# Autonomous Systems Sub-Terrain Challenge


1st Alp Cankaya
*Aerospace*
*TUM*
Munich, Germany
go73juh@mytum.de

2nd Can Uludoğan
*Aerospace*
*TUM*
Munich, Germany
can.uludogan@tum.de

3rd Erdem Ekinci
*Aerospace*
*TUM*
Munich, Germany
erdem.ekinci@tum.de

4th Esra Kaplan Yılmaz
*Aerospace*
*TUM*
Munich, Germany
esra.kaplan@tum.de

5th Reha Oğuz Uslu
*Aerospace*
*TUM*
Munich, Germany
reha.uslu@tum.de



***Abstract*—In this paper we proposed a methodology to autonomously capture locations of the objects-of-interest and generate a 3D voxel-grid representation in a cave environment using a quadcopter. The quadcopter is equipped with a depth camera (Intel(R) RealSense(TM)) and a semantic camera.**



***Index Terms*—point cloud, 3-D voxel-grid, object detection, trajectory planning**


## I. Introduction

An unmanned aerial vehicle (UAV) can be basically defined as an aircraft with no human pilot or passenger onboard, which either can be operated remotely or function as an autonomous system. UAV's are utilized in various environments to perform tasks with different levels of complexity. These include roles such as reconnaissance, product deliveries, infrastructure inspections, as well as military applications.

This paper focuses on developing an autonomous UAV system that can perform mapping and navigation in GNSS-denied subterranean environment. The system is built using the Robot Operating System (ROS) and the three-dimensional simulation environment is provided with Unity 3D.

The following tasks are aimed for the system to perform:

1) Navigation of the quadcopter to the cave entrance via predefined waypoints,
2) Autonomous navigation of the quadcopter inside the cave by capturing locations of light sources (lanterns-objects of interest) and generating 3D voxel grid representation. Lanterns were modeled in the simulation as shown in Figure 3.

As can be seen in Figure 1, the UAV is initially located a distance from the entrance to the mountain cave. Until it reaches the cave entrance (shown in Figure 2), the quadcopter is controlled and navigated using predefined route markers.

The UAV processes the depth image data transferred from its mounted depth image camera to produce 3D voxel grid representation of the surroundings. Such a model enables finding unexplored frontiers and determining squential waypoints for trajectory generation and control (through transformation of corresponding data to PCL cloud and centroid computation of the frontiers). Furhermore, in this navigation phase, D* algorithm is also proposed to be utilized simultaneously.

Fig. 1: Initial Position of the quadcopter and the Simulated Environment

Fig. 2: Cave Entrance

The D* (Dynamic A*) algorithm is an efficient pathfinding and replanning method commonly used in robotics and autonomous navigation. It is an extension of the A* algorithm designed to dynamically update paths in response to changes in the environment, making it well-suited for real-time applications where obstacles or terrain can change. Basically, the algorithm functions as follows:

1) Initial Path Calculation: The algorithm first computes an optimal path from the goal to the start (backward search).
2) Dynamic Updates: If obstacles appear or the cost of certain areas changes during execution, D* efficiently updates only the affected portion of the path rather than

recalculating everything from scratch.

3) Reusing Previous Computation: Unlike A*, which recomputes the path from scratch upon environmental changes, D* intelligently reuses past computations to update the path incrementally.
4) Efficient Repairs: When an obstacle appears, the algorithm only modifies the path in affected regions and propagates the changes in a best-first search manner.

Fig. 3: Lantern Located Inside the Cave

As soon as the quadcopter reaches the cave entrance (shown in Figure 2), mode transition occurs. From this time on, the UAV navigates inside the cave by utilizing the applied afore-mentoned algorithm and determining the locations of the lanterns in this environment.

## II. Contributions

Table I represents the contributions of each team member to the development of the project.

TABLE I: Team Members and Their Contributions

| Name | Contributions |
|---|---|
| Esra Kaplan Yılmaz | Navigation (Path Planning, Trajectory Generation, Manual Flight), Vision (Point Cloud Generation, Frontier Exploration, Octomapping) |
| Alp Çankaya | Navigation (Path Planning, Trajectory Generation, Manual Flight), Vision (Point Cloud Generation, Frontier Exploration, Octomapping) |
| Erdem Ekinci | Navigation (Manual Flight), Vision (Point Cloud Generation, Lantern Detection & Logging) |
| Can Uludogan | Vision (Point Cloud Generation, Octomapping), Navigation (Path Planning, Trajectory Generation, Frontier Exploration) |
| Reha Oguz Uslu | Vision (Point Cloud Generation, Octomapping), Navigation (Trajectory Generation, Frontier Exploration) |

## III. ROS Graph and Overiew of the Nodes

The ROS Graph of the project is shown in Figure 5. The nodes we created for this project are `octamap_server`, `drone_frontier_exploration`, `depth_proc_manager`, `depth_to_cloud`, `drone_state_machine` and `planner`. The nodes can be classified into three groups according to their usage: Trajectory Generation Nodes, Sensing & Visual Detection Nodes and State Machine node.

The Trajectory Generation group is responsible for making the quadcopter reach the cave entrance, creating way points in the cave environment, and following the generated way points. The latter is for generating a point cloud, a 3-D voxel grid representation, and detecting the objects-of-interest (lanterns). The working principle and function of each node are explained in detail in the following sections.

## IV. Trajectory Generation Nodes

### A. Desired State Publisher

This node (`desired_state_pub_node`) is used for initial tests and the detection of the desired trajectory until reaching the cave. The node is given in (`desired_state_pub`) package in the source code folder. The node publishes desired state according to the keyboard inputs. The arrow keys assign translational inputs. While the key ”e” and ”f” rotates the quadcopter counterclockwise and clockwise respectively. Moreover the keys ”w” and ”s” increases and decreases the height of the quadcopter respectively. This node is used when manual inputs are necessary but not used for the autonomous operation.

### B. Trajectory Planning Node

This node retrieves waypoints and velocity parameters from an external source and continuously listens to the current UAV state, including position and velocity. It generates trajectories using `mav_trajectory_generation` and operates in two distinct modes. In the first mode, it follows waypoints defined in a parameter list stored in `config/trajectory_config.yaml`. Once a predefined trigger point is reached, it transitions to goal-based motion. The planner is stateful and reactive, meaning it dynamically adjusts based on incoming sensor data and triggers, ensuring adaptability to changing conditions.

It is also important to note that the UAV switches to goal-based (mode 2) when it gets within 1.5 meters to the trigger point, which corresponds to the final defined waypoint.

### C. Drone Frontier Exploration Node

This ROS node implements a frontier-based exploration algorithm using Octamap for mapping the environment. With this implementation, the UAV can identify frontier points which correspond to the boundaries between known and unknown space. These points can be clustered to find the largest unexplored region, and the centroid of the largest cluster can be determined as the next goal. Furthermore, this goal is published to the ”Trajectory Planning Node” for navigation.

More detailed information can be provided as follows: Initially, the node `drone_frontier_exploration` subscribes to `true_body` and `octamap_full` to obtain the

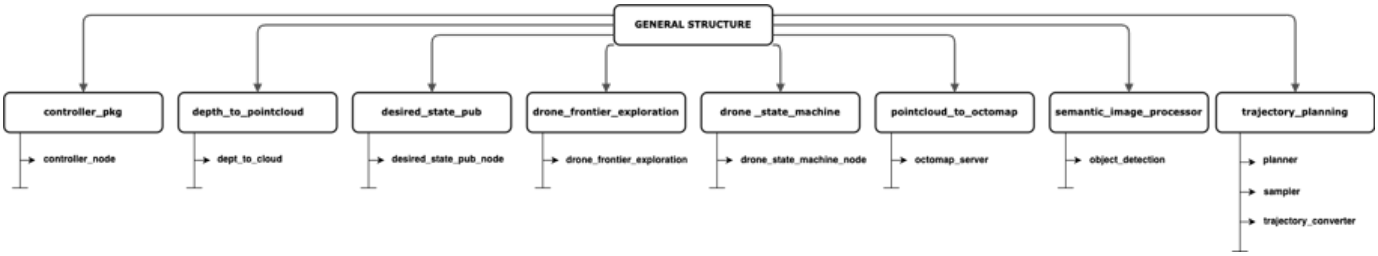


Fig. 4: General structure of the workspace

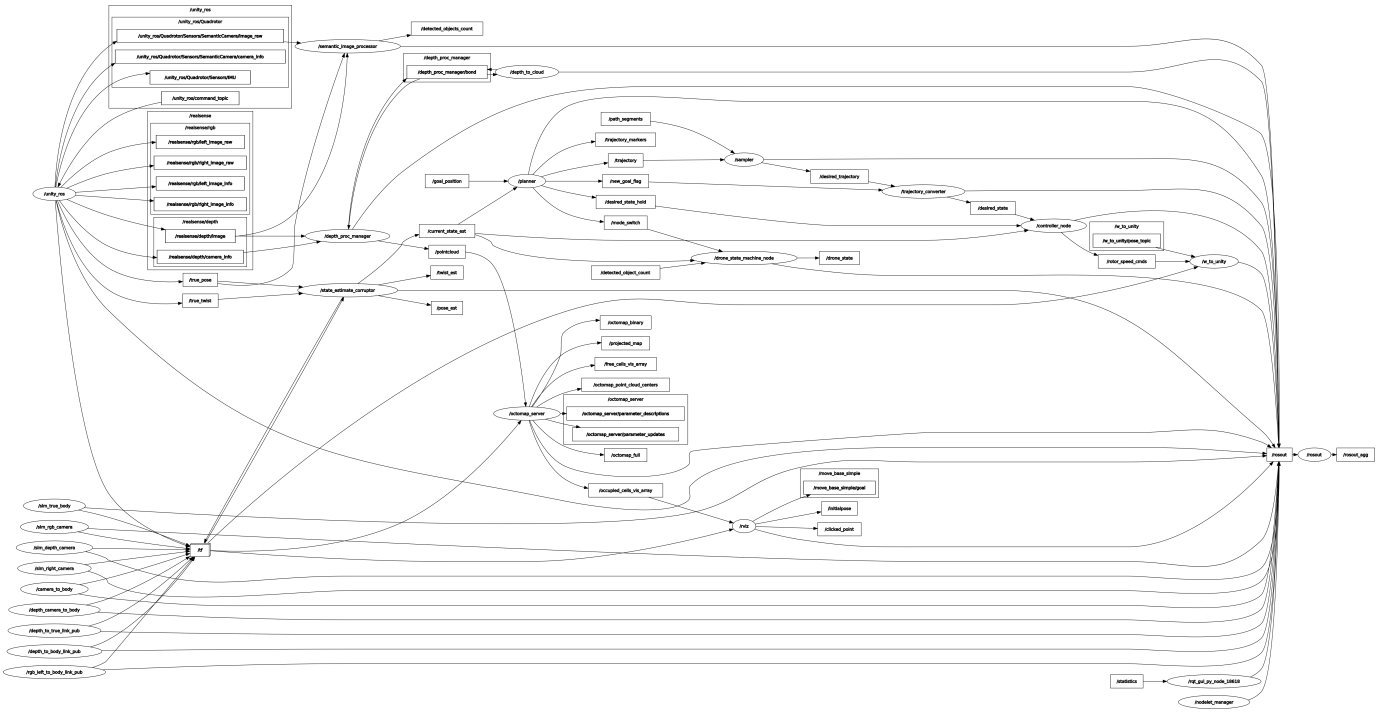

Fig. 5: The nodes, publishers, and subscribers used in the project.

drone's current position and 3D environment map, respectively. Also, it specifies `goal_position` as the data to be published. The functions mentioned below are executed periodically to perform corresponding actions:

1) `updateDronePosition()`: This function stores the UAV position. It is executed whenever the UAV moves.
2) `updateOctomap()`: This function stores a new 3D map. It is executed when the Octomap updates.
3) `performExploration()`: This function runs the **frontier search**. It is executed every 3.3 seconds.
4) `findFrontierPoints()`: This function identifies frontiers. It is executed during exploration.
5) `createFrontierPointCloud()`: This function converts frontiers to a PCL cloud. It is executed during exploration.
6) `findLargestCluster()`: This function finds the biggest unexplored area. It is executed during exploration.
7) `computeGoalCentroid()`: This function computes the center of the frontier. It is executed during exploration.
8) `publishExplorationGoal()`: This function publishes the new goal for the drone. It is executed during exploration.

## V. Sensing and Visual Detection Nodes

### A. Object Detection

This node (`object_detection`) is responsible for detecting and logging the locations of the objects-of-interest (lanterns) in the world frame. The node is included in the (`semantic_image_processor` package. For that purpose, the semantic camera, which filters out every other object except the lanterns, is utilized. The view of the semantic camera is shown in Figure 6.

The node subscribes to several topics. The Semantic Camera (`/unity_ros/Quadrotor/Sensors/SemanticCamera/image_raw`) provides an RGB image where different objects have unique colors. The Depth Camera (`/realsense/depth/image`) supplies depth information for each pixel in the scene. The True Pose topic

Fig. 6: Output of the semantic camera

(`/true_pose`) provides the quadcopter's position and orientation in the world frame.

The object is detected in the semantic image frame by identifying non-black pixels using OpenCV2 [1]. Since the depth camera is completely aligned with the semantic camera, the depth of the non-black pixels is retrieved from the corresponding pixel locations in the depth camera image. To achieve this, a subset with a maximum number of 100 pixels is selected, and the depth values of these selected pixels are averaged. Using the averaged depth along with the pixel coordinates, the location of the object in the body frame of the quadcopter is computed. This computation is performed using the given focal length of the camera and the number of pixels in the camera view.

To log each detected object separately, each object is assigned a unique ID. Any object is considered unique if its detected position is beyond a certain threshold distance from previously logged objects. The object positions, along with their unique identifiers, are saved in the (`object_positions.txt`) file, which is located in the "scripts" folder in the (`semantic_image_processor` package. Additionally, the node publishes the topic named (`detected_objects_count`. When the count is 4, all the objects-of-interest are captured. The very first lantern which is given as a reference outside the cave is neglected using a threshold. If any detected object is close to the reference object (named as "target coordinates" in the script) to a certain threshold we don't consider it.

### *B. Depth Image to Point Cloud*

The depth image output is used for generating a point cloud. The (`depth_to_cloud`) serves for this purpose. The nodelet is an instantiation of the `depth_image_proc/point_cloud_xyz` nodelet which is provided as a standard ROS package `pointcloud`. The instantiation is implemented in `depth_to_pointcloud` package in our source code folder.

### *C. Point Cloud to Octomap*

The 3-D voxel grid representation is generated using the `octomap_server_node` of the `octomap_server` package which is a standard ROS package [3]. The launch file to use this node for our project is implemented in `pointcloud_to_octomap` package which is located in the source code folder. The package makes the

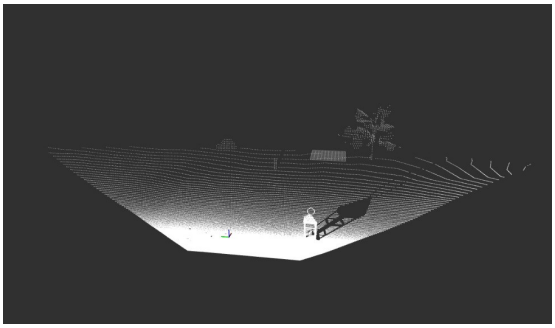

Fig. 7: RViz Visualization of the point cloud

node subscribe the `pointcloud` topic and outputs the `occupied_cells_vis_array` topic. The 3-D voxel grid representation is presented in Figure 7. The representation is visualized by selecting `occupied_cells_vis_array` topic as the MarkerArray topic on RViz.

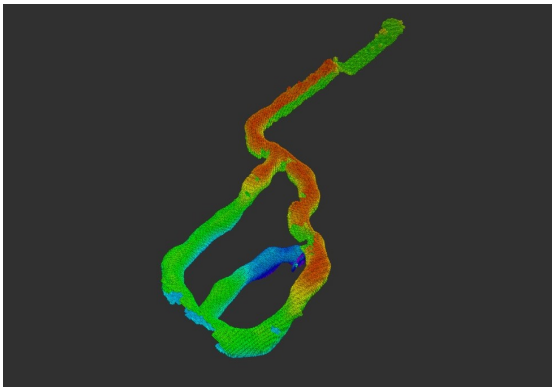

Fig. 8: 3-D voxel grid representation of the cave environment

## VI. State Machine Node

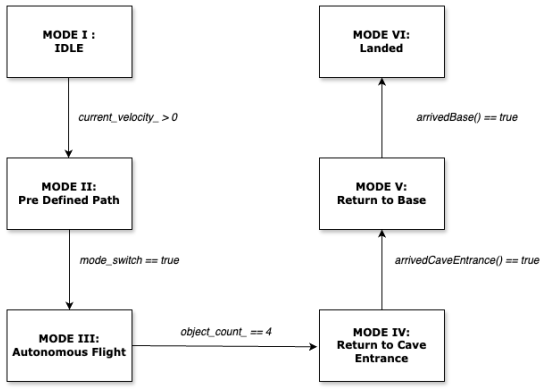


Fig. 9: State machine diagram

### *A. Mode I: Idle*

- Initial state of the drone.
- Transitions to Mode Pre-Defined Path when the velocity is greater than zero.

### B. Mode II: Pre-Defined Path

- The drone follows a pre-set trajectory with 14 waypoints, used for reaching the cave entrance.
- Transitions to Autonomous Flight when the drone reaches the cave entrance and turns mode_switch flag to true.

### C. Mode III: Autonomous Flight

- The drone performs frontier-based exploration to autonomously map the environment.
- Frontier exploration package is launched.
- Transitions to Mode IV when the identified object count is four.

### D. Mode IV: Return to Cave Entrance

- The drone navigates back to the cave entrance after completing the exploration phase.
- If the drone reaches the cave entrance, it switches to Mode V.

### E. Mode V: Return to Base

- The final state where the drone safely lands at the base.
- The drone is driven to base with a pre-defined trajectory of 14 waypoints again in the opposite direction.
- When it reaches the base, it switches to Landed Mode.

### F. Mode VI: Landed

- The final state where the drone safely lands at the base.

## VII. Controller Node

To make the UAV move along the trajectory, the geometric control model proposed by Taeyoung Lee [4] is used. The controller is included in the package named `controller_pkg`.

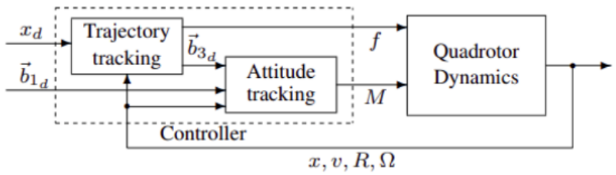


Fig. 10: Controller structure

The controller follows a prescribed trajectory of the location of the center of mass, $x_d(t)$, and the desired direction of the first body-fixed axis, $\vec{b}_{1d}(t)$. The tracking errors are calculated based on the pose and velocity of the quadcopter.

## VIII. Results

All in all, the majority of the objectives of the assignment have been achieved. The developed algorithms for mapping and navigation have demonstrated promising performance. Using depth camera data processed with OctoMap structure, the system successfully generated detailed 3D voxel grid representations of the surroundings. Moreover, the implemented frontier selection algorithm enabled the system to detect boundaries of known and unknown space, transform this data into PCL cloud, determine the biggest unexplored area and compute the corresponding centroid. Thus, the system has become able to attain further trajectory waypoints sequentially for its trajectory planner and controller nodes, and navigate autonomously within the cave. Last, The object detection algorithm detected and logged the locations of the objects-of-interest (lanterns) in the world frame by utilizing semantic image processor.

The major challenge for the team was to find a solution for the drone to navigate when it reaches to two-way junction. We observed some errors in the frontier selection algorithm at that instance, and thus further waypoints could not be generated and the drone stopped moving. Therefore, the UAV could not locate the last lantern located in the cave beyond this point. The lantern positions are presented in Table II. Utilization of D* algorithm has been proposed to overcome this obstacle, but it could not be implemented successfully due to time limitations. Solution for this problem can be designated as future studies to enhance overall performance of the developed algorithm structure.

TABLE II: Lantern Positions

| Lantern Number | Position x-y-z (m) |
|---|---|
| 1 | -598.66, -9.32, 6.00 |
| 2 | -742.92, -230.00, 2.31 |
| 3 | -1044.59, -158.45, -37.56 |
| 4* | -800.31 -282.1 -78.14 |

** The last lantern position is obtained via manual flight*

## References


[1] opencv2 - ROS Wiki, ROS Wiki. [Online]. Available: https://wiki.ros.org/opencv2. [Accessed: Mar. 2, 2025].

[2] Depth Image Processing, ROS Wiki. [Online]. Available: https://wiki.ros.org/depth_image_proc. [Accessed: Mar. 2, 2025].

[3] Octomap Server, ROS Wiki. [Online]. Available: https://wiki.ros.org/octomap_server. [Accessed: Mar. 2, 2025].

[4] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, USA, Dec. 2010, pp. 5420–5425.