Cleaning Robot - TDD Group

Goal

The goal of this task is to develop a cleaning robot; the robot moves in a room and cleans the dust on the floor along the way. To clean the dust, the robot is equipped with a **cleaning system** placed below it. When the robot is turned on, it turns on the cleaning system.

The robot moves into the room, thanks to two **DC motors**, one that controls its wheels and another that controls its rotation, by executing a command, which a **Route Management System (RMS)** sends to the robot. While moving in the room, the robot can encounter obstacles; these can be detected thanks to an **infrared distance sensor** placed in the front.

The robot can check the charge left in its internal battery. To do so, it is equipped with an **Intelligent Battery Sensor (IBS**).Furthermore, a recharging **LED** is mounted on the top of the robot to signal that it needs to be recharged.

The room, where the robot moves, is represented as a rectangular grid with x and y coordinates. The origin cell of the grid – i.e., (0,0) – is located at the bottom-left corner. A cell of the grid may contain or not an obstacle. The RMS keeps track of the room layout, including the last known positions of the obstacles in the room.

To recap, the following sensors, actuators, and systems are present:

- A DC motor to control the wheels in order to move the robot forward.
- A DC motor to control the rotation of the body of the robot, in order to make it rotate left or right.
- An RMS, sending commands to the robot.
- An infrared distance sensor used to detect obstacles.
- An IBS to determine the battery charge left.
- A recharge LED.
- A cleaning system.

The communication between the main board and the other components happens with GPIO pins; GPIO communication is configured in BOARD mode. For further details on how to use the GPIO library refer to the **mock.GPIO** class in the source code.

Handle any error situation that you may encounter by throwing the **CleaningRobotError** exception.

Instructions

Depending on your preference, either clone the repository at https://github.com/Espher5/cleaning_robot or download the source files as a ZIP archive; afterwards, import the project into PyCharm.

Take a look at the provided project, which contains the following classes:

- **`CleaningRobot`**: you will implement your methods here.
- **`CleaningRobotError`**: exception that you will raise to handle errors.
- **`CleaningRobotTest`**: you will write your tests here.
- **`mock.GPIO`**: contains the mocked methods for GPIO functionalities.

Remember, you are **NOT ALLOWED** to modify the provided API in any way (i.e., class names, method names, parameter types, return types). You can however add fields, methods, or even classes (including other test classes), as long as you comply with the provided API.

Use **TDD** to implement this software system.

The requirements of the software system to be implemented are divided into a set of **USER STORIES**, which serve as a to-do list; you should be able to incrementally develop the software system, without an upfront comprehension of all the requirements. **DO NOT** read ahead and handle the requirements (i.e., specified in the user stories) one at a time in the order provided.

When a story is **IMPLEMENTED**, move on to the **NEXT** one. A story is implemented when you are confident that your software system correctly implements all the functionality stipulated by the story's requirement. This implies that all your **TESTS** for that story and all the tests for the previous stories **PASS**. You may need to review your software system as you progress towards more advanced requirements.

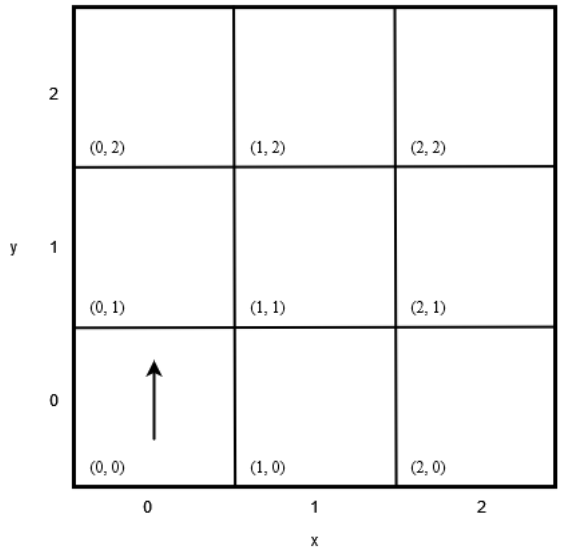
At the end of the task, fill out the post-questionnaire where, among other things, you will be asked to share your project (either as a link to a GitHub repository or as a sharing link to a ZIP file). The post-questionnaire is available at: https://forms.gle/HuUgizBoDuQCeQXv5.

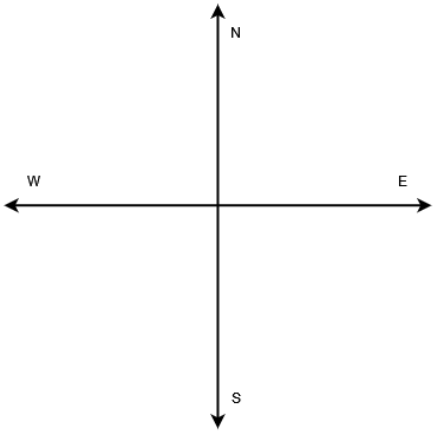
User Stories

Remember to use **TDD** to implement the following user stories.

1. Robot deployment

The robot has a status, namely a string (without white spaces) formatted as follows: (x,y,h). The pair (x,y) represents the position of the robot in the room (in terms of x and y coordinates) while h is the heading – i.e., the direction the robot is pointing towards; the direction can be: N (North), S (South), (E) East, or (W) West. The robot assumes the North is parallel to the y axis. The robot starts its duty from the origin position—i.e., the position with coordinates (0,0), facing North (see the figure below).





Requirement:

- Implement **`CleaningRobot.initialize_robot() -> None`** to set the status of the robot to "(0,0,N)".
- Implement **`CleaningRobot`**. **`robot_status() -> str`** to retrieve the current status of the robot.

Example:

- The robot status “(0,0,N)” indicates that the robot lies in the cell with x and y coordinates both equal to 0. The heading of the robot is N – i.e., it is pointing North.

2. Robot startup

When the robot is turned on, it first checks how much battery is left by querying the IBS. If the capacity returned by the IBS is equal to or less than 10%, the robot turns on the recharging led. Otherwise, the robot turns on the cleaning system and sends its status to the RMS. In any case, the robot stands still.

The IBS is connected on pin 11 (BOARD mode). The communication with the sensor happens via the GPIO input function. For this sensor, the value returned by the GPIO input function is assumed to be a percentage indicating the amount of battery charge left.

The recharge LED is connected on pin 12 (BOARD mode). The communication with the sensor happens via the GPIO output function.

Finally, the cleaning system is connected on pin 13 (BOARD mode). Assume this component to be an independent system; the communication with it happens via the GPIO output function.

The pin for the IBS, the one for the recharge LED and the one for the cleaning system have already been set up in the constructor of the `CleaningRobot` class.

Since at this stage of development there is no way to determine the state of the physical LED and cleaning system, use the `self.battery_led_on` and `self.cleaning_system_on` instance variables to keep track of their states.

Requirement:

- Implement `CleaningRobot.manage_battery() -> None` to control the behavior associated with the battery level.

3. Robot movement

To move into the room, the robot must receive a command from the RMS; when this happens, the robot controls the wheel motor and the rotation motor in order to execute the command and finally returns its new status to the RMS.

The robot moves one cell forward when it receives the command “f”. If the robot receives the command “r” or “l”, it turns right or left respectively – i.e., it rotates clockwise or counterclockwise 90° around itself. The robot cannot move backwards.

The two motors that control the wheels of the robot and the rotation of the body are DC motors. The pins to connect them to the main board have already been set up in the constructor of the `CleaningRobot` class.

In order to control them, please use the following two methods in the `CleaningRobot` class:

- `activate_wheel_motor() -> None` activates the wheel motor to make the robot move forward.
- `activate_rotation_motor(direction: str) -> None` activates the rotation motor to make the robot turn left or right, based on the `direction` parameter (“l” to turn left or “r” to turn right).

Requirement:

- Implement `CleaningRobot.execute_command(command: str) -> str` to execute the command corresponding to the string given by the RMS and return the status of the robot.

Example:

- If the robot status is “(0,0,N)” and it receives the command “f”, the robot moves one cell forward – i.e., the robot controls the wheel motor in order to move one cell forward – and returns the new status “(0,1,N)”.
- If the status of the robot is “(0,0,N)” and it receives the command “r”, the robot rotates clockwise 90° – i.e., the robot controls the rotation motor in order to rotate 90° – and returns the new status “(0,0,E)”. Similarly, if it receives the command string “l”, the robot rotates counterclockwise 90° and returns the new status “(0,0,W)”.

- If the status of the robot is “(1,1,N)” and it receives the command “f”, the robot moves forward and returns the new status “(1,2,N)”. If the robot receives the command “l” afterwards, it turns left and returns the new status “(1,2,W)”.

Hint:

- Remember that you can use the `robot_status() -> str` method in the `CleaningRobot` class to retrieve the current status of the robot.
- Use the class variables provided in the `CleaningRobot` class to update the rotation of the robot (`N`, `E`, `S`, `W`)

4. Obstacle detection

While executing a command, the robot can encounter an obstacle in a cell; the robot uses the infrared distance sensor to determine if there is an obstacle in front of it and thus avoid bumping into that obstacle. If an obstacle is detected, the robot cannot move beyond the obstacle; it will instead return its new status, including the positions of the encountered obstacle. In this case, the robot status is a string formatted as follows: “(x,y,h)(xo,yo)”. The triple “(x,y,h)” represents the usual position and heading of the robot while the pair “(xo,yo)” represents the position of the encountered obstacle.

In case multiple commands are executed in sequence, and multiple obstacles are found, the system only keeps track of the last one encountered.

The infrared sensor is connected to the main board on pin 15 (BOARD mode); communication with the sensor happens via the GPIO input function. The pin has already been set up in the constructor of the `CleaningRobot` class.

Finally, the output of the infrared sensor is an analog signal which changes intensity according to the distance between the sensor and the object (i.e., 2.5V when an object is 50 cm away and ~0V when the object is out of the max range of the sensor). For this exercise, let’s assume the input can be classified into just these two categories:

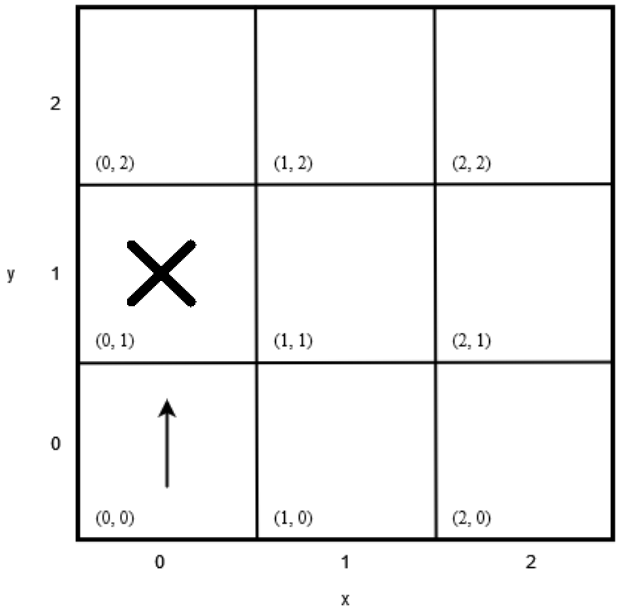
- Non-zero value: it indicates that an object is present in front of the sensor (i.e., an obstacle).
- Zero value: nothing is detected in front of the sensor.

Requirement:

- Implement **`CleaningRobot.obstacle_found() -> bool`** to determine whether an obstacle is in front of the robot (as detected by the infrared distance sensor).
- Update the implementation of **`CleaningRobot.execute_command(command: str) -> str`** to include the obstacle processing steps.
- Update the implementation of **`CleaningRobot`**. **`robot_status() -> str`** to retrieve the current status of the robot, including the possible encountered obstacle.

Example:

- Let us suppose that there is an obstacle in the room at coordinates (0,1) (see figure below). The robot with initial status "(0,0,N)", after executing the command string "f", returns the following status: "(0,0,N)(0,1)".



Hint:

- To keep track of the position of the obstacle after detecting it, use the **`self.obstacle`** instance variable in the **`CleaningRobot`** class.

5. Robot recharge.

Before making any kind of movement/rotation, the robot checks how much battery is left by querying the IBS. When the capacity returned by the IBS is equal to or less than 10%, the robot shuts down the cleaning system, turns on the recharging led, and stands still (i.e., it doesn't update its position in any way); otherwise.

Requirement:

- Update the implementation of `CleaningRobot.execute_command(command: str) -> str` to control the behavior associated with the battery level.
- Update the implementation of `CleaningRobot.manage_battery() -> None` to control the behavior associated with the battery level.