobot` class attributes.
- Override `move(double distance, double angle)`. The max `distance` value is 100. Each meter the robot moves, 1% of the battery's charge is consumed. When this method is called, the following happens:
- Check the battery's charge is enough to drive the given distance. If the current charge is enough then there is no need to charge the battery, otherwise, charge the battery.
- Once the robot has enough charge, proceed as follows.
  - Call `read_data(5)`

  - Call `rotate(angle)`
  - Call `accelerate(2)`
  - Sleep for (`distance`-2) seconds
  - Call `decelerate(2)`
  - Call `brake()`
  - Print: <model_> drove <distance> m.
  - Call `print_status()`

5.1.7 Class LeggedRobot

This is a concrete class for legged robots.

- `height_` – Distance between the robot's feet and the ground.
- `leg_strength_` – Strength of the robot's legs (1-10 scale).
- `number_of_legs_` – Number of legs of the robot. Possibilities are 2, 4, and 6. If not provided, this attribute defaults to 2.
- `kick()` – The robot gives a kick with a strength of `leg_strength_`. This method only prints the message:
  <model_> kicks with a strength of <leg_strength_>
- `jump(double amount)` – The robot jumps `amount` times its `leg_strength_` in centimeters. `amount` is in meter. For instance if `leg_strength_` is 2 and `amount` is 10, the robot jumps: `height_` = 0.10 × 2 = 20 cm. This method also prints:
  <model_> jumps at a height of <height_> cm above the ground
- Override `rotate(double)` by just add a line which prints:
  LeggedRobot::<model_> rotated <angle> degrees.
- Override `print_status()` as shown in Lecture 8. Print the base class attributes and the `LeggedRobot` class attributes.
- Override `move(double distance, double angle)`. The maximum value for `distance` is 10 meters. For every centimeter the robot jumps above the ground, it uses up 1% of its battery charge. For instance, if `height_` is 0.30, 30% of the battery is consumed. Each kick consumes 1% per `leg_strength_`. When this method is called, the following happens:
- Check the battery's charge for one jump and one kick. If the current charge is enough then there is no need to charge the battery, otherwise, charge the battery.
- Once the robot has enough charge, proceed as follows.
  - Call `read_data(5)`
  - Call `rotate(angle)`
  - Call `jump(distance)`
  - Call `kick(leg_strength_)`
  - No need to print anything here as the methods `kick` and `jump` already print a message.

  - Call `print_status()`

5.2 Objects

To test your classes with dynamic polymorphism, reuse the approach seen in Lecture 8 (slide 64). Replace
`void get_status(const std::vector<std::unique_ptr<RWA2::MobileRobot>>& robots)`
with
`void move_robot(const std::vector<std::unique_ptr<RWA2::MobileRobot>>& robots)`.
In the body of the function, call `robot->move(value1, value2)` where `value1` and `value2` are values of your choice.

Do not forget to add sensors to your robots.

5.3 Documentation

All classes, methods, and attributes must be documented using Doxygen. Generate the HTML documentation in the folder doc of your project.

6 Submission

- Zip the workspace and upload it on Canvas when you are done with the assignment.
- Any submission passed the deadline will incur a penalty, even if it is 1 minute after the deadline. Rules which pertain to penalty can be found in the syllabus.
- There is no extension for this assignment unless you provide documented justifications. If you have other submissions due around the same time, you are responsible for organizing your work to submit the assignment on time.
- Finally, a grade of 0 (zero) will be assigned for any plagiarized submission.

7 Grading Rubric

A new grading rubric will be provided soon. In the meantime, make sure you have done the following:

- The access specifiers used in your code should match the ones from the class diagram.
- Doxygen documentation should only be in *.h files.
- You need to comment your code.
- You need to follow best practice for OOP: Mark overriden methods with `virtual` and `override`.
- Follow the naming convention and be consistent.
- Test your program with different values.
- main.cpp should only contain the `main` and the `move_robot` functions.
- Each *.h must contain header guards or `#pragma once`
- The instructions show what some methods must print in the terminal. You have to exactly print what is provided in this document.