Autonomous Navigation and Mapping with a UAV

*Note: project report of lecture Autonomous System by Professor Markus Ryll at Technical University Munich

1st Li, Yu   2nd Huang, Sheng Kai   3rd Chou, Yu Chen   4th Liao, Hung Yueh

***Abstract*—This report introduces how the autonomous navigation and mapping with an UAV (unmanned aerial vehicle) is achieved. A system is built to simulate an autonomous UAV exploring, navigating, and mapping autonomously in an unknown, enclosed and static environment. The simulation is conducted in Unity, while the whole system is powered by ROS (Robot Operating System). The procedures include computer vision, probabilistic 3D mapping, path planning algorithm and quadrotor motion control.**

I. Introduction

In the simulation, the quadrotor is controlled by the speed of its four rotors, which are regulated through a controller. There are four sensors on the quadrotor. The IMU (inertial measurement unit) ,two RGB cameras and one depth camera. A picture of the quadrotor is shown in Fig. 1.

Goal of the project is to reconstruct a comprehensive 3D Reconstruction of the surroundings. This project simplified the challenge and aimed on a moderate goal. Thus, the quadrotor is set to fly at a constant altitude based on preferred grid-based path planning approach. Nevertheless, a 3D map is constructed within the range of the cameras. Base on this preset condition, the following sections will describe the project from five aspects: simulation, vision, mapping, path planning and control. A clear overview of the operation between parts is given by Fig. 2.

Fig. 1. The quadrotor

II. Simulation

The simulation is carried out in Unity. The map used in this project simulate an industrial park with transportation streets between fabric halls (Fig. 3). The quadrotor in Unity environment is equipped with sensor script and receives sensor date such that it communicates with ROS host via TCP server.

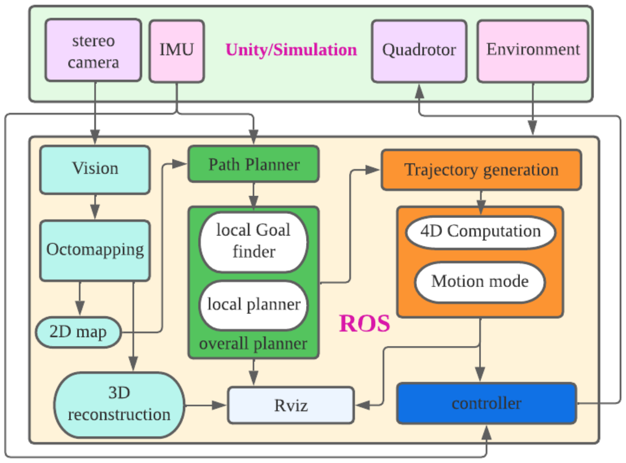

Fig. 2. System Structure

The data is then processed for the use of other subsystems. There are two types of simulated sensor data: the IMU detects the motion of the quadrotor and the cameras receive real-time image of the environment.

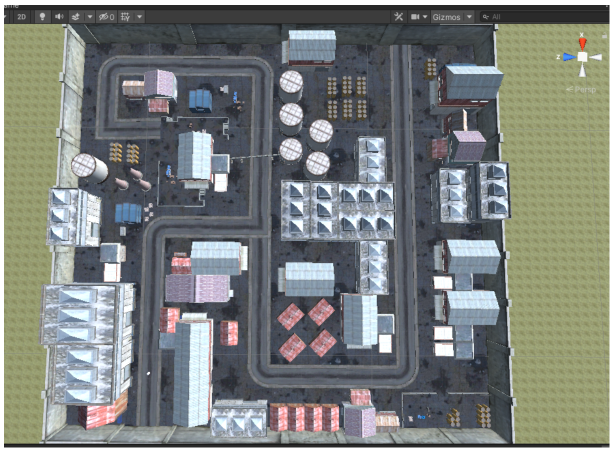
Fig. 3. Simulation environment constructed in Unity

A. *IMU for odometry*

The term ”odometry” stands for the process to use motion sensors to calculate the relative position in respect of the initial state. In the real world, position estimated by IMU has accumulative error, which should be calibrated by other measures like GPS [1] . In our simulation, the position information is directly utilized with tiny artificial defined noise.

When transmitting the position information, the Unity node publishes the position in the message type "nav_msgs :: Odometry" to a ROS topic.

B. Camera and image processing

The raw data received from the camera is processed by several ROS nodes before it is reconstructed in a map. The raw data is encoded and published from Unity. It is then converted into a point cloud via a depth image processor. Next, the point cloud is converted into a volumetric occupancy grid map via octomap, a probabilistic 3D mapping framework [2].

In Unity, the depth camera is simulated by a program that publishes the virtual distances calculated from the quadrotor to obstacles in the ROS message type "sensor_msgs::Image". It doesn't specify which 3D object detection technologies is physically implemented. The next chapter will introduce the stereo cameras, which are commonly used in depth detection.

III. Vision

A. Stereo camera

A stereo camera generates a point clouds as followed: a stereo camera captures two images of the same scene from slightly different perspectives. Following, corresponding features in both images are matched through an image recognition algorithm. Then, disparities (differences in pixel positions) between matched features are estimated. With disparities of these matched features, 3D positions of these features are triagulated, resulting in a point cloud.

Reference [4] explains the mathematical methods to obtain a point in 3D space given its image in two views and the camera matrices of those views. It suggests a minimization method to obtain a 3D point in presence of measurement errors. (Fig. 4)

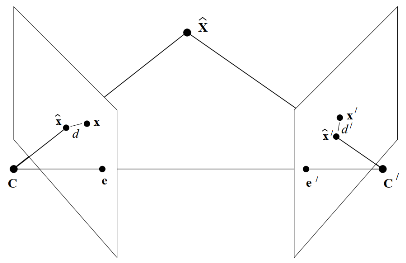

Fig. 4. Minimization of geometric error. The estimated 3-space point $\hat{X}$ projects to the two images at $\hat{x}$, $\hat{x}'$. The corresponding image points $\hat{x}$, $\hat{x}'$ satisfy the epipolar constraint, unlike the measured points x and x. The point $\hat{X}$ is chosen so that the reprojection error $d2 + d2$ is minimized. [4]

B. Voxelization

Point cloud obtained directly from a depth camera is not feasible for post-processing due to noises. A voxel grid filter provided by PCL (point cloud library) [6] is applied to remove noises and reduce the computational load in mapping. Through PCL, the input data of message type "sensor_msgs::PointCloud2" is converted into "pcl::PointCloud", which represent a grid of cubic volumes of equal size.

C. Uncertainty of reconstruction

Measurement error in camera causes a probability distribution of measured points. This leads to uncertainty by reconstructing the 3D points. The accuracy of reconstruction is determined through the angle between the rays, as shown in Fig. 5. Uncertainty of the points' location increases as the rays become more parallel [4]. The uncertainty of reconstruction is treated with a probabilistic updating method under the Octomap framework introduced in the following chapter.

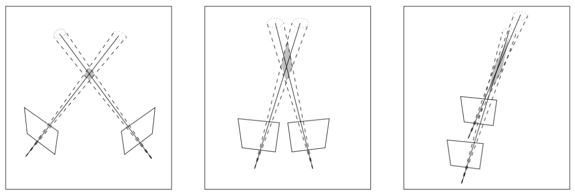

Fig. 5. The shaded region in each case illustrates the shape of the uncertainty region, which depends on the angle between the rays. [4]

IV. Mapping

To navigate the quadrotor autonomously requires a map presenting the real-time updated environment that can be analyzed by a path planner. Point cloud is not feasible in this case due to it's unlimited memory consumption, lack of updating method and inability to differentiate unexplored or free spaces. To cope with these challenges, the Octomap framework introduced in [2] is implemented in this project.

A. Octree

Octomap store the map in a recursively hierarchical data structure called Octree. Octree divides a 3D cube into 8 equal subdivisions. Each cube is represented by a node in Fig. 6, which is assigned to either one of the states: free, occupied or unknown. This division can be carried out recursively to achieve wished resolution.

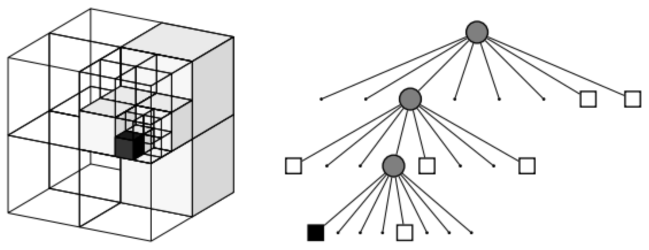

Fig. 6. Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corresponding tree representation on the right. [2]

B. Probabilistic updating

To decides weather a node is occupied, Octamap uses a update formula introduced by [7]. Every voxel is assigned to a probability value, which is calculated from the previous probability and the newest sensor data:

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

Setting the initial value to $P(n)=0.5$ and using the logit notation, Equation (1) can be rewritten as:

$$L(n \mid z_{1:t})=L\left(n \mid z_{1:t-1}\right)+L\left(n \mid z_t\right) \quad (2)$$

with

$$L(n)=log\left[\frac{P(n)}{1-P(n)}\right] \quad (3)$$

By setting a threshold to the probabilistic value, a node is determined to be occupied or free.

V. Path Planner

The autonomous navigation process can be divided into path planning and trajectory generation. The path planner determine which way-points the quadrotor should pass by, while the trajectory generator further define the motion along the path, such as the velocities and accelerations at the way-points. This section focuses on the path planner. The trajectory generator will be introduced in the next section.

The path planner is built under following assumption: the unknown world is enclosed by high wall and all objects are static. The to-be-explored area is predefined, such that the size of the area is known. The current position o the quadrotor in the world coordinate is also known, as stated in II. Besides, the path is exclusively planned in the known area.

The path planner is constructed of two layers of algorithm, named as ”Overall Layer Planner” and ”Local Goal Finder”. They are explained separately in the subsections. Their correlation is shown in the structure of the planner in Fig.7

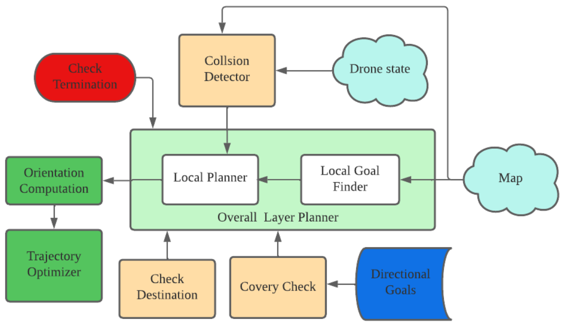

Fig. 7. Overall structure of planner mechanism

A. Global Goal Point Generator

The the overall path planning is inspired by Voronoi map [8]. As of in [8], the known area is artificially separated into several sub-area with a goal point within it. The quadrotor is programmed to visit these goal points in a predefined sequence.

As long as the goal point has not yet been included into the explored area, the local goal finder would guide the quadrotor to explore and approach the goal point. The approach to the goal point is an iterative process, which is stated in the next two subsections. Once the goal point is explored, the quadrotor will head to the next goal. The exploration will end when a certain percentage of the whole world is explored.

The global goal generator algorithm is shown in Alg.1.

Algorithm 1 Global Goal Point Generator

```
1: $DirectionGoalsInit();$
2: while $unkonwn/wholemap \leq threshold$ do
2:     for each $Goal \in \mathcal{G}oals$ do
3:     if Goal $is\ not$ visited then
4:         $Path \leftarrow LocalPlanner(Goal)$
5:         return $Path$
6:         break
7:     end if
8:     $Trajectory \leftarrow TrajectoryOptimizer(Path)$
8:     end for
9: end while=0
```

B. Local Goal Point Finder

When trying to approach the goal point, an intermediate local goal point has to be determined first. A global goal point can not be directly used in the path planner since it lays in unknown area. Thus, candidates of local goal points are listed and assessed.

A local goal point is a middle point of a edge. Such edge is defined as the boundary between known and unknown area. An edge is segmented by an obstacle. Among all candidates, the one that stands nearest to the (current) global goal point would be checked with Dijkstra algorithm to examine it's accessibility. If the local goal point is proved to be feasible, a path will be generated from the current position. (8)

The local goal finder algorithm is shown in Alg.2.

C. Path Planner

Having the local goal point, a path from current position to the local goal point is generated with Dijkstra algorithm. To avoid collision with obstacles, extra buffer space between the obstacles and the path is reserved.

An additional break condition is set while the quadrotor is flying along the path. As stated in IV-B, it is possible that a free space is judged to become an obstacle because of new sensor data. In this case, the path is leading into an newly emerged obstacle. A regeneration of path would be required at this point.

The path planner algorithm is shown in Alg.3.

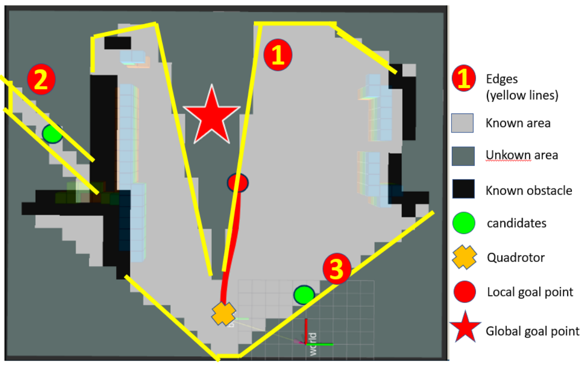

Fig. 8. this example shows three edges, three candidates for local goal point, and the chosen local goal point. Candidate nearest to global goal point is chosen

Algorithm 2 Local Goal Point Finder

```
Input: $GridMap$
Input: $CurrentDirection$
1: $Boundaries \leftarrow BoundaryFinder(Map)$
1: for each $Boundary \in Boundaries$ do
2:     $Segments \leftarrow SegmentsSelector(Boundary)$
3:     $LocalGoals \leftarrow MiddlePoint(Segments)$
3: end for
4: return $LocalGoals$
4: for each $Each \in LocalGoals$ do
5:     $Finals \leftarrow SortDistanceTo(CurrentDirection)$
6: return $Finals$
```

VI. Trajectory Generator

The path generated by path planner consists of a sequence of way-points. These watpoints are given to trajectory generator. A trajectory describes the motion of the quadrotor. It tells the quadrotor to pass by way-points at determined velocity and acceleration on certain timestamps. The trajectory is given to controller, which finally send the torque signal to the four propellers.

A. Trajectory

A minimum acceleration optimization method is applied in this project. Velocity and acceleration at way-points are chosen, so that the acceleration before and after way-points is smooth.

Algorithm 3 LocalPlanner / Modified Dijkstra

```
Input: $Goals \leftarrow Finals$
0: for each $Goal \in Goals$ do
1: $path \leftarrow Dijkstra(Goal)$
2: if $path$ not $empty$ then
3:         $ModifiedPath \leftarrow Modify(path)$
4:     break
5: end if
6: return $ModifiedPath$
```

In order to expand the explored area, the quadrotor should do a 360° rotation after it arrived the local goal point. Afterward, the next path will be planned according to the newly updated map.

To do this trick, the trajectory planner send a series of way-points with the same position but increasing yaw angles.

VII. Geometric Control

The quadrotor is controlled with a geometric control model proposed in [9]. The target trajectory from the trajectory planner is compared with the actual pose. Propellers are set to adjust the pose and position according to the difference.

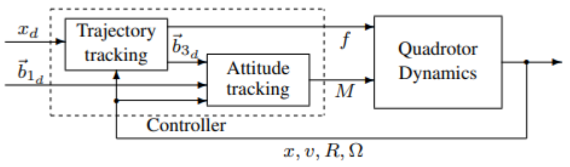

Fig. 9. Controller structure [9]

VIII. Result and Summery

The result of 3D reconstruction is shown in Fig.10.

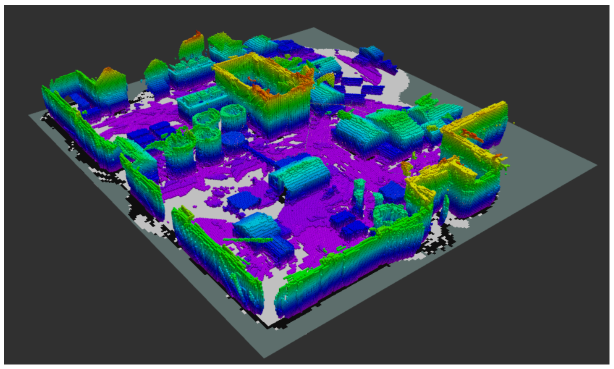

Fig. 10. Result of 3D Reconstruction of the environment

The quadrotor successfully navigates and maps the unknown environment autonomously.

For further development, the quadrotor should be tested in different maps to detect possible problems of the algorithm.

References

[1] Ben-Ari, M., Mondada, F. ”Robotic Motion and Odometry.” In: Elements of Robotics. Springer, Cham, 2018.
[2] Kai M. Wurm, Armin Hornung, Maren Bennewitz. ”OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees” on Autonomous Robots, 2013.
[3] Henry Alexander Ignatious, Hesham-El-Sayed, Manzoor Khan. ”An overview of sensors in Autonomous Vehicles”. International Workshop on Smart Communication and Autonomous Driving (SCAD 2021) November 1-4, 2021, Leuven, Belgium.
[4] Richard Hartley, Andrew Zisserman. ”Multiple View Geometry in Computer Vision”, 2004, pp.310 – 324.
[5] Elon Musk on Cameras vs LiDAR for Self Driving and Autonomous Cars, https://www.youtube.com/watch?v=HM23sjhtk4Q

[6] R. B. Rusu and S. Cousins, ”3D is here: Point Cloud Library (PCL)”, IEEE International Conference on Robotics and Automation, 2011.

[7] H. Moravec and A. Elfes, ”High resolution maps from wide angle sonar,” Proceedings. 1985 IEEE International Conference on Robotics and Automation, St. Louis, MO, USA, 1985, pp. 116-121

[8] Thomas Chevet, Cristina Stoica Maniu, Cristina Vlad, Youmin Zhang, ”Voronoi-based UAVs Formation Deployment and Reconfiguration using MPC Techniques”, 2018.

[9] Taeyoung Lee, Melvin Leok, N. Harris McClamroch. ”Geometric Tracking Control of a Quadrotor UAV on SE(3)”. 49th IEEE Conference on Decision and Control December 15-17, 2010

[10] Markus Achtelik, Michael Burri, Helen Oleynikova, Rik Bähnemann, Marija Popović, ”https://github.com/ethz-asl/mav_trajectory_generation”. Autonomous Systems Lab, ETH Zurich.

[11] C. Richter, A. Bry, and N. Roy, “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in International Journal of Robotics Research, Springer, 2016.