Autonomous Systems:
Group Project: Sub-Terrain Challange

1 General Information

1.1 Administration

The task is a team effort. Every team member shall approximately equally contribute to the challenge. We ask you to use gitlab or github to store and document you contributions. Therefore, regularly commit your code.

1.2 Access to Code and Simulation

The source code and the simulation can be accessed as follows:

- The source code can be accessed via the course repository: https://github.com/TUM-AAS/SubTerrainChallenge_ROS2 (folder challenge)
- The Unity simulation can be accessed via Sync&Share: https://syncandshare.lrz.de/getlink/fi7Vw11aA5WwyMBRPVQY (folder challenge)

1.3 Timeline

- Today: Announcing the projects
- Today: Release of code

1.4 Submission

The project submission consists of three parts:

- Source Code (Deadline: 03.03.2026 - End of day)
- Documentation (Deadline: 03.03.2026 - End of day)
- Presentation (Deadline: 04.03.2026 - 10am - 1pm)

The presentation will not be graded but shall help the reviewers to quicker understand your code structure, limitations and issues you had during implementation.

Your source code and documentation must be submitted via a link to a github repository. In both cases you should include the src folder of your ROS2 workspace as well as the source code folder(s).

1.4.1 Source Code

While writing code, please pay attention to writing clean code, think about how to structure your code into individual ROS packages and nodes.

Please also include a readme.md file with detailed step by step installation and launch instructions necessary to run your code base. Make sure that your project can be launched by a single launch file. We don't want to launch your code from multiple terminals. Indicate in your readme which Ubuntu version has been used. As a hint, before submission run your code from a fresh installation of Ubuntu to make sure you have all necessary installation steps in your documentation!

You are allowed to use any free available code but you are NOT allowed to use source code from other groups of this course.

1.4.2 Documentation

The documentation shall be approx. 4-6 pages but is not limited to that. The goal of the documentation is to help us understanding your code. In your documentation you should include a short description of every ROS node and package and its functionality. Please let us as well know in case you were not able to fulfill all requirements and explain the problems. Document where you used external code, not written by yourself. Finally, discuss who of the team members worked on which components. Beyond that, you must include

- a ROS graph, indicating who worked on which part
- figures, and plots presenting your results,
- a bibliography.

1.4.3 Presentation

Please prepare an oral presentation of your documentation. The presentation length must not be longer than 5 minutes. An introduction to the field and topic is not required. The goal of the presentation is to give us a quick overview over your project, explain how you implemented the individual components, discuss limitations and issues and give us feedback about the project work.

2 Task

2.1 Goals

The goal of the project is two-folded.

- Find as fast as possible four objects of interest (lanterns) in the cave environment and determine the location of the objects (for reference an additional latern is placed at the starting point of the drone). Output the latern pose (publish or write to a file).
- Generate either a 3D voxel-grid representation or a mesh representation of the cave environment. Output the representation of the cave (publish or write to a file).

The core parts, including but not limited are:

- A Unity simulation environment. A base version will be provided to you.

- A ROS-Simulation-Bridge providing ROS interfaces (topics, services). It will communicate with the simulation via TCP while at the same time providing relevant information to other ROS nodes. A base version will be provided to you. You are free to adjust the code.
- A quadrotor controller, enabling position control of the drone. A base version will be provided to you. You are free to adjust the code.
- A state machine for your robot, managing the take-off, travelling and landing at the goal location.
- A perception pipeline that converts the depth image, first to a **point cloud** and second to a **voxel-grid** representation of the environment.
- A path planner that generates a path through the environment to the goal location.
- A trajectory planner that plans a trajectory based on the found path.

2.2 Grading

You can reach a maximum of 100 points in the following categories:

- Functionality and Performance (max. 80p)
  - Successfully working perception pipeline (max. 20p)
  - Successfully working path and trajectory planning (max. 20p)
  - Successfully arriving at the entrance of the cave (max. 5)
  - Successfully finding all four objects of interest (max. 10p)
  - Successfully building a voxel-grid/mesh representation of the cave environment(max. 10p)
  - Success in the full mission (take-off, trajectory tracking and landing at goal location) (max. 10p)
  - Time to complete the mission (max. 5p)
- Code and Architecture Quality (max.10p)
  - Sensible separation in packages, nodes and functions (max. 4p)
  - Sensible separation in services, topics, etc (max. 4p)
  - Code quality, readability, and traceability (max. 2p)
- Written Summary (max. 10p)
  - Sound explanation of your architecture, model, and design choices (max. 5p)
  - Clear documentation who did what, which code is your own, which code is reused (max. 5p)

Very important: We will subtract up to 30p if your code does not build or performs differently then documented. We will do individual grades if we see that individual team members contributed significantly less or more then others.

2.3 Tips

How do start:

1. Download the source code and copy it to your ROS workspace
2. Build the code from your terminals

3. Copy the unpacked simulation files (see Sync&Share folder for the files) into the folder devel/lib/simulation
4. Run 'roslaunch simulation simulation.launch' from your terminal (don't forget to make the executable Simulation.x86_64 executable)

Here are a couple of hints regarding the implementation. The hints are just suggestions, you are free so solve the task differently:

- Use the semantic camera to detect the objects of interest.
- You are allowed to use a predefined waypoint navigation to come to the entrance of the cave. During the exploration of the cave a pre-defined waypoint navigation is not allowed. The system must act autonomously.
- Generating point cloud from depth image: use *depth_image_proc* in http://wiki.ros.org/depth_image_proc.
- Generating occupancy Grid: use *Octomap* in http://wiki.ros.org/octomap.
- **Please ping us in case you have any questions or if you get stuck in some subtasks.**
- Use a global map as your voxel grid representation.