Robotics + Python assignment

For this assignment you must complete Lab 7 – Hardware-in-the-Loop Simulation, including the challenge to find the shortest path. A full description of the tasks with implementation details is available at: https://felipenmartins.github.io/Robotics-Simulation-Labs/Lab7/.

You will work in pairs (or individually, if you prefer) and **demonstrate your working simulation on the 5th of June** (shown in digirooster as *4/Robotics/R&P Demonstration*).

Assessment

Submit to Blackboard your Python and MicroPython files (or a link to GitHub repository) *before* the demonstration.

On the demonstration day, each team will be assigned a time slot for presenting their working simulation and code in **5 minutes**. Please, practice your demonstration and be prepared! Make sure that you can complete your presentation on time! During the demonstration, you are responsible to show the lecturers that you successfully completed the tasks as required.

Even when executed in pairs, the final grade is individual. Your grade will be influenced by the answers to the questions of the lecturers.

Grading

The assessment consists of two parts:

- Python (40% of the final grade)
- Robotics (60% of the final grade)

The final grade will be calculated as:

$$Grade_{final} = \frac{40}{100} Grade_{Python} + \frac{60}{100} Grade_{Robotics}$$

You pass this assignment when $Grade_{final} \geq 5.5$.

A description of how each grade is calculated is given in the rubric below.

Python Programming grade

$Grade_{Python}$**: maximum of 11 points** (1 extra point as bonus)

Code Correctness, Structure, and Clarity (3 points)

- **2.0**: Code runs without errors, performs intended functions correctly, and is logically structured (uses functions/classes appropriately, avoids redundancy).
- **1.0**: Code includes clear inline comments that explain the *intention* behind non-obvious parts of the code.

Code Quality and Design (1.5 points)

- **0.5**: Clear and consistent naming of variables, functions, and classes. Avoids ambiguous names and hardcoding where inappropriate.
- **1.0**: Demonstrates good use of scope and Python best practices:
    - Avoids excessive use of global variables.
    - Uses default arguments appropriately.
    - Loops and operations are placed efficiently (e.g., avoids unnecessary computations inside loops, especially in ESP32).

Modularity and Reusability (2.5 points)

- **1.0**: Code is broken into reusable, testable components (e.g., modular functions or classes).
- **1.5**: Code is generalized enough to be used on different maps or setups with minimal changes.

Version Control and Collaboration (up to 2 points)

- **1.0**: Code is maintained in a version-controlled repository (e.g., GitHub), with meaningful commit history or use of branches.
- **1.0**: *Bonus* For each feature added there is at least a separate commit, with no superfluous commits in-between, except for bug-fixes.

Documentation and Reproducibility (2 points)

- **1.0**: Each function has a docstring explaining its purpose, inputs, outputs, and side effects (if any).
- **1.0**: A README file is included that:
    - Lists dependencies (e.g., pip packages, Python version).
    - Explains how to reproduce the main experiment or result.

Robotics grade

$Grade_{Robotics}$**: maximum of 11 points** (1 extra point as bonus)

HiL Simulation Setup (3 points)

- **3.0**: Correctly implements the Hardware-in-the-Loop (HiL) simulation using:
    - The provided environment.
    - MicroPython code on the ESP32.
    - Python code on Webots.
- **Evidence**: A working Webots simulation where the robot is controlled via code running on the ESP32.

Path-Planning on the ESP32 (up to 4 points)

You get points for implementing **one** of the options:

- **3.0**: Dijkstra's algorithm is implemented and running on the ESP32 to compute the shortest path between the current robot position and a pre-defined goal node.
- **4.0**: A* or D* Lite algorithm is implemented and running on the ESP32 to compute the shortest path between the current robot position and a pre-defined goal node using valid heuristics.

- **Evidence**: The simulated robot on Webots consistently follows the shortest path from its position to any pre-fedined goal node while controlled by the code running on the ESP32.

Obstacle Detection and Re-Planning (2 points)

- **2.0**: The robot uses its proximity sensors to detect obstacles, and re-plans its path automatically upon detecting a new obstacle.
- **Evidence**: When an obstacle is detected using proximity sensors, the robot generates and follows a new valid path to the goal.

Path Visualization (1 point)

- **0.5**: Map and planned path are plotted after the end of the simulation.
- **0.5**: Visualization is updated while the simulation is running, showing the planned path and current node of the robot.
- **Evidence**: The map and planned path for the robot are plotted on screen after (or during) the simulation.

Wireless Communication (1 point)

- **1.0**: Implements wireless communication between the ESP32 and the computer using Wi-Fi or Bluetooth, instead of connecting the ESP32 to the computer via USB.
- **Tip**: A second ESP32 connected via USB may be used to wirelessly communicate with the ESP32 that runs the controller code.