WAREHOUSE PATH PLANNING (100 Points)

In modern automated warehouses, the efficient and safe movement of mobile robots is essential to maintaining high throughput and reliability. This assignment focuses on implementing a path planning software for autonomous warehouse robots operating in such a structured environment.

The warehouse is organized into a grid-like environment where fiducial markers are embedded on the floor. These fiducials provide accurate localization data to the robot, allowing it to determine its position and orientation within the global coordinate frame.

These are the definitions that should be used when considering a warehouse layout:

- A **robot** is a mechanical device with a well-defined physical volume. It can be laden or unladen. If the robot is laden, it carries a payload with a known and fixed physical size.
- **Obstructions** are volumes on the floor that are occupied and have a very high cost. They are dynamic in nature, as, for example, the obstacle detection onboard the robots may detect fallen boxes that block a portion (a cuboid) of the floor.
- **Obstacles** are structural elements of the warehouse that are naturally fixed and known at the time of startup. They have the maximum possible cost.
- The floor consists of evenly spaced **fiducials**, each representing a unique (x, y) coordinate in the warehouse map.
- Each **physical element** of the system (robots, walls, etc.) is represented as an axis-aligned 3D cuboid with defined width, length, and height.
- The **floor map** is a data structure that contains the volumes of every obstacle, obstruction, and robot.

The objective is to design and implement a path planner (with an algorithm of your choice) capable of computing an optimal or near-optimal path for the robot to navigate from a starting location to a target location while considering the volume of goods being transported to avoid collision with other robots, obstacles, and obstructions on the floor.

To simplify the problem, you can assume that only one robot can plan and move at a time. During planning, collision checking between cuboids can be implemented using Axis-Aligned Bounding Box (AABB) intersection tests.

SUBMISSION

Python or C++ is the preferred implementation language. For Python, provide a plain PY file (no Jupyter Notebook). If you are writing in C++, please include a `CMakeLists.txt` file and any other compilation instructions. Your code should run immediately without any additional steps on our part.

Your solution may use any numerical libraries for pre-processing, fundamental calculations (e.g., linear algebra), and visualization. However, the core portion must be implemented from scratch. If you are unsure about a specific library, please ask the teaching staff for guidance first.

Submit your solution as a ZIP file with all the files via Canvas. Include a `README.txt` file that clearly explains all its assumptions.