Autonomous UAV Exploration: Navigating Sub-Terrain Challange

Xin, Yutong
Technical University of Munich
yutong.xin@tum.de

Wang, Zihao
Technical University of Munich
z.h.wang@tum.de

Abstract

This paper presents our project on autonomous UAV navigation aimed at addressing the complexities of cave exploration. We elucidate our methodologies for generating occupancy grids from point clouds, creating a global map for navigation, and selecting as well as generating frontiers and paths for exploration. We also discuss the lantern identification and localization, which serve as primary targets within the cave. Despite the limitations posed by a two-person team, we have met most of the targets, including environmental perception, path planning, and target detection.

1. Introduction

This document is a project for the course autonomous system 23-24 WS and the main task is to implement autonomous navigation for a UAV. The scenario is a UAV and a cave that is some distance away. The UAV needs to reach the entrance of a cave, enter the cave, and navigate autonomously in the cave, avoid obstacles, and find and locate the four lanterns present in the cave. The group members are: Yutong Xin and Zihao Wang. The specific division of tasks is as follows:

Yutong Xin Environmental Perception, Path Planning, State Machine, Paper Writing

Zihao Wang Trajectory Tracking, Lantern Detection, State Machine, Paper Writing

2. Environmental Perception

In this section, the information from the depth camera is utilized. Initially, the depth camera data is converted into point cloud information. Subsequently, the point cloud data is processed through the octomap server to transform it into grid information, thereby generating a three-dimensional occupancy grid map of the environment. To facilitate more efficient and rapid path planning, a two-dimensional projection map is extracted within the altitude range of the UAV.

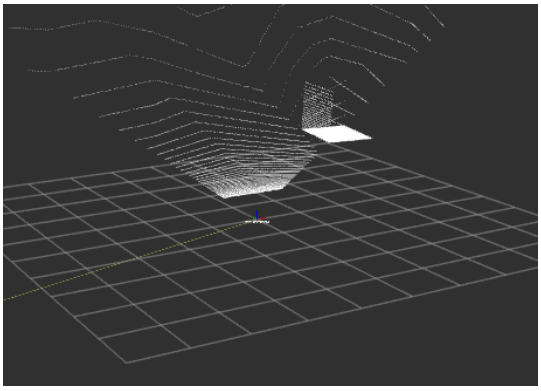

Figure 1: Original point cloud

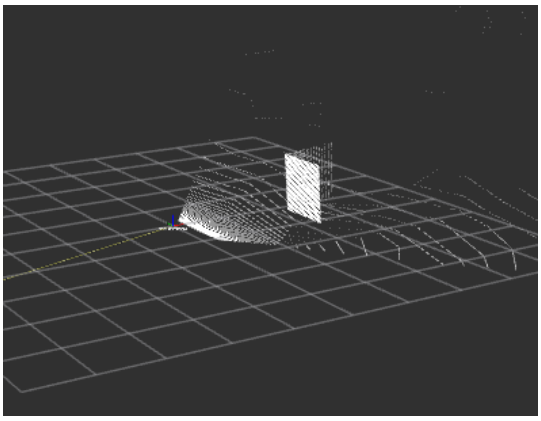

Figure 2: Point cloud after transformation

2.1. Processing of Depth Camera Data

To obtain a global map, it is necessary to first convert depth information into point cloud data. Utilizing the depth_image_proc package [2], the depth_image_procpoint_cloud_xyz nodelet plugin is employed to convert the depth information, which will then be published on the topic *points*. It is important to

note that due to the rotational discrepancy between the camera coordinate system and the world coordinate system, a transformation needs to be applied to the obtained point cloud data to correctly publish them in the world coordinate system, the difference is shown in Fig.1 and Fig.2.

Next, we will generate a grid map using the octomap_server[1] package. OctoMap is a three-dimensional mapping tool based on an octree structure. It provides a complete 3D representation including free space and unmapped areas. Moreover, sensor data based on occupancy grids can be fused and updated over multiple measurements. In our project, we can generate environment maps by subscribing to octomap_full and octomap_binary published by the octomap_server package. This will facilitate autonomous obstacle avoidance and trajectory tracking for the drone.

The occupancy grid is shown in Fig3.

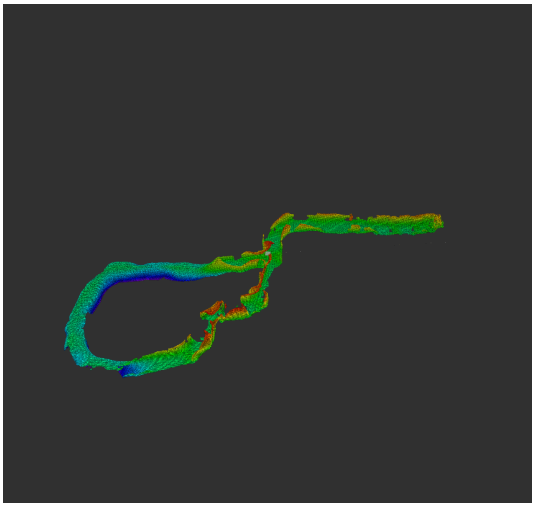

Figure 3: Occupancy gird

2.2. Projected Map Generation

To simplify the problem, we try to transform the three-dimensional map into a two-dimensional representation for autonomous navigation of the UAV. To achieve this, we filter the grid map at the UAV's own height, resulting in a two-dimensional projected map. This map depicts obstacles surrounding the aircraft while maintaining a constant flight altitude. This approach enables the aircraft to fly as far as possible at a certain height, facilitating autonomous navigation.

Fig.4 shows the global projected map at the UAV's height, it represents the boundaries of the UAV throughout the whole journey.

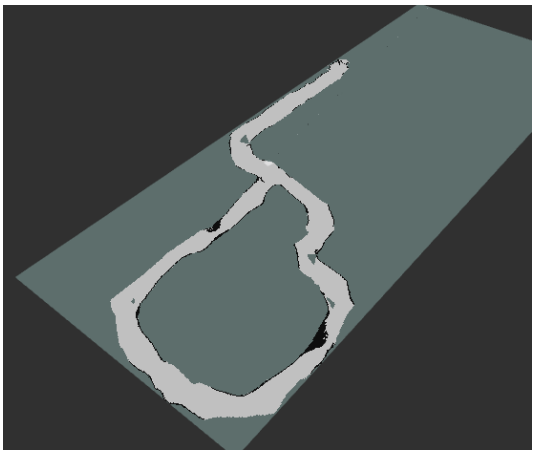

Figure 4: Global projected map

3. Path Planning

As we need to explore the entire cave, locate four lanterns, and are unaware of the final destination for the drone, we must continuously select suitable local targets during the exploration process to guide the UAV forward.

In this section, we identify the goal points for the drone locally and generate a path leading to these goals.

3.1. Local Goal Selection

As mentioned in section 2.2, we employ a two-dimensional projected map instead of a three-dimensional grid map, our focus is solely on determining whether there are suitable goal points within the UAV's altitude plane for navigation.

Hereby the algorithm is chosen as follows: for all non-obstacle points within the drone's field of view, calculate their distances to the UAV and assess their accessibility. Among all reachable points, select the one farthest from the UAV as the local goal point.

The complete algorithm is outlined as Alg.3.1.

Algorithm 1 Local Goal Selection

Input: $ProjectedMap$
Input: $MaxDistance =$ INF_MIN
for each $Point \in Points$ **do**
  if $Point.Value = 0$ **then**
    $Distance \leftarrow DistanceToUAV(Point)$
    if $MaxDistance < Distance$ **then**
      $MaxDistance = Distance$
      $LocalGoal \leftarrow Point$
    end if
  end if
end for

3.2. Path Generation

In pathfinding algorithms, commonly used options include the Dijkstra algorithm, RRT algorithm, and A algorithm. For this project, the A algorithm has been chosen for the following reasons:

- Maintains a relatively small search space during exploration, ensuring high efficiency in pathfinding.
- Higher likelihood of quickly finding the shortest path.
- Performs better on a simple two-dimensional map.

The A algorithm utilizes an estimation function (heuristic) as a criterion (Func.1) to search through all possible reachable nodes, which is shown in Alg.2 [4]. The path generation is completed iteratively by adding points from the open list to the closed list.

$$f(x) = g(x) + h(x) \tag{1}$$

Where $g(x)$ represents the cost incurred so far to reach the current node, and $h(x)$ represents the estimated cost from the current node to the target node.

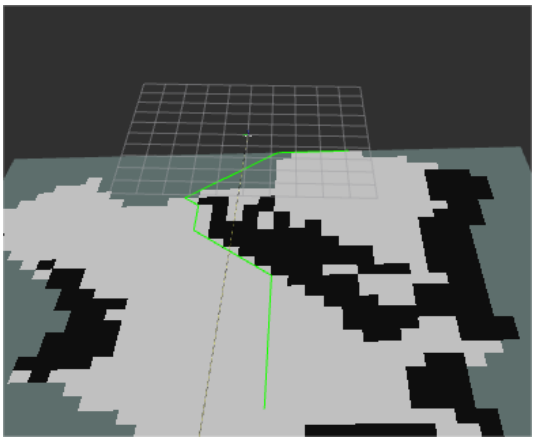

Figure 5: local path Planner

4. Trajectory Tracking

Given preselected target points and a path comprised of numerous points, the subsequent step pertains to the tracking and following of these target points. Within the exploration module, pertinent information is subscribed to, with the path and goal perpetually updated to reflect the most current routing guidance. Consequently, a global variable `current path` are initialized. The current path's local points are ingested by setting an index, thereafter assessing the proximity to these local points within the path. Should the distance be sufficiently close, the index is incremented to acquire new local points. Upon reaching the target point, a brief rotation of the aerial vehicle is enacted to survey the surrounding environment of the new location, subsequently updating the `current path` before entering the next cycle to continue the tracking process.

Thus, several states are established in the exploring process: *Initialize*, *MoveToGoal*, and *Observing*.

Algorithm 2 A* Algorithm pseudocode

Input: $Start, Goal$
while $OpenSet \neq \Phi$ **do**
  $CurNode \leftarrow$ EXTRACT-MIN-F($OpenSet$)
  if $CurNode = Goal$ **then**
    return $BestPath$
  end if
  for each $NeighborNode \in CurNode$ **do**
    if $NeighborNode \in ClosedSet$ **then**
      Continue;
    else$NeighborNode \in OpenSet$
      $CalculateF$
      $Node.Parent \leftarrow CurNode$
      $OpenSet \leftarrow Node$
    end if
  end for

5. State Machine

We split the whole process into two phases: flying to the cave and cave exploration.

5.1 Move To Cave

Note that our state machine is based on the exploration process starting from the cave entrance. Prior to entering the cave exploration state, it is necessary to move to the entrance of the cave. The movement from the initial point to the cave entrance follows: first, a slight elevation is initiated to prevent collisions with the ground that could affect the motion; then, the cave entrance coordinates are set to $[-380, 10, 14.5]$, the total flight time is set to 30 seconds, and the expected position to be reached at any given time, referred to as the *desired_pose*, is calculated. Upon reaching the entrance, the UAV rotates a full circle to completely scan the entire cave entrance, which facilitates the correct continuation of the subsequent exploration.

5.2 Cave Exploration

For the cave exploration phase, the status of the UAV will be divided into eight stages, namely *Initial, FollowPath, GoUp, GoDown, Turn, Counter, DetectLantern* and Finish.

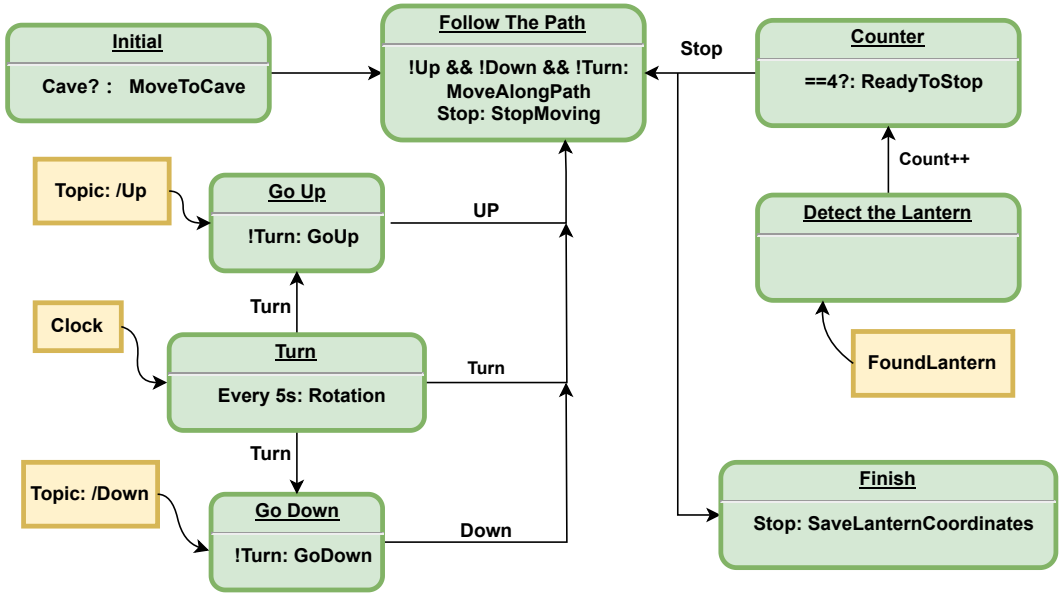

Figure 6: Workflow of state machine

By controlling the UAV through the state machine, shows in Figure 6, it will be guided to smoothly follow the prescribed path.

The initial state values are first set, and by subscribing to the *path*, the UAV can obtain the latest local target point. Since our target points are selected on the two-dimensional plane where the UAV is located, this involves movement along the z-axis. The UAV subscribes to the *up* and *down* topics to determine whether to ascend or descend. This signal originates from the detection of the distance to the walls above and below the UAV. Moreover, a timer is set to remind the UAV to rotate every 10 seconds to more comprehensively explore the cave.

The states *FollowPath, GoUp, GoDown*, and *Turn* follow the relationship: *FollowPath* is the general state, when the Boolean value controlling *Turn* is 1, *Turn* is prioritized; at this time, the UAV records its current position and rotates in place. After completing the rotation, it continues with *FollowPath*. When rotation is not necessary and there is a Boolean value of 1 for Up or Down, the UAV enters one of these two states to ascend or descend.

Last but but least, the LanternDetector remains active throughout the exploring process. When a target is detected, it returns the coordinates of the lantern. A threshold is set to address the interference caused by slight variations in the lantern position returned each time, ensuring that nearby lantern coordinates are not considered as new lanterns. The exploration ends when all four lanterns have been discovered.

6. Target Localization

There are also quests to find objectives during the cave exploration. The goal is four lanterns scattered throughout the cave. The localization of them is achieved through a sophisticated process implemented in the *lanternDetector* class. The entire process is delineated into three main stages, detailed below, starting with the acquisition of an image that has undergone segmentation from a semantic camera. This is followed by the extraction of the mask and overlaying this mask onto the depth image to ascertain the lantern's depth and positional information. Given that these coordinates are initially situated within the camera's pixel coordinate system, a conversion into the camera coordinate system is necessitated, subsequently requiring a transformation into the world coordinate system to procure the positional information.

6.1 Acquisition of Information in Pixel Coordinate System

The first stage involves acquiring images that have been segmented by a semantic camera. After subscribing to "SemanticCamera" and receiving the segmented images, the program extracts a mask highlighting the region occupied by the lantern. The target is considered detected when the number of non-black pixels in this mask exceeds the $Pixelsthreshold$. This mask is saved and then applied to the acquired depth image, enabling the algorithm to determine the depth and position of the lantern's points relative to the camera's pixel coordinate system. This step is crucial

as it lays the foundation for further transformations required for target localization.

6.2 Coordinate Transformation

The depth and position information obtained from the previous step are in the pixel coordinate system and require a transformation to the camera coordinate system. The relationship between them is shown in Fig 7 [3]. According to

Figure 7: Coordinate transformation.

this, the transformation between the pixel coordinate system and the camera coordinate system satisfies:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} (u - c_x)d/f_x \\ (v - c_y)d/f_y \\ d \end{bmatrix} \tag{2}$$

where (u, v) are the coordinates in the pixel coordinate system, (x_c, y_c, z_c) are the coordinates in the camera coordinate system, d is the depth information, and f_x, f_y, c_x, c_y are camera intrinsic parameters, which can be obtained by subscribing the camera information. A The intrinsic matrix $\mathbf{K}$ is defined as:

$$\mathbf{K} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

where $f_x = 120$, $f_y = 120$, $c_x = 160$, and $c_y = 120$ here.

However, it is observed that for the standard pixel-to-camera coordinate transformation, the origin of the pixel coordinate system is located at the top-left corner of the image, with the camera's coordinate system's z-axis perpendicular and pointing outward from the camera, the x-axis pointing to the right, and the y-axis pointing down. This standard camera coordinate system does not align with the camera coordinate system in use here. Our actual camera coordinate system's x-axis coincides with the standard camera's z-axis, the y-axis points in the negative direction of the standard camera's x-axis, and the z-axis points in the negative direction of the standard camera's y-axis. Thus, the formula here is transformed as follows:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} d \\ -(u - c_x)d/f_x \\ -(v - c_y)d/f_y \end{bmatrix} \tag{4}$$

Accordingly, we transform all obtained pixel points using this formula to get their positions in the camera coordinate system (i.e., the positions of various points on the lantern relative to the camera). Then, we take the mean of these pixel points as the position of the lantern and obtain $x_{c,mean}, y_{c,mean}, z_{c,mean}$.

Subsequently, we obtain the transformation between the world coordinate system tf and the "Quadrotor/DepthCamera" by subscribing to a `tf_listener`. The final position of the observed target is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = tf \begin{bmatrix} x_{c,mean} \\ y_{c,mean} \\ z_{c,mean} \end{bmatrix} \tag{5}$$

6.3 Validate by the Test Lantern

There is a lantern could be used as a test at the starting point of the environment, and we test and output the position of this lantern to verify the correctness of the acquisition of the target lantern for this session. The position coordinates published from the node were [(-59.415594, 0.812031, 6.318246)], which was found to coincide with the actual position of the lantern.

6.4 Lantern Detection Results

We successfully found three out of the four targets.

Table 1: Lantern Position

Name	Position
Lantern 1	$(-600.5, -9.4, 4.2)$
Lantern 2	$(-1044.9, -158.3, -38.2)$
Lantern 3	$(-742.7, -229.3, 1.5)$
Lantern 4	-

7. Result

To sum up, we have completed a range of tasks including the retrieval of occupancy grids from point clouds, the creation of a global map, the selection and generation of frontiers and paths, and the choice of the control algorithm. But we still have one lantern that we haven't found. This may be due to the fact that we are only concerned with maps in the

two-dimensional plane of the vehicle and ignore turnoffs. We should probably consider building the map in the whole 3D plane, to obtain more comprehensive map information.

References

[1] https://github.com/OctoMap/octomap.

[2] depth_image_proc. http://wiki.ros.org/depth_image_proc. Accessed: 12-March-2024.

[3] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[4] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.