Design of an Autonomous Drone for Subterranean Exploration *

*Note: This is the project report of Team 4 for the Autonomous Systems course (LRG6300 - WS23)

Baran Ozer
Technical University of Munich
Mechatronics, Robotics and Biomechanical Eng M.Sc.
baran.oezer@tum.de

Cem Kucukgenc
Technical University of Munich
Mechatronics, Robotics and Biomechanical Eng M.Sc.
cem.kucukgenc@tum.de

Erencan Aslakci
Technical University of Munich
Robotics, Cognition, Intelligence M.Sc.
ge97jed@mytum.de

Serdar Soyer
Technical University of Munich
Aerospace M.Sc.
serdar.soyer@tum.de

Hunkar Suci
Technical University of Munich
Aerospace M.Sc.
hunkar.suci@tum.de

***Abstract*—Navigating UAVs (Unmanned Air Vehicle) in GPS-denied subterranean environments is a critical study for the applications such as cave exploration and mining, where human entry may be dangerous. Nevertheless, achieving an effective 3D UAV navigation algorithm poses a significant challenge. This paper introduces a drone designed for exploring hazardous GPS-denied subterranean environments. The drone utilizes its depth camera to construct the point cloud of its surroundings. This point cloud is then processed through the OctoMap library to generate a 3D voxel grid representation of the environment. To identify the frontiers (unexplored spaces), a frontier selection algorithm is utilized to extract the voxels at the boundaries of explored and unexplored areas. For navigation, the drone employs an advanced path planning algorithm, RRT* (Rapidly-exploring Random Trees Star), which uses the identified frontiers as waypoints to plan its path. This path is then relayed to the mav_trajectory_generation package, developed by the Autonomous Systems Lab of ETH Zurich [3], to create a feasible trajectory. Finally, the trajectory is sampled and sent to the drone's controller to achieve a 3D frontier-based autonomous exploration algorithm.**

***Index Terms*—Autonomous Aerial Systems, Subterranean Exploration, 3D Path Planning, Mapping, Perception**

I. Introduction

The ability of UAVs (Unmanned Air Vehicle) to perform complex tasks has made them valuable in a variety of scenarios, including emergencies and inspections. In particular, UAVs can investigate risky or unexplored areas like caves, mines and tunnels, which assists first responders and scientists. Nevertheless, these places generally lack of GPS signals which makes it challenging for the drones to navigate and gather data. [1] [2]

Our research introduces a high-level UAV system designed specifically for navigating and mapping in GPS-denied subterranean environments without human intervention. Utilizing its depth camera, the drone constructs a detailed 3D map of its surroundings, which enables it to navigate autonomously and avoid obstacles. The proposed system has the ability to identify the lanterns inside the cave using its semantic camera and yield their locations in the environment utilizing its depth camera.

Fig. 1. The simulation environment

The depth image data from depth camera of the UAV was processed using the OctoMap package to produce a 3D voxel grid representation of the surroundings. This model helps find unexplored frontiers, which are subsequently used by RRT* (Rapidly-exploring Random Trees Star), an advanced path planning method, to securely guide the UAV across these areas. The ”mav_trajectory_generation” package, developed by the Autonomous Systems Lab of ETH Zurich [3], was used to optimize and carry out the planned path, showcasing the UAV's effective autonomous exploration and mapping of subterranean areas.

II. Task Distribution

In the development of the autonomous drone for subterranean exploration, task distribution among team members was strategically planned to leverage individual expertise and

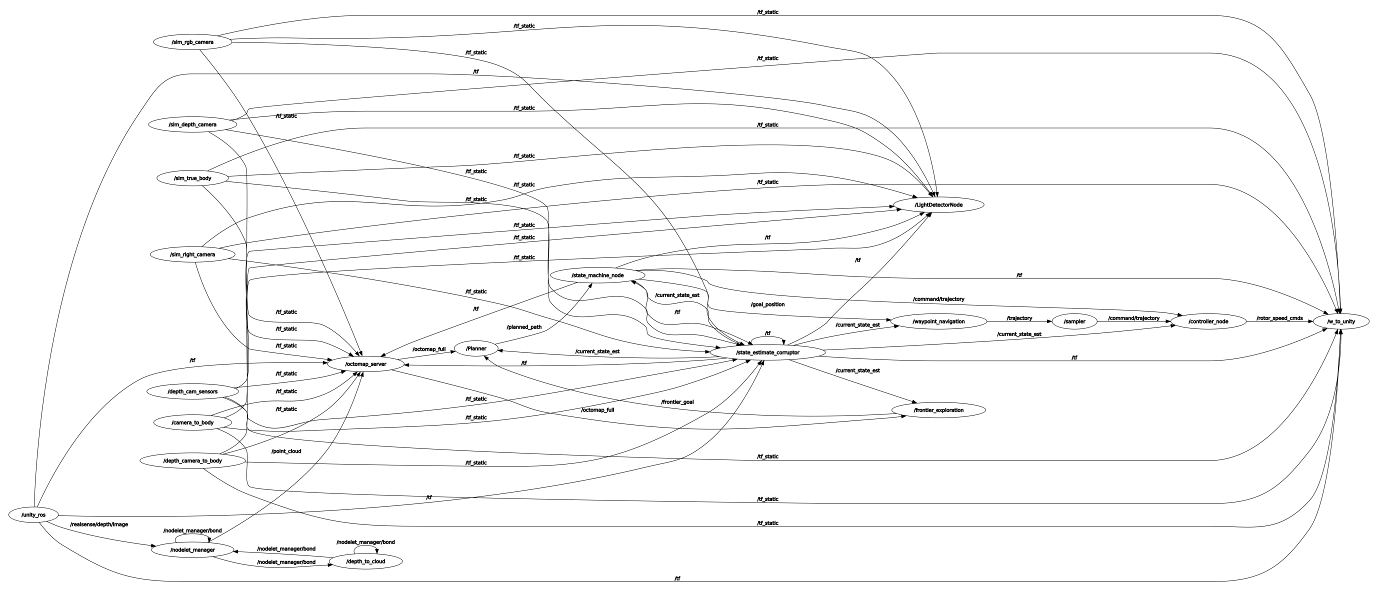

Fig. 2. The nodes, publishers, and subscribers used in the project.

TABLE I
TASK DESCRIPTIONS OF GROUP MEMBERS

Member Name	Task Description
Baran Ozer	State Machine, Navigation (Path Planning, Trajectory Generation), Vision (Point Cloud Generation, Octomapping)
Cem Kucukgenc	Navigation (Path Planning, Trajectory Generation), Vision (Point Cloud Generation, Octomapping), DevOps
Erencan Aslakci	Vision (Light Detection), Navigation (Frontier Exploration)
Hunkar Suci	Navigation (Path Planning, Frontier Exploration), Vision (Light Detection)
Serdar Soyer	Vision (Octomapping, Light Detection), Navigation (Frontier Exploration)

ensure efficient progress. The following table describes the specific contributions of each team member.

III. GENERATED PACKAGES

A. State Machine Package

The state machine package in a ROS environment is specifically tailored for orchestrating the complex behavior of drones through operational states, ensuring smooth transitions based on the drone's current position, mission objectives, and environmental interactions. It employs a sophisticated framework that leverages ROS topics for robust inter-node communication, enabling different components of the drone's system to exchange information seamlessly and react to dynamic changes in real-time. Additionally, the package utilizes ROS timers to efficiently manage periodic tasks and actions, such as regular status updates, sensor data processing, and autonomous decision-making processes, ensuring timely responses to the ever-changing demands of the drone's operational environment.

B. Navigation Package

The navigation package within a ROS framework is intricately designed to empower drone with the capability to autonomously navigate through complex environment, leveraging sophisticated path planning algorithms to chart efficient. The functionality lies the implementation of the RRT* algorithm, renowned for its efficacy in computing optimal paths that navigate the drone from its current position to a specified target location in the presence of obstacles and constraints.

C. Vision Package

The drone's perceptual capabilities by enabling the detection and tracking of objects within its operational environment. This package adeptly integrates semantic data—information that provides context about the nature and category of objects—with depth data, which offers precise measurements of distances between the drone and these objects. Such a comprehensive fusion of semantic and depth information allows for a nuanced understanding of the environment, facilitating not only the identification of objects but also the discernment of their shapes, sizes, and relative positions.

IV. GENERATED ROS NODES

A. Frontier Exploration Node

The primary goal of the Frontier Exploration node is to enable autonomous exploration of unknown environments by systematically identifying and navigating towards unexplored areas, thereby facilitating comprehensive environment mapping and situational awareness. This capability is crucial for a wide range of applications, from search and rescue operations to autonomous surveying and inspection in complex, unknown terrains.

B. Planner Node

The Planner node aims to provide a robust solution for autonomous navigation in complex 3D environments. By combining path planning with real-time collision avoidance and adaptive goal setting, it enables drones or robotic systems to navigate safely and efficiently, whether for exploration, mapping, or specific mission objectives. The integration of OMPL for planning and FCL for collision detection, along with the dynamic environment representation offered by OctoMap, exemplifies a comprehensive approach to solving navigation and planning challenges in robotics.

C. State Machine Node

The node defines a state machine for a drone navigation system. It manages the drone's flight states and goals, starting with takeoff from an initial position, then flying to predefined coordinates such as a lamp position and a cave entrance, and finally performing specific maneuvers like hovering, exploring, landing, turning, and moving forward. It subscribes to the current state and planned path topics to receive and process navigation data, publishes desired trajectories and goal positions, and uses timers to periodically update the drone's mission state. The node handles path planning by adjusting the orientation of the drone based on the direction to the next point and inserting midpoints when necessary.

D. Waypoint Navigation Node

The node executes autonomous navigation tasks based on predefined waypoints. Waypoints are specific geographic locations or coordinates that the drone is programmed to fly to as part of its mission. The waypoint navigation node orchestrates the movement of the drone from one waypoint to another, ensuring it follows the designated path accurately and safely.

E. Sampler Node

The node is developed at ETH Zurich [3] for sampling and publishing UAV trajectories. It subscribes to planned trajectory segments, converts them into a format suitable for UAV control, and then either publishes the entire trajectory at once or iteratively samples and publishes trajectory points at a set frequency. This node can also respond to a stop service call to halt the publishing of trajectory commands and can interface with a position hold service to manage UAV stability during trajectory execution. The primary functionality includes sampling trajectory points based on polynomial representations, handling timing for publishing, and integrating with the larger UAV control framework in ROS.

F. Nodelet Manager Node

Nodelets are a ROS feature designed to optimize the computational efficiency of processing data by running multiple nodes in a single process, thus avoiding the overhead associated with inter-process communication. Nodelet management; manages multiple nodelets, optimize data transfer, dynamic nodelet handling, sharing memory and communicating with other ROS nodes, utilize ROS messaging and service capabilities, and leverage existing ROS tools and libraries for development and debugging.

G. Depth to Cloud Node

The **depth_to_cloud** node, is a node that transforms depth image data from a depth camera into a 3D point cloud format. Point clouds are a collection of data points in space and are commonly used in the context of 3D perception for robots. This transformation is a key step in enabling a robot to understand its surroundings in three dimensions, which is crucial for tasks such as navigation, mapping, and object recognition. By running within the **nodelet_manager**, **depth_to_cloud** can efficiently process depth images as soon as they are available and output corresponding point clouds for immediate use by other components of the robotic system.

H. Octomap Server Node

The **octomap_server** node in a ROS environment is responsible for creating, updating, and distributing a 3D occupancy map, which provides a volumetric representation of the environment. This map is based on the OctoMap library, which employs octrees to efficiently encode both free and occupied spaces. The **octomap_server** node subscribes to 3D point clouds and integrates this data into a global 3D map.

As it receives point cloud messages, the **octomap_server** updates its internal octree representation of the environment by marking the spaces where obstacles are detected as occupied and spaces that are observed to be clear as free. This 3D map can then be used for various robotic applications, including navigation, where understanding the 3D structure of the surroundings is crucial for obstacle avoidance and path planning. The server can also respond to requests from other nodes for map data, providing them with the necessary information to make informed decisions about movement and interaction with the environment.

I. Light Detector Node

This ROS node operates as a light detection system, utilizing input from semantic and depth camera images to identify light sources in the environment. It subscribes to camera-related topics to gather semantic and depth data, along with camera calibration information. The node processes this data to locate and calculate the positions of light sources, transforming them into a global reference frame. It then publishes these positions, ensuring that it only reports new detections. This functionality allows for dynamic interaction with the environment, potentially aiding in tasks such as navigation and mapping.

V. Given ROS Nodes by Template

A. Controller Node

The ROS node is designed to control a quadrotor UAV by implementing a geometric tracking controller based on the paper by Lee, Leoky, and McClamroch [4]. It subscribes to the current and desired state topics to receive information about the UAV's position, velocity, orientation, and angular velocity, as well as the target trajectory. Using this data, it calculates the required propeller speeds to maneuver the UAV accordingly and publishes these commands. The node runs a control loop at a specified frequency to continuously adjust the UAV's motion, with tunable parameters to optimize the controller's performance for responsiveness and stability.

B. Unity Ros Node

Serving an interface between ROS and the Unity engine. Unity is a powerful cross-platform game development environment widely used for creating interactive 3D and 2D applications. When integrated with ROS, unity is used to simulate robot environments, visualize sensor data, and control robots within a visually rich and interactive context.

C. State Estimate Corruptor Node

The Node is for simulation and testing purposes, specifically to introduce errors or noise into the drone's state estimates. State estimates include the drone's position, velocity, orientation (attitude), and possibly other kinematic or dynamic states that are crucial for navigation, control, and stability. The purpose of this node is to simulate inaccuracies in state estimation that might occur due to sensor noise, environmental factors, or algorithmic limitations in a real-world scenario.

D. Sim True Body Node

The node involves in simulation environments, specifically dealing with providing accurate, simulated data about the drone's position, orientation, and possibly other physical properties, in the drone's body frame of reference. Providing ground truth data that the node simulates and publishes the drone's true position, orientation (pitch, roll, yaw) and possibly velocity, relative to a fixed world frame, but expressed in the drone's body frame. This data is considered true within the simulation context, serving as a ground truth for testing and validation purposes.

E. Sim Depth Camera Node

The Sim Depth Camera node associated with interfacing and processing data from a depth camera attached to the drone, focusing a lightweight approach. This suggests that the node is designed to efficiently handle depth data, possibly by simplifying the processing or focusing on specific tasks that require minimal computational resources.

F. Sim RGB Camera Node

Interface with an RGB camera on the drone, focusing on capturing and processing color images. The node considers that it is intended for efficient operation, possibly with a streamlined set of functionalities optimized for quick processing or reduced computational load.

G. Sim Right Camera Node

Handling the data stream from a camera mounted on the right side of the drone. The camera or its data processing is intended to be lightweight, possibly focusing on efficiency or a specific subset of image processing.

H. Camera to Body Node

For transforming camera data from the camera's coordinate frame to the drone's body frame of reference. This transformation is crucial for integrating visual information accurately with the drone's navigation and control systems. Converted camera data takes image data or spatial data captured by the camera and transforms it from the camera's coordinate system to the drone's body coordinate system. Since cameras can be mounted in various orientations or positions on the drone, this step ensures that the visual data aligns with the drone's own coordinate system for processing and interpretation.

I. Depth Camera to Body Node

The node serves the purpose of transforming depth camera data to the drone's body frame of reference. Depth cameras provide 3D spatial data by measuring the distance from the camera to objects in its field of view, which is invaluable for navigation, obstacle avoidance, and environment mapping.

J. Depth Cam Sensors Node

Handling interfacing with and processing data from a depth camera mounted on the drone. Depth cameras are essential for various applications, including navigation, obstacle avoidance, and 3D environment mapping, as they provide detailed information about the distances of objects from the camera's viewpoint.

K. W to Unity Node

The node takes data from ROS topics, such as sensor data, state estimates, or navigation paths, and converts it into a format that Unity can understand. This often involves serialization of ROS message types into byte streams or strings that can be sent over a network.

VI. Vision

A. Conversion of Depth Images to Point Clouds

The primary sensor utilized for perception within our system is a depth camera. This technology generates depth images, where the value of each pixel indicates the distance from the camera's plane to the corresponding point in the physical environment. The conversion of these two-dimensional depth images into three-dimensional point clouds is facilitated by the pinhole camera model. According to this model, the spatial relationship between the coordinates on the image plane (u, v) and their corresponding real-world coordinates (x, y, z) can be formulated as follows:

$$\begin{aligned} x &= \frac{(u - c_x)z}{f_x}, \\ y &= \frac{(v - c_y)z}{f_y}, \end{aligned} \tag{1}$$

The focal lengths of a camera along the x and y axes are represented by f_x and f_y respectively, which may differ due to factors such as lens distortion or the non-square shape of the image sensor. The optical center of the image is represented by c_x and c_y. By representing these relationships in a matrix form

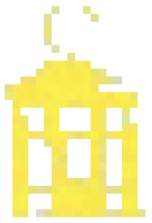

Fig. 3. Light detected by semantic camera with RGB value (4, 235, 255)

as given in Equation 2 in [6] and considering the transformation between the camera frame and the world frame, we can accurately reconstruct three-dimensional positions from two-dimensional image coordinates. The practical application of this theoretical framework is achieved through the efficient conversion of depth images into point clouds for further analysis using the **depth_image_proc** [7] package.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = z \cdot K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2}$$

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

B. Point Cloud Voxelization

To mitigate the effects of noise and potential outliers present in point cloud data, our methodology incorporates a voxel grid filter from the Point Cloud Library (PCL) [8]. This approach divides the point cloud into a grid of voxels (small cubic volumes), within which all points are aggregated into a single centroid. This voxelization technique not only cleanses the data by removing noise but also reduces its size, thereby optimizing the computational demands of subsequent mapping and spatial analysis operations.

C. Light Detection

In addition to depth information, our system leverages semantic image processing to detect specific objects, such as lights, within the environment. This is accomplished by analyzing RGB images obtained from the same depth camera and applying a color-based filter to identify pixels with the RGB value of (4, 235, 255). This specific color value is used as a marker for light sources within the scene. Upon detection of these markers, the corresponding depth information from the depth images is utilized to ascertain the precise location of these lights in the three-dimensional space. This can be formulated as given:

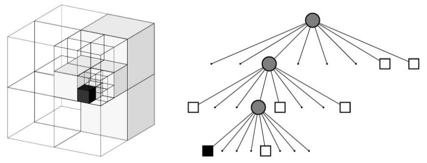

Fig. 4. Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corresponding tree representation on the right. [4]

1) Apply a color filter to the RGB image to create a binary mask where the pixels with RGB values of (4, 235, 255) are white (value 1) and all other pixels are black (value 0).
2) Use this mask to filter out the irrelevant parts of the depth image, effectively isolating the depth values of the detected light sources.
3) Convert the resulting depth image into a point cloud using the camera's intrinsic parameters, as described in the previous section.
4) Transform the point cloud from the camera frame to a global frame of reference using the ROS transformation framework (tf).
5) Calculate the centroid of the point cloud representing the detected object to determine its location in the global frame.

The mathematical formulation for the centroid calculation based on the point cloud data is given by:

$$\text{centroid} = \frac{1}{N} \sum_{i=1}^{N} P_i \tag{4}$$

where P_i represents the 3D points in the cloud that correspond to the detected light, and N is the number of these points. The centroid coordinates (x, y, z) provide the precise location of the light source in the global frame.

The object detection is then further validated to ensure that each detection is unique. This is performed by verifying that the detected object's location is a certain distance away from previously detected objects. If the object is indeed unique, its location is published as a new detection.
This synergy between semantic image processing and depth analysis allows for the accurate identification and localization of objects of interest, enhancing the system's ability to interact with and understand its surroundings effectively.

VII. Mapping

Point clouds provide the advantage of continuous space representation without the requirement for discretization, allowing a large region to be mapped. But they require a large amount of memory, and they don't have a built-in way to categorize space as free, occupied, or undefined. Octomap is used to efficiently generate probabilistic maps in three dimensions.

A. Framework for Octomap

Until maximum precision is reached, Octomap divides each volume (voxel) into eight equal-sized cubes using an octree-based structure to represent three-dimensional spaces as cubes. Figure 4 shows this procedure. Three states are distinguished for voxels: free, occupied, and unknown. In order to achieve memory efficiency, nodes whose children are all in the same state should not have their information stored.

B. Updating Probabilistic Maps

Octomap combines the sensor and occupancy models using a recursive binary Bayes filter as mentioned in [5]. The probability of a map node, given sensor data up to time t, is calculated as follows:

$$P(n \mid z_{1:t}) = \left[1+\frac{1-P(n \mid z_t)}{P(n \mid z_t)}\frac{1-P(n \mid z_{1:t-1})}{P(n \mid z_{1:t-1})}\frac{P(n)}{1-P(n)}\right]^{-1} \quad (5)$$

Here, n represents a map node, and $z_{1:t}$ is the set of sensor readings up to time t. The term $P(n \mid z_t)$ is the probability based on the current sensor reading z_t, and P_n is the initial probability, set to 0.5. The original probability form might yield values outside the [0, 1] range. To address this, we use a log-odds representation for the occupation probability, defined as:

$$l(x) = \ln\left(\frac{x}{1-x}\right) \quad (6)$$

Consequently, the log-odds of a node given sensor data up to time t can be updated incrementally as:

$$L(n \mid z_{1:t}) = L(n \mid z_{1:t-1}) + L(n \mid z_t) \quad (7)$$

Since log-odds can be converted back to probabilities, nodes store log-odds values instead of probabilities. These log-odds are infinitely additive, meaning that if a node is observed as occupied (k) times, it would require at least (k) non-occupied observations to revert its status. While suitable for static environments, this can lead to slow updates in dynamic settings. To allow for quicker adaptation, log-odds are clamped between artificial limits as follows:

$$L(n \mid z_{1:t}) = \max\left(\min\left(L(n \mid z_{1:t-1}) + L(n \mid z_t), l_{\max}\right), l_{\min}\right), \quad (8)$$

Here, $l_{\min}$ and $l_{\max}$ are the lower and upper bounds for the log-odds values. This clamping ensures that Octomap can update more responsively in dynamic environments.

C. 3D Mapping

In 3D mapping, an Octomap server node is specifically designed to analyze point clouds from the ground level upwards, employing a higher resolution to construct a detailed 3-dimensional map. This approach allows for the generation of a comprehensive 3D map that captures the intricacies of the environment, suitable for applications requiring depth and detail in the spatial representation. The final 3D map produced is showcased in Figure 5 illustrating the effectiveness of using high-resolution point cloud data for accurate 3D spatial mapping.

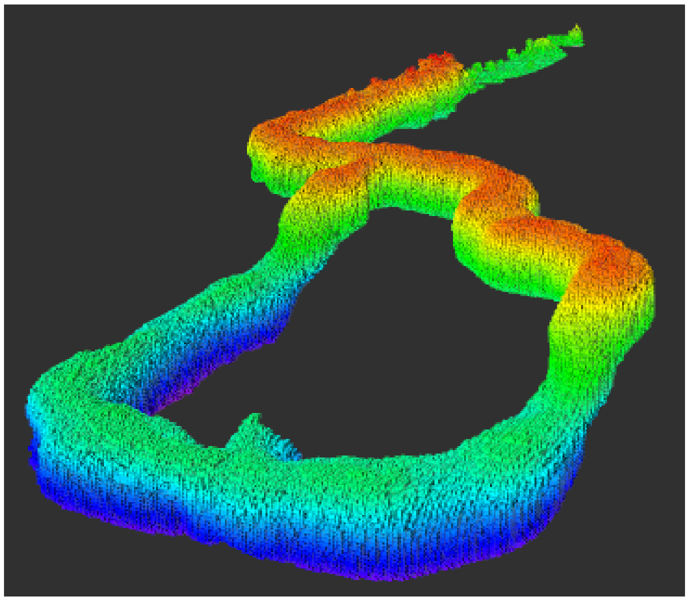

Fig. 5. The explored cave

VIII. Navigation

A. Exploration

In the realm of robotic exploration, efficiently and effectively navigating and mapping unknown environments is crucial. Frontier detection stands as a fundamental component in autonomous exploration, enabling robots to identify the boundaries between explored and unexplored areas. To leverage this functionality, the code subscribes to the "octomap_full" topic to obtain 3D map data [6]. Upon retrieval, the data is converted to an octomap::OcTree data structure, which is then traversed to perform frontier detection. Voxel Iteration is the algorithm that iterates over each voxel in the OcTree. It evaluates each voxel to ascertain if it constitutes a frontier. Following this detection, frontier voxels are clustered based on their adjacency to form coherent frontier regions. This clustering enables the determination of the largest frontier region selected for exploration.

B. Clustering with Optics

Optics is an algorithm for finding density-based clusters in spatial data. The algorithm is used with exploration algorithm together in navigation package with the aim to achieve autonomous exploration and mapping an unknown environment efficient and intelligent manner. Frontier-based exploration, where the drone moves towards the boundaries between known and unknown areas frontiers, or other heuristics designed to maximize area coverage and map fidelity. Combined goal of exploration and clustering proves intelligent path planning, robust map building and adaptive exploration.

C. Path Generation with RRT*

In navigating towards the identified goals, the system employs the RRT* algorithm [7], an advanced variant of

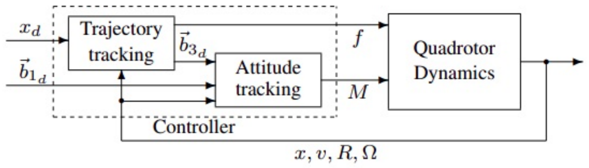

Fig. 6. Controller structure

the Rapidly-exploring Random Trees (RRT) method. RRT* iteratively constructs a tree by randomly selecting points in the space and connecting them in a manner that minimizes the path cost, considering both the length of the path and the UAV's safety. This process involves rigorous collision checking against the occupancy grid to avoid obstacles. Unlike the A* algorithm, RRT* is particularly well-suited for complex 3D environments, as it can more effectively navigate around obstacles and through narrow passages. The algorithm continuously refines paths to the goal, ensuring that the most efficient and safe route is selected over time.

D. Trajectory Planing and Execution

Following path generation, the navigation system uses the mav_trajectory_generation package to calculate a minimum snap trajectory, based on the waypoints identified by RRT*. All waypoints are used as constraints to moderate UAV speed, reducing the risk of mapping inaccuracies caused by rapid movements. This trajectory planning approach also enhances safety by ensuring that the UAV's path remains collision-free, even as it adjusts its speed and direction [12]. The UAV then executes the planned trajectory, with continuous monitoring for goal attainment, emerging obstacles, or the need to adjust its course in response to new information.

IX. Controller

For the controller for our UAV, we use the geometric controller proposed in [4]. The structure of the controller can seen in Figure 6. It consists of two tracking controllers, namely trajectory and altitude tracking.
Trajectory tracking takes a prescribed trajectory of the location of the center of mass, x_d as input and outputs the total thrust f by,

$$f = (k_1 \boldsymbol{e}_p + k_2 \boldsymbol{e}_v + mg\boldsymbol{e}_3 + m\boldsymbol{a}_d)^\top \boldsymbol{R}\boldsymbol{e}_3 \tag{9}$$

Altitude tracking takes the desired direction of the first body-fixed axis, $\vec{b}_{1_d}$ as input and outputs the total torque M by,

$$\begin{aligned} \boldsymbol{R_d} &= [b_{1_d} b_{2_d} b_{3_d}]^\top \\ \boldsymbol{e_R} &= \frac{1}{2}\left(\boldsymbol{R_d}^\top \boldsymbol{R} - \boldsymbol{R}^\top \boldsymbol{R}_d\right) \\ \boldsymbol{M} &= k_4 \boldsymbol{e}_R + k_5 \boldsymbol{e}_\omega - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} \end{aligned} \tag{10}$$

where,

$$\begin{aligned} b_{3_d} &= \frac{k_1 * \boldsymbol{e}_p + k_2 * \boldsymbol{e}_v + m * g * \boldsymbol{e}_3 + m * \boldsymbol{a}_d}{\|k_1 * \boldsymbol{e}_p + k_2 * \boldsymbol{e}_v + m * g * \boldsymbol{e}_3 + m * \boldsymbol{a}_d\|} \\ b_{1_d} &= \text{ free} \\ b_{2_d} &= b_{3_d} \times b_{1_d} \end{aligned} \tag{11}$$

X. Results

The implemented 3D UAV algorithm for navigating and mapping GPS-denied subterranean environments yielded promising results. Utilizing depth camera data processed through the OctoMap library, the system successfully constructed detailed 3D voxel grid representations of the surroundings. Through frontier selection algorithms, the UAV effectively identified unexplored areas and utilized them as waypoints for path planning using the RRT* algorithm. The integration of the mav trajectory generation package facilitated the execution of optimized paths, showcasing the system's ability to autonomously explore and map subterranean areas.

During the Frontier Exploration and Path Planning processes, we sometimes encounter a segmentation fault. We have examined the error logs and tried using smart pointers, but have not reached a solution. We were receiving this error even before patches were applied to the voids in the cave in the simulation. After the new simulation version with patching edit, we began to receive the error less frequently and reached further locations in the cave. However, some voids still exist. We therefore thought that the remaining voids might be the reason for this error. Please note that, despite the segmentation fault, when the `rosrun navigation_pkg frontier_exploration` command is executed in a seperate terminal to restart the explorer, the drone continues its mission.

XI. Summary

In summary, this project introduced a UAV system specifically designed for navigating and mapping GPS-denied subterranean environments autonomously. By leveraging depth camera data processing and advanced path planning algorithms, the system demonstrated the capability to construct detailed 3D maps and navigate through challenging terrains. The successful integration of the mav trajectory generation package further enhanced the system's autonomy and efficiency in exploration tasks. Overall, this research contributes to the advancement of autonomous aerial systems for subterranean exploration, with potential applications in cave exploration, mining, and other hazardous environments.

References

[1] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Autonomous Navigation and Mapping in GPS-denied Environments Using a UAV," in Journal of Field Robotics, Vol. 29, No. 5, pp. 911-928, 2012.
[2] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar, "Fast, Autonomous Flight in GPS-denied and Cluttered Environments," in Journal of Field Robotics, Vol. 35, No. 1, pp. 101-120, 2018.
[3] Autonomous Systems Lab, ETH Zurich, "mav_trajectory_generation: A planning framework for MAVs," https://github.com/ethz-asl/mav_trajectory_generation, 2020.
[4] Lee, Taeyoung, Melvin Leok, and N. Harris McClamroch. "Geometric tracking control of a quadrotor UAV on SE (3)." 49th IEEE conference on decision and control (CDC). IEEE, 2010.
[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," Autonomous Robots, 2013.

[6] H. Moravec and A. Elfes, ”High resolution maps from wide angle sonar,” Proceedings. 1985 IEEE International Conference on Robotics and Automation, 1985.

[7] R. Hartley and A. Zisserman, ”Multiple View Geometry in Computer Vision,” Cambridge University Press, 2003.

[8] Willow Garage (2008) image pipeline [Source Code]. https://github.com/ros-perception/image pipeline/tree/noetic.

[9] R. B. Rusu and S. Cousins, ”3D is here: Point Cloud Library (PCL),” 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1-4, doi: 10.1109/ICRA.2011.5980567.

[10] OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, Autonomous Robots, Volume 34, Issue 3, pp 189–206, 2013.

[11] RRT*: Optimal Sampling-based Path Planning. S. Karaman and E. Frazzoli, International Journal of Robotics Research, Volume 30, Issue 7, pp 846–894, 2011.

[12] D. Fox, W. Burgard, and S. Thrun, ”The Dynamic Window Approach to Collision Avoidance,” IEEE Robotics & Automation Magazine, Vol. 4, No. 1, pp. 23-33, March 1997.