

Article

# Autonomous Navigation and Obstacle Avoidance for Small VTOL UAV in Unknown Environments

Cheng Chen <sup>1</sup>, Zian Wang <sup>2,\*</sup>, Zheng Gong <sup>3</sup>, Pengcheng Cai <sup>3</sup>, Chengxi Zhang <sup>4</sup> and Yi Li <sup>5,\*</sup><sup>1</sup> School of Aerospace Engineering, Shenyang Aerospace University, Shenyang 110000, China<sup>2</sup> China Academy of Launch Vehicle Technology, Beijing 100076, China<sup>3</sup> Department of Aerospace Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China<sup>4</sup> Key Laboratory of Advanced Control for Light Industry Processes, Ministry of Education, School of Internet of Things Engineering, Jiangnan University, Wuxi 214122, China<sup>5</sup> School of Electrical Engineering and Automation, Tianjin University of Technology, Tianjin 300384, China

\* Correspondence: wangzian@nuaa.edu.cn (Z.W.); ly@email.tjut.edu.cn (Y.L.)

**Abstract:** This paper takes autonomous exploration in unknown environments on a small co-axial twin-rotor unmanned aerial vehicle (UAV) platform as the task. The study of the fully autonomous positioning in unknown environments and navigation system without global navigation satellite system (GNSS) and other auxiliary positioning means is carried out. Algorithms that are based on the machine vision/proximity detection/inertial measurement unit, namely the combined navigation algorithm and indoor simultaneous location and mapping (SLAM) algorithm, are not only designed theoretically but also realized and verified in real surroundings. Additionally, obstacle detection, the decision-making of avoidance motion and motion planning methods such as Octree are also proposed, which are characterized by randomness and symmetry. The demonstration of the positioning and navigation system in the unknown environment and the verification of the indoor obstacle-avoidance flight were both completed through building an autonomous navigation and obstacle avoidance simulation system.

**Keywords:** autonomous navigation; obstacle avoidance; target detection; VI-SLAM



**Citation:** Chen, C.; Wang, Z.; Gong, Z.; Cai, P.; Zhang, C.; Li, Y.

Autonomous Navigation and Obstacle Avoidance for Small VTOL UAV in Unknown Environments. *Symmetry* **2022**, *14*, 2608. <https://doi.org/10.3390/sym14122608>

Academic Editors: Sergei D. Odintsov and Jan Awrejcewicz

Received: 16 September 2022

Accepted: 17 November 2022

Published: 9 December 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

With the development of UAV technology, UAVs are playing an increasingly essential role in some routine tasks or even under special circumstances in both civil and military applications [1–3]. For example, some UAVs can be used for military reconnaissance, autonomous identification and attack, and they can be also used to explore an unknown region and map it.

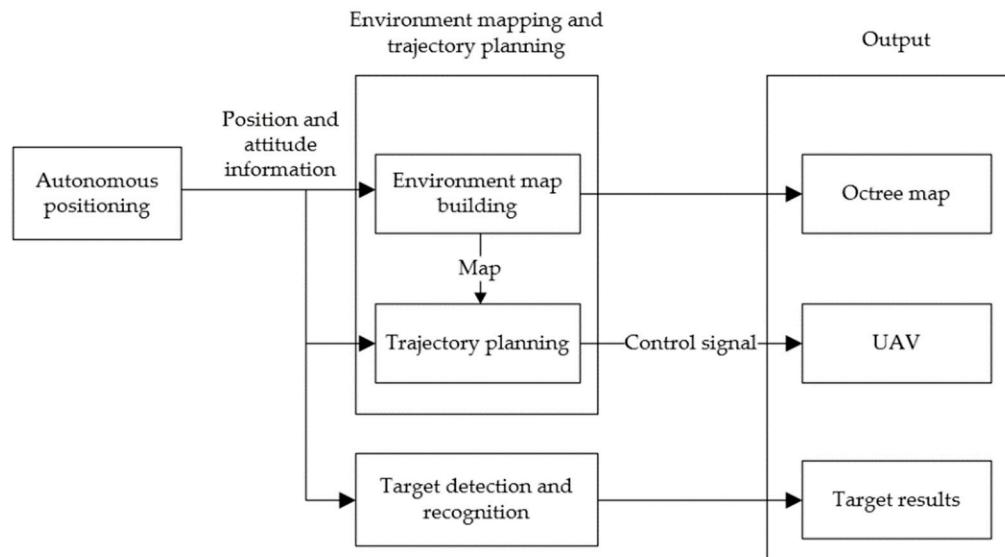
The survival capability of drones is a major problem, especially in some complex or even unknown environment; as a result, autonomous navigation is introduced. While external information should be introduced into the navigation system for better effects of flight control, the path planning and obstacle avoidance during autonomous navigation in unknown environments becomes a crucial issue for unmanned surface vehicles (USVs) [4].

A detection and avoidance system was presented for the autonomous navigation of UAVs in urban air mobility (UAM) applications by Enrique Aldao et al. [5]. The principle and navigation method of astronomical spectral velocity measurement, as well as the technical realization of the solar atomic frequency discriminator for autonomous navigation (SAFDAN) based on atomic frequency discrimination velocity measurement were comprehensively introduced by Wei Zhang et al. [6]. A self-trained controller for autonomous navigation in static and dynamic (with moving walls and nets) challenging environments (including trees, nets, windows, and pipe) using deep reinforcement learning, simultaneously trained using multiple rewards was introduced by Ramezani Dooraki Amir [7]. A

visual predictive control (VPC) scheme adapted to the autonomous navigation problem among static obstacles was proposed by Durand Petiteville A. [8]. Nowadays the majority of quadrotor drones are manually operated and use global positioning system (GPS) signals for navigation, thus greatly limiting the flight range of drones and consuming a lot of manpower and material resources. To solve the problem, Liu Liwen et al. [9] proposed a method of realizing autonomous flight and conflict avoidance of quadrotor UAVs by using a multi-sensor system and deep learning methods in extreme flight conditions through track prediction. Moreover, in the research of Sina Sajjadi [10], a vision-based target-tracking problem was formulated in the form of a cascaded adaptive nonlinear model predictive control (MPC) strategy. A typical ASV/USV unit with standard radio remote control system to the fully autonomous mode was modernized by Specht C et al. [11]. A method of the obstacle avoidance planning of unmanned surface vehicles based on an improved artificial potential field was proposed by S Xie et al. [12]. Navigation problems of unmanned aerial vehicles (UAVs) flying in a formation in a free and an obstacle-laden environment were investigated in the work of Xiaohua Wang et al. [13]. An unmanned underwater vehicle (UUV) simulator, an extension of the open-source robotics simulator Gazebo to underwater scenarios, was described in the work of Musa Morena Marcusso Manhães et al. [14].

This paper completes the development of an autonomous positioning algorithm and mapping and trajectory planning algorithm. Algorithms that are based on the machine vision/proximity detection/inertial measurement unit, namely the combined navigation algorithm and indoor SLAM algorithm, were designed and realized. Additionally, obstacle detection, the decision-making of avoidance motion and motion planning methods are also proposed. An autonomous navigation and obstacle avoidance simulation system is proposed. A target recognition algorithm was developed and finally the proposed autonomous navigation and obstacle avoidance simulation system was demonstrated and verified through physical experiments. The proposed algorithm and system play an important role in many practical systems and applications, such as sweeping robots, driverless cars, virtual reality technology (VR) and intelligent robots. According to the experiment results, the maximum error along x direction is less than 0.5 m, less than 0.6 m along y direction, and 0.4 m along z direction. The yaw angle error is less than  $5^\circ$ , and absolute error is less than 0.3 m. The calculated closed-loop error is about  $0.3/70 = 0.4\%$ .

The proposed autonomous navigation and obstacle avoidance system, mainly consisting of three components, namely autonomous positioning, environment mapping and trajectory planning and target detection and recognition, was used to realize autonomous environmental exploration without GPS. Its workflow and output are shown in Figure 1.



**Figure 1.** The structural diagram of the autonomous system.

Section 2 introduces the autonomous positioning algorithm and its simulation results. The detailed design and the mechanism of map-building and trajectory-planning algorithm are provided in Section 3. The detailed design of the target detection and recognition algorithm are in Section 4. In Section 5, the validation of the proposed algorithm is verified by a flight test and the test environment; the results of the flight test are also introduced in this part.

### 1.1. Autonomous Positioning

Visual–inertial simultaneous localization and mapping (VI-SLAM) [15–17] was used to solve the autonomous positioning problem of UAVs without GPS. The system established a global coordinate system by regarding the take-off position as the origin and estimated the relative pose of the UAV by the fusion of the measurement information of the vision and inertial navigation system.

### 1.2. Map Building and Trajectory Planning

Mapping and path planning were used to solve the motion planning problems of UAVs [18]. By building a raster map and running a path search algorithm, the UAV could be guided to specific targets and avoid known obstacles at the same time.

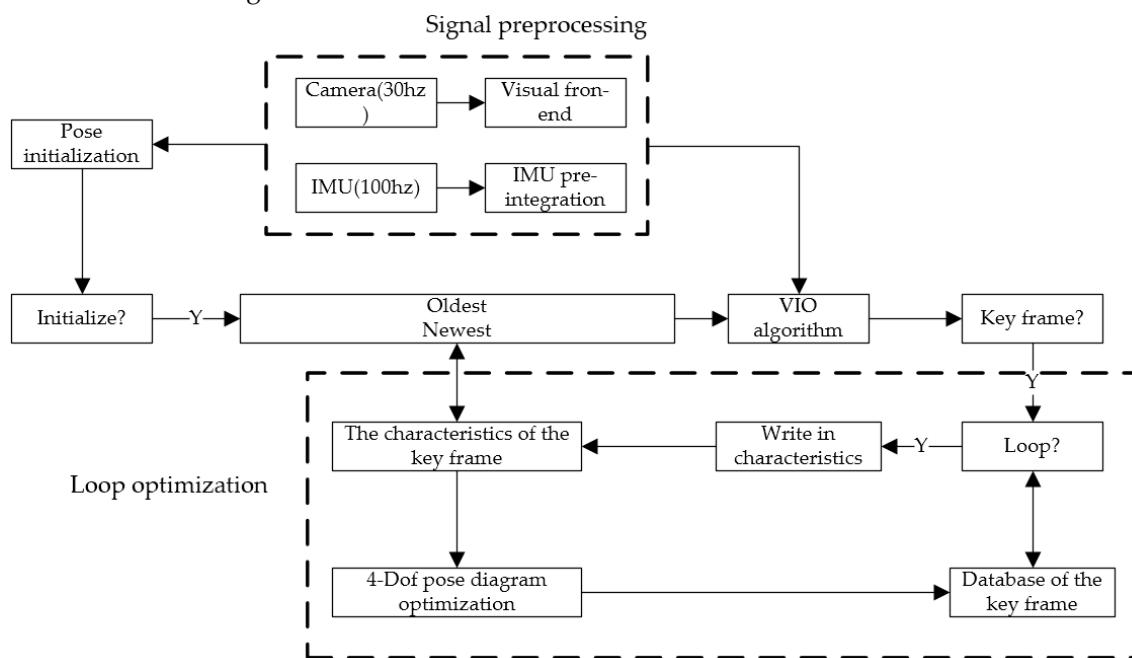
### 1.3. Target Detection and Recognition

The target detection and recognition system was used to search, detect and classify the targets in the field of vision, and provide reference information for the subsequent behavior decisions [19].

## 2. Autonomous Positioning

### 2.1. The Introduction of the Autonomous Positioning Module

The VI-SLAM algorithm adopts binocular and inertial measurement units (IMU). According to the operation process, the system is divided into four parallel parts: signal preprocessing thread, pose initialization thread, VisualInertial Odometry (VIO) thread, and loopback optimization thread [20,21]. The operation flow of the system is shown in Figure 2:



**Figure 2.** The flow diagram of VI-SLAM.

## 2.2. The Preprocessing of Signals

This section describes the VIO preprocessing procedures. For visual measurement, we tracked features between successive frames and detected new features in the latest frame. For IMU measurement, pre-integration was adopted between two consecutive frames. Due to the high measurement noise of the low-cost IMU, the offsets of the inertial components were obtained by external calibration during the pre-integration process.

### 2.2.1. Visual Front-End

The basic task of visual front-end is to extract feature points, which mainly includes two parts: feature tracking and key frame selection.

#### (1) Bidirectional Kanade–Lucas–Tomasi (KLT) tracking

For the binocular system, the left and right visual images,  $R_k$  and  $L_k$  were obtained in each time sequence. Firstly, Harris corner detection was used on the two images, and even distribution of feature was ensured by setting the minimum interval of pixels between two adjacent features. Then a KLT sparse optical flow algorithm and polar line search algorithm were used for feature matching, an RANSAC algorithm of basic matrix model was also used to remove outer points, and the matched binocular feature point pairs were obtained. We then applied the same to  $R_{k+1}$  and  $L_{k+1}$ .

Next, bidirectional KLT tracking was adopted for the feature points in  $R_k$  and  $R_{k+1}$ ; that is, a KLT matching and RANSAC screening was carried out from  $R_k$  to  $R_{k+1}$ , and then the remaining matching points were used for a matching and screening from  $R_{k+1}$  to  $R_k$  to ensure feature stability to the maximum extent.

#### (2) The selection strategy of the key frames

At the visual front-end, the key frame selection was performed simultaneously, and there were two selection criteria. The first one was the mean parallax from the previous keyframe. If the mean parallax of the feature points tracked between the current frame and the latest key frame exceeded a certain threshold, the frame would be considered as a new keyframe. Another one was tracking quality. If the number of tracked features was under a certain threshold, we treated this frame as a new key frame, which avoided the complete loss of tracking features.

### 2.2.2. IMU Pre-Integration

The rotation error of the Euler angle was parameterized by IMU pre-integration. Here, the pre-integration mode proposed by Vins-Mono was used, the covariance transfer function was derived through the IMU error state dynamics under continuous time and the bias correction was introduced to correct the error.

The measurement results of original gyroscope and accelerometer of IMU,  $\hat{\omega}$  and  $\hat{a}$  are shown as follows:

$$\begin{aligned}\hat{a}_t &= a_t + b_{a_t} + R_w^t g^w + n_a \\ \hat{\omega}_t &= \omega_t + b_{\omega_t} + n_{\omega}\end{aligned}\quad (1)$$

IMU measurement values were measured in the body coordinate system, which is the resultant force that balances the gravity and platform dynamics, and it can be affected by accelerometer offset  $b_a$ , gyroscope offset  $b_{\omega}$  and additional noise. Under the assumption that the additional noise in the measured value of accelerometer and gyroscope is Gaussian noise,  $n_a \sim N(0, \sigma_a^2)$ ,  $n_{\omega} \sim N(0, \sigma_{\omega}^2)$ . Accelerometer offset and gyroscope offset were modeled as random walks and their derivatives are  $n_{b_a} \sim N(0, \sigma_{b_a}^2)$  and  $n_{b_{\omega}} \sim N(0, \sigma_{b_{\omega}}^2)$ , respectively.

$$\begin{aligned}\dot{b}_{a_t} &= n_{b_a} \\ \dot{b}_{\omega_t} &= n_{b_{\omega}}\end{aligned}\quad (2)$$

Given two moments corresponding to the body coordinate system  $b_k$  and  $b_{k+1}$ , the position, velocity, and direction states can be transmitted by inertial measurement values in the world coordinate system between time intervals  $[t_k, t_{k+1}]$ :

$$\begin{aligned} p_{b_{k+1}}^\omega &= p_{b_k}^\omega + v_{b_k}^\omega \Delta t_k \\ &\quad + \iint_{t \in [t_k, t_{k+1}]} (R_t^\omega (\hat{a}_t - b_{a_t} - n_a) - g^\omega) dt^2 \\ v_{b_{k+1}}^\omega &= v_{b_k}^\omega + \int_{t \in [t_k, t_{k+1}]} (R_t^\omega (\hat{a}_t - b_{a_t} - n_a) - g^\omega) dt \\ q_{b_{k+1}}^\omega &= q_{b_k}^\omega \otimes \int_{t \in [t_k, t_{k+1}]} \frac{1}{2} \Omega (\hat{\omega}_t - b_{\omega_t} - n_\omega) q_t^{b_k} dt \end{aligned} \quad (3)$$

where

$$\Omega(\omega) = \begin{bmatrix} -[\omega]_x & \omega \\ -\omega^T & 0 \end{bmatrix} \cdot [\omega]_x = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_x & 0 & -\omega_x \\ -\omega_y & \omega_z & 0 \end{bmatrix} \quad (4)$$

$\Delta t_k$  is the span of the interval  $[t_k, t_{k+1}]$ .

Clearly, the state transmission of IMU requires the rotation, position and velocity of the coordinate system  $b_k$ . When these initial states change, we need to retransmit the IMU measurement values. Especially in optimization-based algorithms, IMU measurement values need to be retransmitted between them every time the pose is adjusted, and this transfer strategy is computationally demanding. In order to avoid retransmission, a pre-integration algorithm was introduced.

After changing the reference coordinate system from the world coordinate system to the local coordinate system  $b_k$ , pre-integration can be only applied to the relevant part of the linear acceleration  $\hat{a}$  and angular velocity  $\hat{\omega}$  as follows:

$$\begin{aligned} \mathbf{R}_w^{b_k} \mathbf{p}_{b_{k+1}}^w &= \mathbf{R}_w^{b_k} \left( \mathbf{p}_{b_k}^w + \mathbf{v}_{b_k}^w \Delta t_k - \frac{1}{2} \mathbf{g}^w \Delta t_k^2 \right) + \boldsymbol{\alpha}_{b_{k+1}}^{b_k} \\ \mathbf{R}_w^{b_k} \mathbf{v}_{b_{k+1}}^w &= \mathbf{R}_w^{b_k} \left( \mathbf{v}_{b_k}^w - \mathbf{g}^w \Delta t_k \right) + \boldsymbol{\beta}_{b_{k+1}}^{b_k} \\ \mathbf{q}_w^{b_k} \otimes \mathbf{q}_{b_{k+1}}^w &= \gamma_{b_{k+1}}^{b_k} n \end{aligned} \quad (5)$$

$$\begin{aligned} \boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \iint_{t \in [t_k, t_{k+1}]} \mathbf{R}_t^{b_k} \left( \hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a \right) dt^2 \\ \boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \int_{t \in [t_k, t_{k+1}]} \mathbf{R}_t^{b_k} \left( \hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a \right) dt \\ \gamma_{b_{k+1}}^{b_k} &= \int_{t \in [t_k, t_{k+1}]} \frac{1}{2} \Omega (\hat{\omega}_t - b_{\omega_t} - n_\omega) \gamma_t^{b_k} dt \end{aligned} \quad (6)$$

Among them, the pre-integration term (6) can be obtained by the IMU measurement value, which regards  $b_k$  as the reference frame.  $\boldsymbol{\alpha}_{b_{k+1}}^{b_k}$ ,  $\boldsymbol{\beta}_{b_{k+1}}^{b_k}$ ,  $\gamma_{b_{k+1}}^{b_k}$  are only related to the IMU offset in  $b_k$  and  $b_{k+1}$ , and have nothing to do with other states. When the offset estimation changed, if the change was small, we adjusted  $\boldsymbol{\alpha}_{b_{k+1}}^{b_k}$ ,  $\boldsymbol{\beta}_{b_{k+1}}^{b_k}$  and  $\gamma_{b_{k+1}}^{b_k}$  according to their first-order approximation to the offset; otherwise, was retransmitted. This strategy saves a lot of computational resources for optimization-based algorithms because the repeated transmission of IMU measurement values is avoided.

Under discrete-time conditions, different numerical integration methods can be used, such as Euler integration, midpoint integration, RK4 integration, etc. The Euler integral is chosen in this section.

At the beginning,  $\boldsymbol{\alpha}_{b_k}^{b_k}$  and  $\boldsymbol{\beta}_{b_k}^{b_k}$  were 0, and  $\gamma_{b_k}^{b_k}$  was a unit quaternion. The average values of  $\boldsymbol{\alpha}$ ,  $\boldsymbol{\beta}$  and  $\boldsymbol{\gamma}$  in (6) were gradually transmitted as follows. The additional noise

$n_a, n_\omega$  were unknown and they were treated as 0 in the actual program. The estimated value of pre-integration is obtained as follows:

$$\begin{aligned}\hat{\alpha}_{i+1}^{b_k} &= \hat{\alpha}_i^{b_k} + \hat{\beta}_i^{b_k} \delta t + \frac{1}{2} \mathbf{R}(\hat{\gamma}_i^{b_k})(\hat{\mathbf{a}}_i - \mathbf{b}_{a_i}) \delta t^2 \\ \hat{\beta}_{i+1}^{b_k} &= \hat{\beta}_i^{b_k} + \mathbf{R}(\hat{\gamma}_i^{b_k})(\hat{\mathbf{a}}_i - \mathbf{b}_{a_i}) \delta t \\ \hat{\gamma}_{i+1}^{b_k} &= \hat{\gamma}_i^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}(\hat{\omega}_i - \mathbf{b}_{\omega_i}) \delta t \end{bmatrix}\end{aligned}\quad (7)$$

where  $i$  is the discrete moment corresponding to the IMU measurement value during time interval  $[t_k, t_{k+1}]$ , and  $\delta t$  is the time interval between IMU measurement value  $i$  and  $i + 1$ .

We then turned our focus to the covariance transmission problem. Since the four-dimensional rotational quaternion  $\gamma_i^{b_k}$  was over-parameterized, we defined its error as the perturbation around its mean value:

$$\gamma_t^{b_k} \approx \hat{\gamma}_t^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta_t^{b_k} \end{bmatrix} \quad (8)$$

where  $\delta\theta_t^{b_k}$  is the three-dimensional small perturbation.

Thus, the linearized equation of the error term under continuous time can be derived as follows:

$$\begin{aligned}\begin{bmatrix} \delta\alpha_t^{b_k} \\ \delta\beta_t^{b_k} \\ \delta\theta_t^{b_k} \\ \delta b_{a_t} \\ \delta b_{\omega_t} \end{bmatrix} &= \begin{bmatrix} 0 & I & 0 & 0 & 0 \\ 0 & 0 & -R_t^{b_k}[\hat{\mathbf{a}}_t - \mathbf{b}_{a_t}]_\times & -R_t^{b_k} & 0 \\ 0 & 0 & -[\hat{\omega}_t - \mathbf{b}_{\omega_t}]_\times & 0 & -I \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta\alpha_t^{b_k} \\ \delta\beta_t^{b_k} \\ \delta\theta_t^{b_k} \\ \delta b_{a_t} \\ \delta b_{\omega_t} \end{bmatrix} \\ &+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ -R_t^{b_k} & 0 & 0 & 0 \\ 0 & -I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} n_a \\ n_\omega \\ n_{b_a} \\ n_{b_\omega} \end{bmatrix} \\ &= F_t \delta z_t^{b_k} + G_t n_t\end{aligned}\quad (9)$$

$\mathbf{P}_{b_{k+1}}^{b_k}$  can be calculated by the recursion and updating of the first-order discrete-time covariance of initial covariance  $\mathbf{P}_{b_k}^{b_k} = 0$ :

$$\mathbf{P}_{t+\delta t}^{b_k} = (\mathbf{I} + \mathbf{F}_t \delta t) \mathbf{P}_t^{b_k} (\mathbf{I} + \mathbf{F}_t \delta t)^T + (\mathbf{G}_t \delta t) \mathbf{Q} (\mathbf{G}_t \delta t)^T \quad t \in [k, k+1] \quad (10)$$

where  $\mathbf{Q}$  is the diagonal covariance matrix  $(\sigma_a^2, \sigma_\omega^2, \sigma_{b_a}^2, \sigma_{b_\omega}^2)$  of the noise.

Meanwhile, the first-order Jacobian matrix  $J_{b_{k+1}}$  of  $\delta z_{b_{k+1}}^{b_k}$  can also be calculated by the recursion of the initial Jacobian matrix  $J_{b_{k+1}} = I$ .

$$\mathbf{J}_{t+\delta t} = (\mathbf{I} + \mathbf{F}_t \delta t) \mathbf{J}_t, \quad t \in [k, k+1] \quad (11)$$

Using Equation (11), covariance matrix  $\mathbf{P}_{b_{k+1}}^{b_k}$  and Jacobian matrix  $J_{b_{k+1}}$  were obtained. The first-order approximation of  $\alpha_{b_{k+1}}^{b_k}, \beta_{b_{k+1}}^{b_k}, \gamma_{b_{k+1}}^{b_k}$  relevant to the offset can be expressed as follows:

$$\begin{aligned}\alpha_{b_{k+1}}^{b_k} &\approx \hat{\alpha}_{b_{k+1}}^{b_k} + J_{b_a}^{\alpha} \delta \mathbf{b}_{a_k} + J_{b_w}^{\alpha} \delta \mathbf{b}_{w_k} \\ \beta_{b_{k+1}}^{b_k} &\approx \hat{\beta}_{b_{k+1}}^{b_k} + J_{b_a}^{\beta} \delta \mathbf{b}_{a_k} + J_{b_w}^{\beta} \delta \mathbf{b}_{w_k} \\ \gamma_{b_{k+1}}^{b_k} &\approx \hat{\gamma}_{b_{k+1}}^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} J_{b_w}^{\gamma} \delta \mathbf{b}_{w_k} \end{bmatrix}\end{aligned}\quad (12)$$

where  $J_{b_a}^{\alpha}$  is the subblock matrix of  $J_{b_{k+1}}$ , and its position corresponds to  $\frac{\delta \alpha_{b_{k+1}}^{b_k}}{\delta b_{a_k}}$ , which also makes sense for  $J_{b_w}^{\alpha}, J_{b_a}^{\beta}, J_{b_w}^{\beta}, J_{b_w}^{\gamma}$ .

When the offset estimation changed slightly, we used Equation (12) to approximately correct the results of pre-integration without retransmission.

Hence, the corresponding covariance  $P_{b_{k+1}}^{b_k}$  of the IMU measurement model could be obtained:

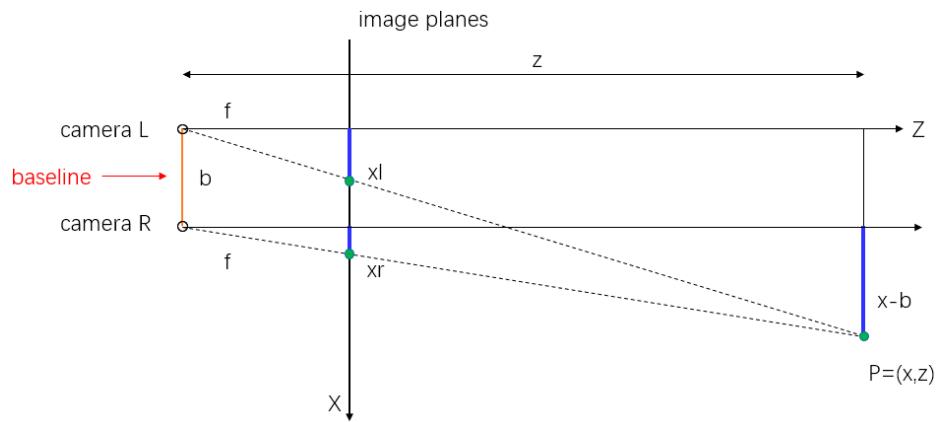
$$\begin{bmatrix} \hat{\alpha}_{b_{k+1}}^{b_k} \\ \hat{\beta}_{b_{k+1}}^{b_k} \\ \hat{\gamma}_{b_{k+1}}^{b_k} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_w^{b_k} \left( \mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \frac{1}{2} \mathbf{g}^w \Delta t_k^2 - \mathbf{v}_{b_k}^w \Delta t_k \right) \\ \mathbf{R}_w^{b_k} \left( \mathbf{v}_{b_{k+1}}^w + \mathbf{g}^w \Delta t_k - \mathbf{v}_{b_k}^w \right) \\ \mathbf{q}_{b_k}^{w^{-1}} \otimes \mathbf{q}_{b_{k+1}}^w \\ \mathbf{b}_{ab_{k+1}} - \mathbf{b}_{ab_k} \\ \mathbf{b}_{wb_{k+1}} - \mathbf{b}_{wb_k} \end{bmatrix} \quad (13)$$

### 2.3. Pose Initialization

The pose initialization part is responsible for establishing the coordinate system and maintaining the feature points and the description of UAV in the coordinate system at the early stage of the operation process of system [22,23]. Compared with the monocular, tightly-coupled VIO system, the binocular system can directly recover the depth of feature points to complete initialization under stationary conditions.

#### 2.3.1. The Depth Estimation of Feature Points

Since the binocular camera system was used, the depth of feature points could be calculated directly from the disparity and the relative pose of the camera. The analysis started with ideal conditions: under the assumption that the left and right cameras were in the same plane (the optical axis was parallel) and the camera parameters (focal length  $f$ ) were identical. Then the depth value could be obtained, as shown in Figure 3:



**Figure 3.** Diagram of the imaging model.

As can be seen from Figure 3 above, an image plane was established with X, Z axes and the distance between point P and the axis of camera R is ' $x - b$ '; the intersection point

of the link between camera L and P in X axis is recorded as ‘xl’ (‘l’ means ‘left’) and the same for ‘xr’; ‘b’ is the length of baseline.

According to the triangle similarity:

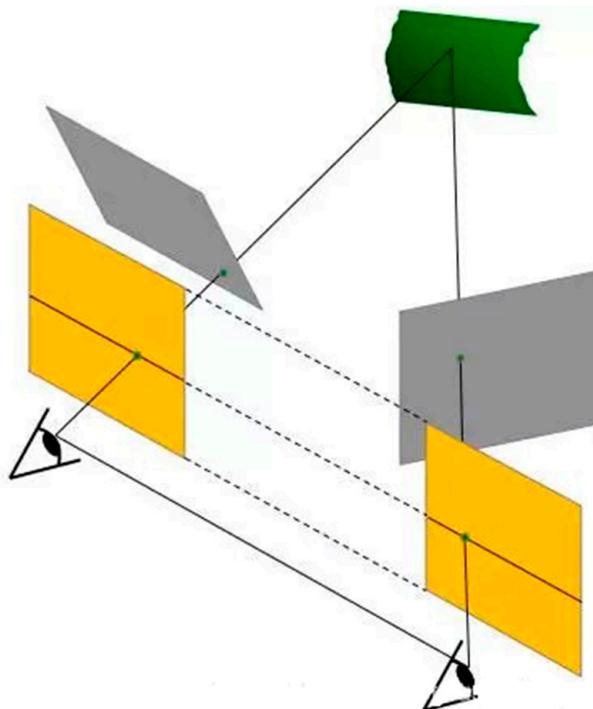
$$\begin{aligned}\frac{z}{f} &= \frac{x}{xl} = \frac{x-b}{yr} \\ \frac{z}{f} &= \frac{y}{yl} = \frac{y}{yr}\end{aligned}\quad (14)$$

where b is the baseline length, and the optical axes of the two cameras are both located in the XOZ plane.

Then the position of point P can be estimated:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} xl \cdot z/f \\ yl \cdot z/f \\ f \cdot b/(xl - xr) \end{bmatrix} = \begin{bmatrix} b + xr \cdot z/f \\ yr \cdot z/f \\ f \cdot b/d \end{bmatrix} \quad (15)$$

For non-ideal camera imaging model, the perturbation included optical axis deviation and image distortion. In this case, it was necessary to correct the image and transform it into the ideal situation, as shown in Figure 4.



**Figure 4.** Image correction.

After obtaining the space coordinates of feature points, they needed to be converted into inverse depth to connect them with the SLAM system. Compared with the direct depth expression, the inverse depth error is more consistent with the Gaussian distribution and has better numerical stability. The conversion formula of inverse depth  $\lambda$  is as follows:

$$\begin{aligned}\lambda &= 1/d \\ &= 1/\sqrt{x^2 + y^2 + z^2}\end{aligned}\quad (16)$$

### 2.3.2. Pose Initialization

When establishing the SLAM coordinate system, the northeast sky coordinate system was established by taking the origin of the camera coordinate system in the first frame as

the origin under state of rest. Then the  $i$ th landmark feature point  $m_i$  in the first frame can be expressed by the inverse depth in the world coordinate system as follows:

$$\begin{bmatrix} \alpha_i \\ \beta_i \\ \lambda_i \end{bmatrix} = \begin{bmatrix} \text{atan}(y/z) \\ \text{atan}(-x/z) \\ 1/\sqrt{x^2 + y^2 + z^2} \end{bmatrix} \quad (17)$$

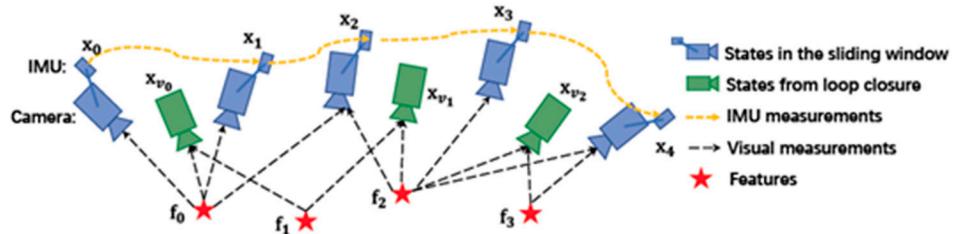
In the tracking process of the second frame, the pose of the second frame was obtained by matching the landmark point in the new frame and the counterpart in the first frame and running pose calculation, and the available landmark points were updated for the subsequent pose calculation.

#### 2.4. VIO Algorithm

As the core part of the pose updating of the VI-SLAM algorithm, the VIO algorithm requires accuracy and running speed at the same time. Therefore, the sliding window method based on a nonlinear optimization strategy was selected. The basic idea of the sliding window method is firstly introduced in this section, and then the calculation methods of IMU and visual measurement residual that needed to be updated in the formula are introduced separately.

##### 2.4.1. Sliding Window Method

After the initialization of the estimator, the binocular VIO based on sliding windows was employed for high-precision and robust state estimation. The diagram of the sliding window method is shown in Figure 5:



**Figure 5.** Sliding window method.

The full-state vector in the sliding window is defined as:

$$\begin{aligned} \mathcal{X} &= [x_0, x_1, \dots, x_n, x_c^b, \lambda_0, \lambda_1, \dots, \lambda_m] \\ x_k &= [p_{b_k}^w, v_{b_k}^w, q_{b_k}^w, b_a, b_g], k \in [0, n] \\ x_c^b &= [p_c^b, q_c^b] \end{aligned} \quad (18)$$

where  $x_k$  is the IMU state when the  $k$ th image is captured. It contains the position, velocity and orientation of IMU in the world coordinate system, as well as the accelerometer offset and gyroscope offset in the IMU body coordinate system.  $n$  is the total number of keyframes,  $m$  is the total number of features in the sliding window, and  $\lambda_l$  is the inverse depth when watching the  $l$ th feature the first time.

Visual inertia BA was used here. We minimized the sum of the prior and the Mahalanobis norm [24] of all the measurement residuals to obtain the maximum posterior estimation:

$$\min_{\mathcal{X}} \left\{ \| \mathbf{r}_p - \mathbf{H}_p \mathcal{X} \|^2 + \sum_{k \in \mathcal{B}} \| \mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}) \|_{\mathbf{P}_{b_{k+1}}^{b_k}}^2 + \sum_{(l,j) \in \mathcal{C}} \rho \left( \| \mathbf{r}_{\mathcal{C}}(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}) \|_{\mathbf{P}_l^{c_j}}^2 \right) \right\} \quad (19)$$

where the Huber norm [25]  $\rho(s)$  is defined as follows:

$$\rho(s) = \begin{cases} 1 & s \geq 1 \\ 2\sqrt{s} - 1 & s < 1 \end{cases} \quad (20)$$

$\mathbf{r}_B(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X})$ ,  $\mathbf{r}_C(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X})$  are the residuals of IMU and visual measurement, respectively, which are defined in detail in Equations (21) and (22). B is the set of all IMU measurements and C is a set of features observed at least two times in the current sliding window. The ceres nonlinear optimization library was used to solve the algorithm.

#### 2.4.2. The Calculation of IMU Measurement Residual

Taking the IMU measurement between two consecutive frames  $b_k$  and  $b_{k+1}$  in the sliding window, according to the IMU measurement model defined in (13), the residual of pre-integration IMU measurement can be defined as:

$$\begin{aligned} \mathbf{r}_B(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}) &= \begin{bmatrix} \delta\alpha_{b_{k+1}}^{b_k} \\ \delta\beta_{b_{k+1}}^{b_k} \\ \delta\theta_{b_{k+1}}^{b_k} \\ \delta b_a \\ \delta b_g \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}_w^{b_k} (\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \frac{1}{2}\mathbf{g}^w \Delta t_k^2 - \mathbf{v}_{b_k}^w \Delta t_k) - \hat{\mathbf{a}}_{b_{k+1}}^{b_k} \\ \mathbf{R}_w^{b_k} (\mathbf{v}_{b_{k+1}}^w + \mathbf{g}^w \Delta t_k - \mathbf{v}_{b_k}^w) - \hat{\beta}_{b_{k+1}}^{b_k} \\ 2 \left[ \mathbf{q}_{b_k}^{w^{-1}} \otimes \mathbf{q}_{b_{k+1}}^w \otimes (\hat{\gamma}_{b_{k+1}}^{b_k})^{-1} \right]_{xyz} \\ \mathbf{b}_{ab_{k+1}} - \mathbf{b}_{ab_k} \\ \mathbf{b}_{wb_{k+1}} - \mathbf{b}_{wb_k} \end{bmatrix} \quad (21) \end{aligned}$$

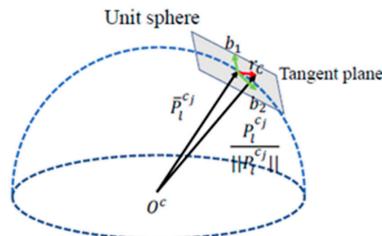
where  $[\cdot]_{xyz}$  is to extract the vector part of quaternion q for error state expression.  $\delta\theta_{b_{k+1}}^{b_k}$  is a three-dimensional error state expression of a quaternion.  $[\hat{\mathbf{a}}_{b_{k+1}}^{b_k}, \hat{\beta}_{b_{k+1}}^{b_k}, \hat{\gamma}_{b_{k+1}}^{b_k}]^T$  is an IMU measurement term that is obtained through the pre-integration of the measurement values of accelerometer and gyroscope measurements containing only noise during the time interval of two consecutive image frames. Accelerometer and gyroscope offset are also included in the remaining terms of the online correction.

#### 2.4.3. Visual Measurement Residual

In contrast to the traditional pinhole camera models in which the reprojection error is defined on the generalized image plane, the measurement residuals of a camera are defined on the unit sphere. The optics of almost all types of cameras, including wide-angle, fisheye or omnidirectional cameras, can be modeled as unit rays connected to the surface of a unit sphere. Assuming that the  $l$ th feature is first observed in the  $i$ th image, the residual of the feature observation in the  $j$ th image is defined:

$$\begin{aligned} \mathbf{r}_C(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}) &= [\mathbf{b}_1 \quad \mathbf{b}_2]^T \cdot \left( \hat{\mathcal{P}}_l^{c_j} - \frac{\mathcal{P}_l^{c_j}}{\|\mathcal{P}_l^{c_j}\|} \right) \\ \hat{\mathcal{P}}_l^{c_j} &= \pi_c^{-1} \left( \begin{bmatrix} \hat{u}_l^{c_j} \\ \hat{v}_l^{c_j} \end{bmatrix} \right) \\ P_l^{c_j} &= \mathbf{R}_b^{b_j} \left( \mathbf{R}_w^b \left( \mathbf{R}_{b_i}^b \left( \mathbf{R}_c^b \frac{1}{\lambda_l} \pi_c^{-1} \left( \begin{bmatrix} u_l^{c_i} \\ v_l^{c_i} \end{bmatrix} \right) + \mathbf{p}_c^b \right) + \mathbf{p}_{b_i}^w - \mathbf{p}_{b_j}^w \right) - \mathbf{p}_c^b \right) \quad (22) \end{aligned}$$

where  $[u_l^{c_i} \ v_l^{c_i}]^T$  is the  $l$ th feature which is observed in the  $i$ th image the first time.  $[\hat{u}_l^{c_j} \ \hat{v}_l^{c_j}]^T$  is the observation of the same feature in the  $j$ th image.  $\pi_c^{-1}$  is a back projection function that converts pixel positions into unit vectors by using internal parameters of camera. Since the degree of freedom of the visual residuals is 2, we project the residual vector onto the tangent plane. As shown below,  $b_1, b_2$  are two randomly chosen orthogonal bases in the tangent plane  $\hat{P}_l^{c_j}$ , and a group of  $b_1, b_2$  can be easily found. In Equation (22), with fixed length,  $P_l^{c_j}$  is the standard covariance in tangent space, as shown in Figure 6.



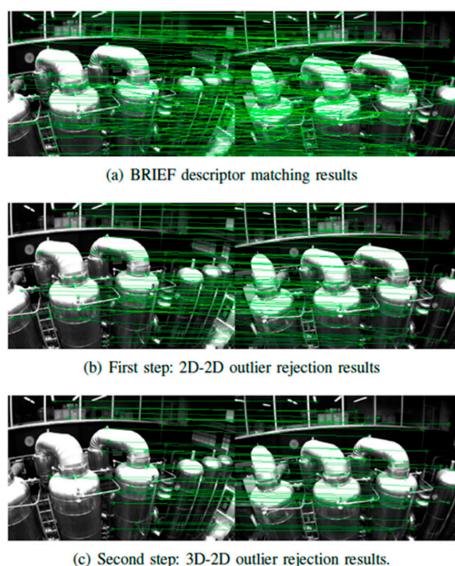
**Figure 6.** Tangent plane of residual projection.

## 2.5. Loopback Optimization

Due to measurement and calibration errors, VIO algorithm drifts may cause reduction in positioning accuracy at any time. The loopback optimization method can form additional restraints and suppress the drift problems by estimating the pose changes between some non-adjacent frames. In the loopback optimization part, the DBoW method was first used for loopback detection, then a bidirectional KLT algorithm was used to determine the matching point pairs. The PNP method was used to solve the pose change between two frames, and finally the loopback edge was written into the pose map for overall optimization.

### 2.5.1. DBoW Loopback Detection

The algorithm, by reference to VINS-MONO, used DBoW2 image similarity evaluation method for loopback detection. The DBoW2 model is the most advanced word bag model, which abstracts images into keyword descriptions for matching. In addition, the pre-stored feature points of the key frame and their descriptors were also used for feature-matching to improve loopback recall. DBoW2 returns loopback detection candidate frames after a temporal and spatial consistency check, as shown in Figure 7.



**Figure 7.** Loopback detection and exterior point elimination (the same method in VINS-MONO).

### 2.5.2. Bidirectional KLT Tracking and PNP Relocation

Like in Section 2.2.1 (1), bidirectional KLT tracking was used to obtain matching feature point pairs between two frame feature points with loopback, and then the PNP algorithm was used to obtain the pose changes between two frames.

### 2.5.3. The Management of 4-Dof Pose Diagram

When creating the pose map, the information  $\hat{\phi}, \hat{\theta}$  obtained by IMU estimation was considered as accurate and they were therefore free from optimization. Therefore, the pose map only contained the remaining 4Dof, namely the yaw angle  $\psi_i$  and its position information  $x, y, z$ , respectively.

Here, the edge residual between frames  $i$  and  $j$  is defined as:

$$\mathbf{r}_{ij}(\mathbf{p}_i^w, \psi_i, \mathbf{p}_j^w, \psi_j) = \begin{bmatrix} \mathbf{R}(\hat{\phi}_i, \hat{\theta}_i, \psi_i)^{-1}(\mathbf{p}_j^w - \mathbf{p}_i^w) - \hat{\mathbf{p}}_{ij}^i \\ \psi_j - \psi_i - \hat{\psi}_{ij} \end{bmatrix} \quad (23)$$

Among them,  $\hat{\phi}_i, \hat{\theta}_i$  are IMU roll and pitch angle estimations that were directly obtained from monocular VIO.

The whole pose map with sequential edges and loop-back edges is optimized by minimizing the following cost function:

$$\min_{\mathbf{P}, \psi} \left\{ \sum_{(i,j) \in S} \|\mathbf{r}_{ij}\|^2 + \sum_{(i,j) \in L} \rho(\|\mathbf{r}_{ij}\|^2) \right\} \quad (24)$$

where  $S$  is the set of all sequential edges and  $L$  is the set of loopback edges. Although tightly coupled relocation was able to reduce false loopbacks, a Huber norm  $\rho(\cdot)$  was introduced to further eliminate false loopbacks. In addition, any high-robustness norm was not used between sequential edges, and VIO was considered to have a strong enough elimination mechanism for exterior points.

## 2.6. Simulation Analysis Test

Before the real flight verification, a physical simulation engine was firstly built in the project, and the ROS Gazebo + Pixhawk scheme was adopted to realize the simulation verification of the algorithm.

### 2.6.1. Simulation Engine Gazebo

Gazebo is a 3-D dynamic simulator that accurately and effectively simulates robot crowds in complex indoor and outdoor environments, as shown in Figure 8. In the same way that game engines provide high-fidelity visual simulations, Gazebo provides high-fidelity physical simulations as well as a full suite of sensor models, and very user-friendly and programs-friendly interactions.

The typical uses of Gazebo include:

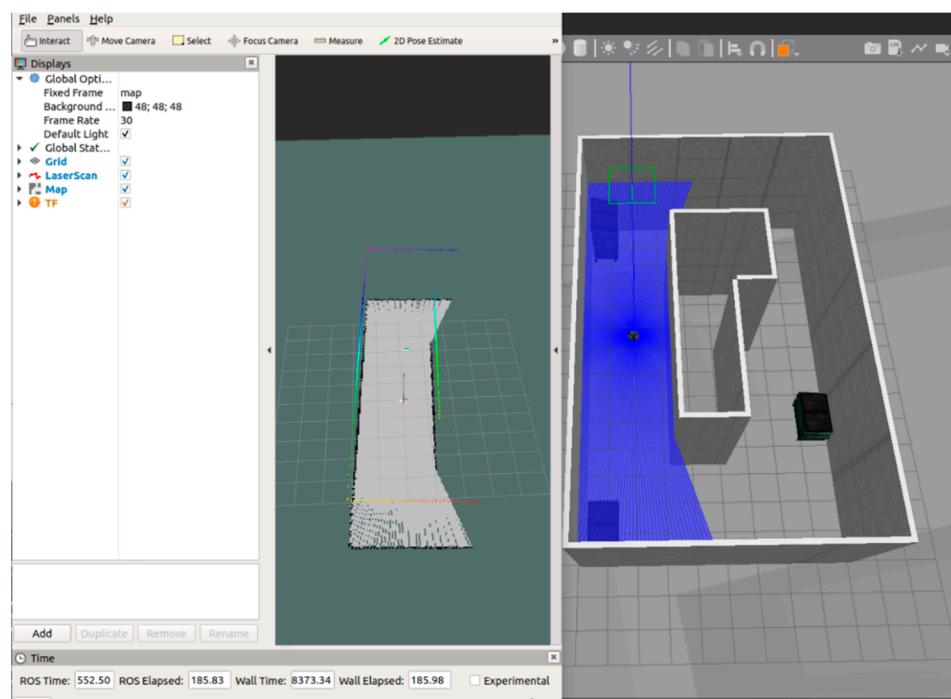
- To test a robot algorithm;
- To design a robot;
- To perform a regression test in actual scenarios.

This engine possesses the following characteristics:

- It contains multiple physics engines;
- It contains a rich library of robot models and environments;
- It contains a variety of sensors;
- The program is convenient to design and has a simple graphical interface.

Gazebo can build a simulation scene for robot tests. It can imitate the real world by adding objects library, garbage bins, ice cream buckets, and even dolls. It can also introduce

2D house design drawings using a building editor and build 3D houses based on the design drawings.

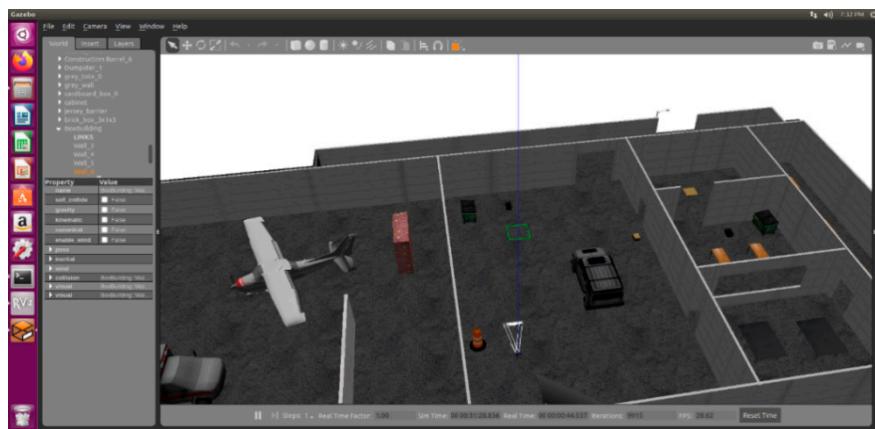


**Figure 8.** Gazebo.

Gazebo has a very powerful sensor model library, including camera, depth camera, laser, IMU and other sensors that are commonly used by robots. In addition, it has a simulation library, which can be used directly. A new sensor can also be created without any basis and have its specific parameters added. A sensor noise model can even be added to make the sensor more realistic.

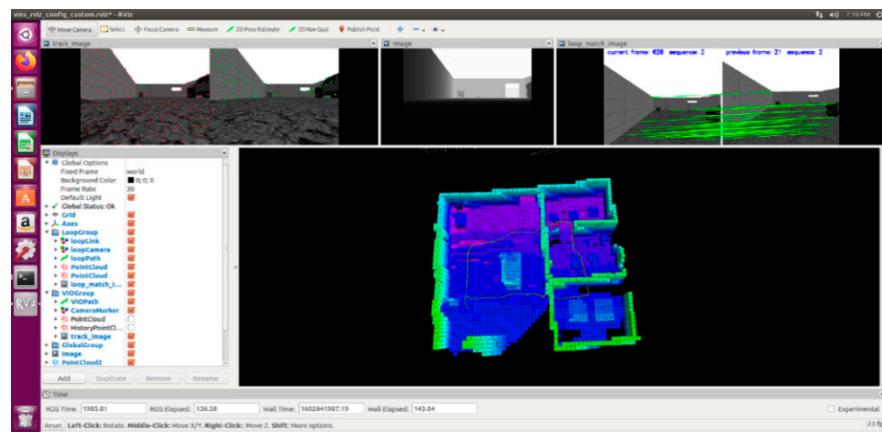
### 2.6.2. Simulation System

The simulation was carried out in the Gazebo engine. A PX4 UAV with a built-in IRIS platform was selected, carrying a RealSense d435i depth camera. The simulation environment was as shown in Figures 9 and 10:



**Figure 9.** Simulation environment.

The flight path of the UAV is shown below:



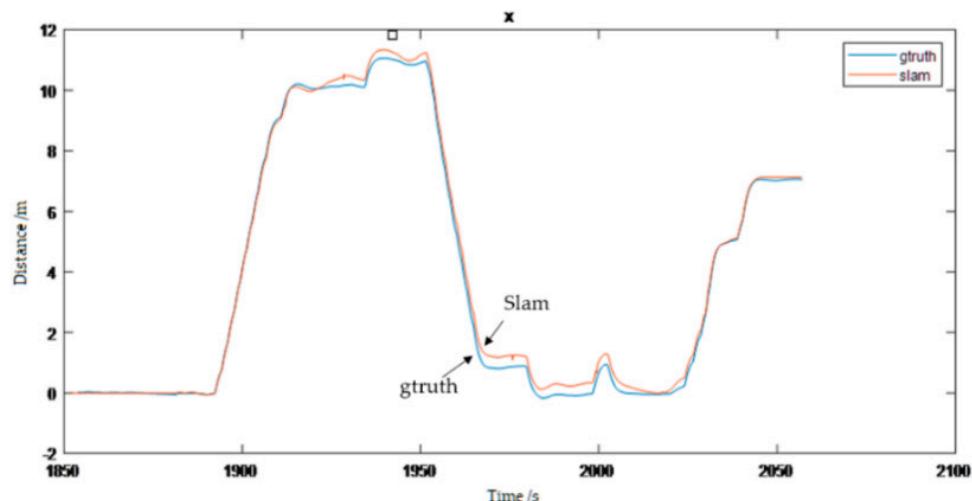
**Figure 10.** Schematic diagram of the flight path.

Since the final degree of freedom of the pose map was 4Dof, the roll angle and pitch angle directly determined by IMU were ignored in the evaluation process, and the accuracy of the four degrees of freedom of x, y, z and yaw were investigated, as shown in Table 1.

**Table 1.** Quantitative interpretation and conclusion of Figures 11–15.

Maximum error along X direction	< 0.5 m	Yaw angle error	< 5°
Maximum error along Y direction	< 0.6 m	Absolute error	< 0.3 m
Maximum error along Z direction	< 0.4 m	Calculated closed-loop error	≈ 0.4%
Standard deviation	X	< 0.05 m	
	Y	< 0.06 m	
	Z	< 0.03 m	

In the  $12\text{ m} \times 14\text{ m}$  orbit with a total length of about 70 m, the maximum error along the x direction was less than 0.5 m, less than 0.6 m along the y direction, and 0.4 m along the z direction. The yaw angle error was less than  $5^\circ$ , and absolute error was less than 0.3 m. The yaw error generally stayed at zero, with some small fluctuations when the yaw angle changed abruptly. The calculated closed-loop error was about  $0.3/70 = 0.4\%$ .



**Figure 11.** Comparison between SLAM and real value along x axis.

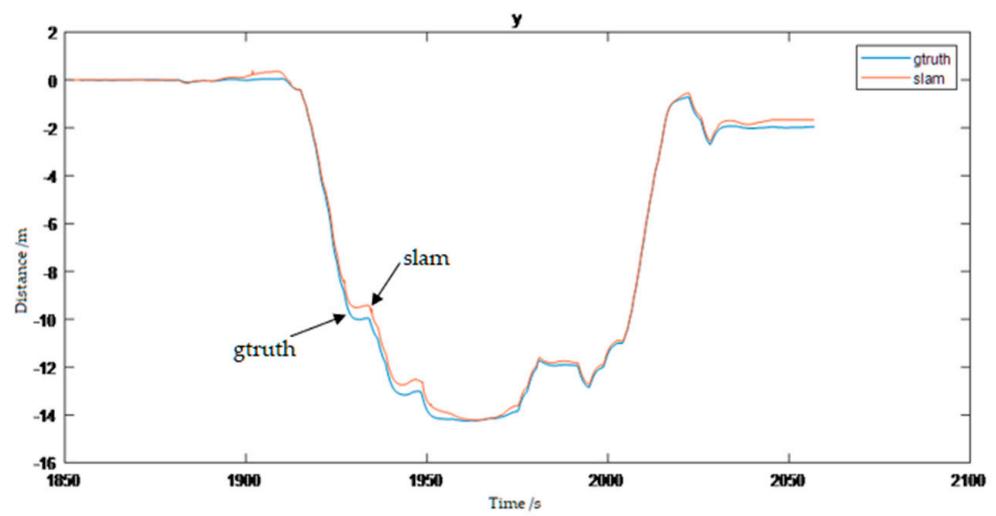


Figure 12. Comparison between SLAM and real value along y axis.

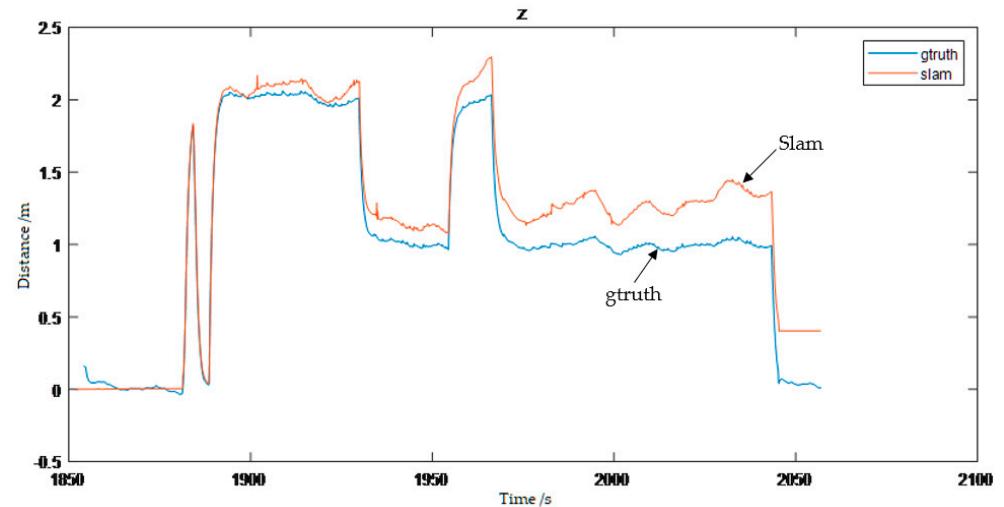


Figure 13. Comparison between SLAM and real value along z axis.

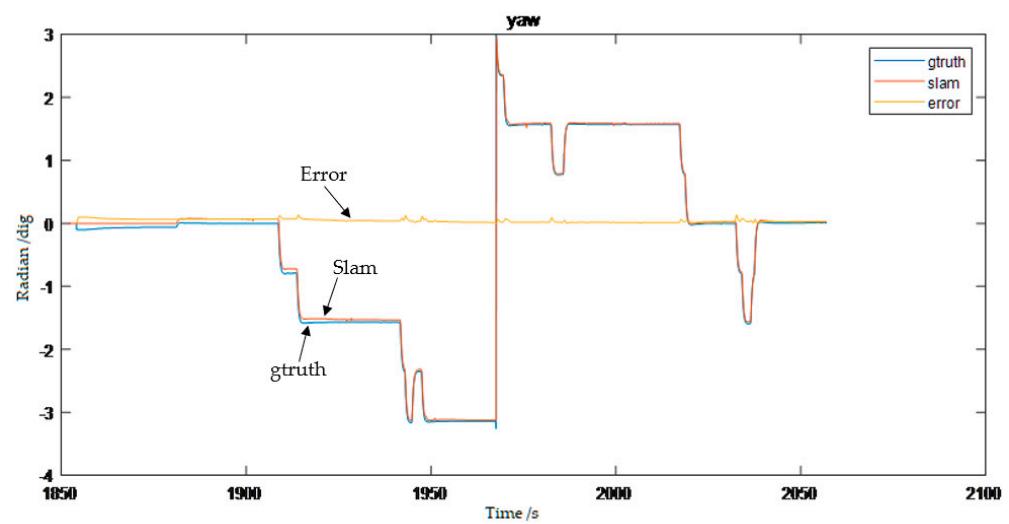
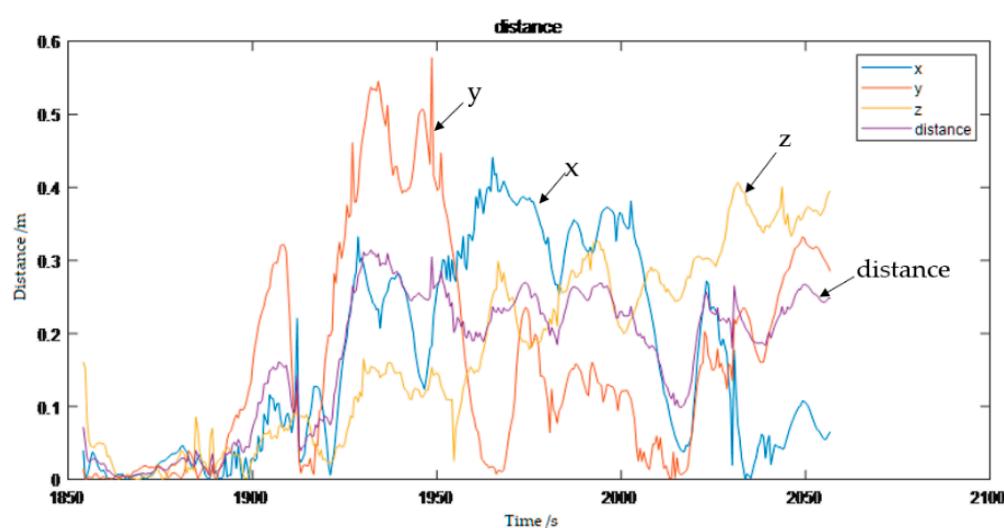


Figure 14. Comparison between SLAM and real value along yaw direction.



**Figure 15.** Diagram of position error.

## 2.7. Section Conclusion

This section introduced the detailed design of the autonomous positioning algorithm and the scene construction and simulation of the algorithm carried out in the Gazebo engine. Through the simulations, under a scene with fixed size, autonomous positioning with a certain extent of accuracy was achieved.

## 3. Detailed Design of the Map-Building and Trajectory-Planning Algorithm

### 3.1. The Introduction of the Autonomous Positioning Module

In the mapping and path planning part, the RGB-D camera was selected as a reliable source of in-depth information. An octree map with mature technology was applied to realize the construction of the three-dimensional map. The RRT\* algorithm was used to realize obstacle avoidance path planning, and finally the third order spline curve  $\beta$  was used for motion smoothing.

### 3.2. Octree Map

The point cloud information output by the RGB-D camera can be directly used to construct the point cloud map, but there are several following obvious defects in the application of a point cloud map:

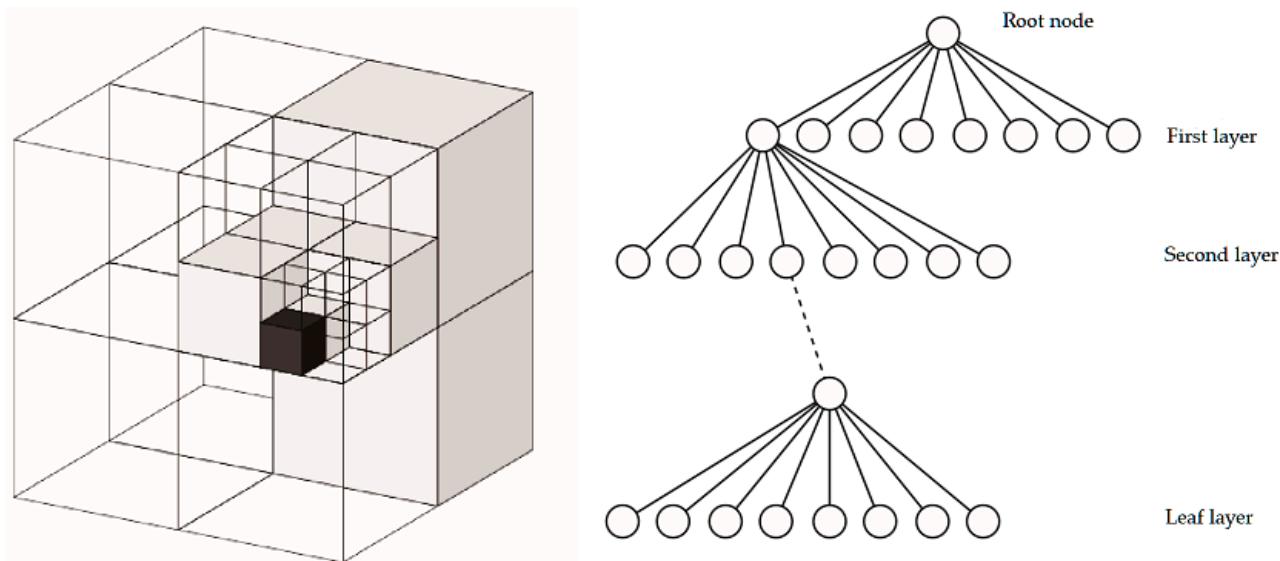
- It has a huge amount of data, and there is serious redundant storage and information redundancy.
- Point cloud maps are stored in continuous space, which means they can't be directly discretized and fast searched.
- This method cannot deal with moving objects and observation errors because we can add objects into the maps but not remove objects from maps.

In order to solve the above problems, the octree map was introduced. This map form has the advantages of flexibility, compressibility, updating and discretization.

#### 3.2.1. The Data Structure of the Octree Map

In a discrete map, it is common to model the 3D space as multiple cubes (voxels). If each facet of the cube is divided into four equally, eight sub-cubes can be obtained until the required precision is reached. If the process of expanding a cube into sub-cubes is regarded as expanding eight sub-nodes from one node, then the process of subdividing the whole space into the smallest sub-space can be regarded as an octo-tree.

The Figure 16 is the octree map structure diagram. The left one shows the process of the cube being split into sub-cubes. If the largest cube is regarded as the root node and the smallest cube as the leaf node, then the octree shown on the right can be formed.



**Figure 16.** The structure diagram of an octree.

An octree map saves storage space because of its data structure. When all the sub-nodes of a cube are occupied or not occupied, there is no need to continue to expand the node; therefore, only an empty root node is needed when a blank map begins to be established. The actual objects are most closely linked, and it is the same with blank space. Therefore, most octree nodes do not need to be expanded to cotyledon nodes, which can save a lot of storage space.

The occupation information stored in each node of the octree is expressed by the probability: 0 means completely blank and 1 means completely occupied. The initial value is 0.5. If the node is detected to be continuously occupied, the value will increase; otherwise, the value will decrease.

### 3.2.2. Node Probability Updating

According to the derivation of octree, assuming that when  $t = 1, 2, \dots, T$ , the observed data is  $z_1, \dots, z_T$ , then the information recorded by the nth leaf node is:

$$P(n | z_{1:T}) = \left[ 1 + \frac{1 - P(n | z_T)}{P(n | z_T)} \frac{1 - P(n | z_{1:T-1})}{P(n | z_{1:T-1})} \frac{P(n)}{1 - P(n)} \right]^{-1} \quad (25)$$

Since the information expressed directly by probability is too complex to be updated, the algorithm uses log-odds as an alternative description method. Set  $y \in \mathbb{R}$  as a probability logarithm,  $x$  as the probability value between 0 and 1, then the transformation between them can be described by logit transformation:

$$y = \text{logit}(x) = \log\left(\frac{x}{1-x}\right) \quad (26)$$

And its inverse transformation is shown below:

$$x = \text{logit}^{-1}(y) = \frac{\exp(y)}{\exp(y) + 1} \quad (27)$$

When  $y$  changes from  $-\infty$  to  $+\infty$ ,  $x$  correspondingly changes from 0 to 1. When  $y = 0$ ,  $x = 0.5$ , so we can judge whether a node is occupied or not by storing the value of  $y$ . When

point clouds are observed continuously in nodes,  $y$  increases by a value; when the observed node is empty,  $y$  decreases by a certain value. Transfer  $y$  to the probability space and utilize the logit inverse transformation when checking the probability.

Set a node as  $n$  and the observed data as  $z$ . The probability value of this node from the beginning to  $t$  is  $L(n|z_{1:t})$ , and the probability at  $t + 1$  is as follows:

$$L(n | z_{1:t+1}) = L(n | z_{1:t-1}) + L(n | z_t) \quad (28)$$

With this log probability, the entire octree map can be updated according to RGB-D data. If the depth of a pixel observed in the RGB-D graph is  $d$ , it means that an occupied point is observed in the space corresponding to the depth value, and there is no obstacle in the path from the camera optical center to this point.

### 3.3. Path Planning

Rapidly exploring random tree (RRT) was selected as the path planning algorithm. Traditional path planning algorithms such as the artificial potential field method, the method of fuzzy rules, genetic algorithm, neural network and simulated annealing algorithm, ant colony optimization algorithm, etc., are not suitable for the path planning of multi-degree-of-freedom robots in complex environments because they all require modeling obstacles in a certain space, and the computational complexity has an exponential relationship with the DOF of robots.

RRT effectively solves the problem of path planning under conditions of high-dimensional space and complex constraints because it avoids space modeling by detecting the collision of sampling points in the state space, avoiding the modeling of the space. The characteristic of this method is that it can search the high-dimensional space quickly and effectively and lead detection to blank areas through random sampling points in the state space and then find a planned path from the starting point to the target point, which is suitable for solving the path planning of multi-degree-of-freedom robots in complex and dynamic environments. Note that the RRT algorithm is probabilistically complete and non-optimal, and path planning only finds a feasible path, which may not be optimal.

#### 3.3.1. Basic RRT Algorithm

RRT is an efficient planning method in multi-dimensional space. It takes an initial point as the root node and generates a randomly extended tree by adding leaf nodes through random sampling [26–28]. When the leaf nodes in the random tree contain the target point or enter the target area, a path from the initial point to the target point can be found in the random tree. The workflow of a basic RRT algorithm is as follows:

Initialize the random root node  $X_{init}$ , which is the starting point of path planning.

A random number  $P$  between 0 and 1 is generated. When  $P < Prob$ , a sampling point is randomly selected from the state space as  $X_{rand}$ . When  $P > Prob$ , the target point is used as  $X_{rand}$ .

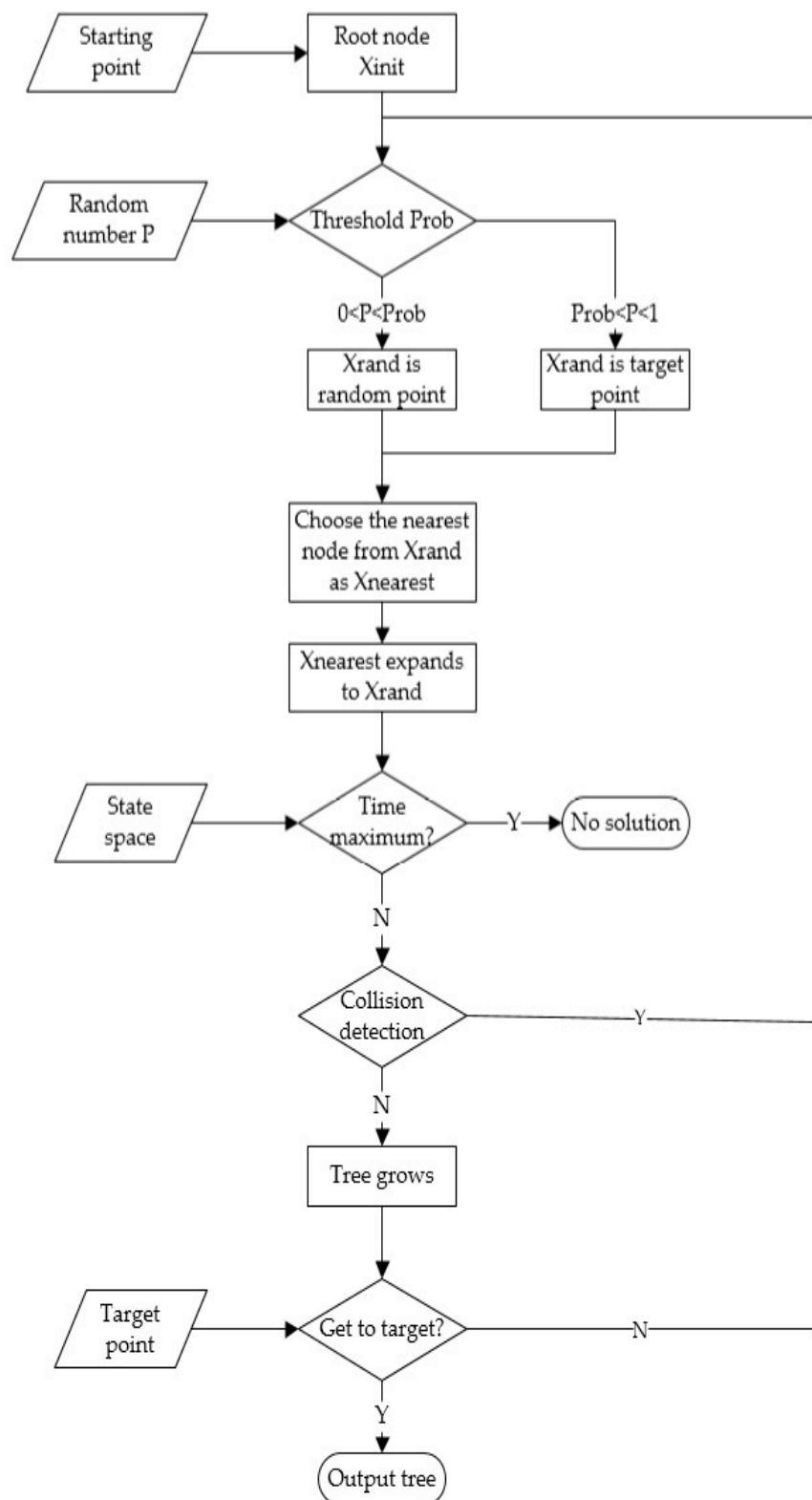
Select the nearest point from  $X_{rand}$  in random tree nodes as  $X_{nearest}$ , expand some distance from  $X_{nearest}$  to  $X_{rand}$  to obtain the new node  $X_{new}$  and the new edge  $L_{new}$ .

Record the running time: if the run times out, it returns no solution.

If  $X_{new}$  and  $L_{new}$  collide with the obstacles in the state space, return to step 2 and repeat it. If there is no collision, then run tree growth, and add  $X_{new}$  into the random tree as  $X_{nearest}$ 's leaf node.

Judge whether  $X_{new}$  is the target point or not; if it is, then output the current random tree; otherwise, return to step 2 and repeat it.

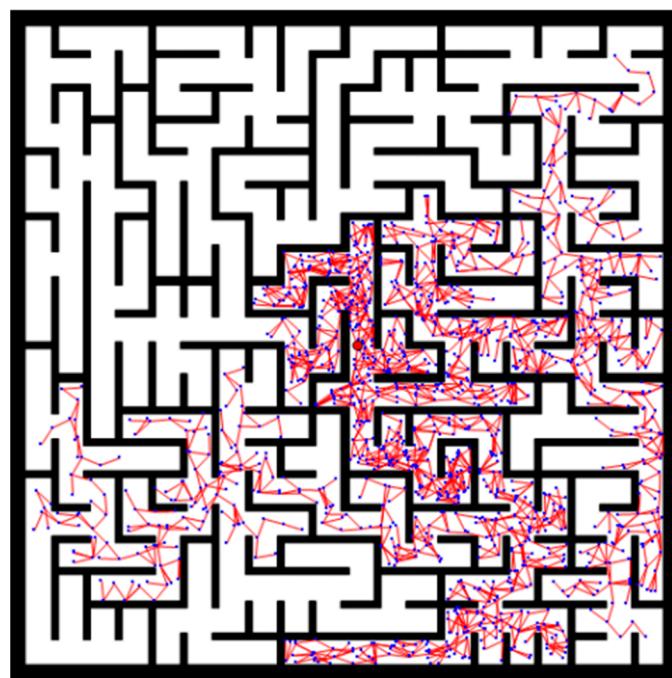
The basic RRT algorithm process is shown in Figure 17 below:



**Figure 17.** Flow of basic RRT algorithm.

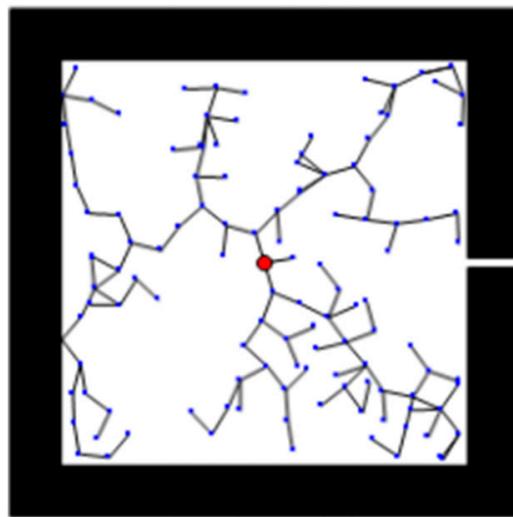
The basic RRT algorithm is not sensitive to the environment and can effectively explore the whole space. However, it also has serious disadvantages in some application conditions:

The basic RRT algorithm is a pure random search algorithm, which degrades the search efficiency significantly when the environment contains many obstacles or narrow channel constraints, as shown in Figure 18.



**Figure 18.** Performance of the RRT algorithm in a maze.

Because the narrow channel area is small, the probability of being touched is low, and this is why it is difficult to find a path in an environment with narrow passageways, as shown in Figure 19.



**Figure 19.** Performance of the RRT algorithm in an environment with narrow passageways.

Because the nodes of the RRT algorithm are completely randomly generated, the path may not be relatively smooth, and it cannot be directly applied for path and motion planning.

### 3.3.2. RRT\* Algorithm

Although RRT is a relatively efficient algorithm that can deal with path planning problems with nonholonomic constraints, and has great advantages in many aspects, the RRT algorithm can't guarantee that the obtained feasible path is relatively optimized. RRT\* was improved based on RRT, mainly by reselecting the parent node and rewiring.

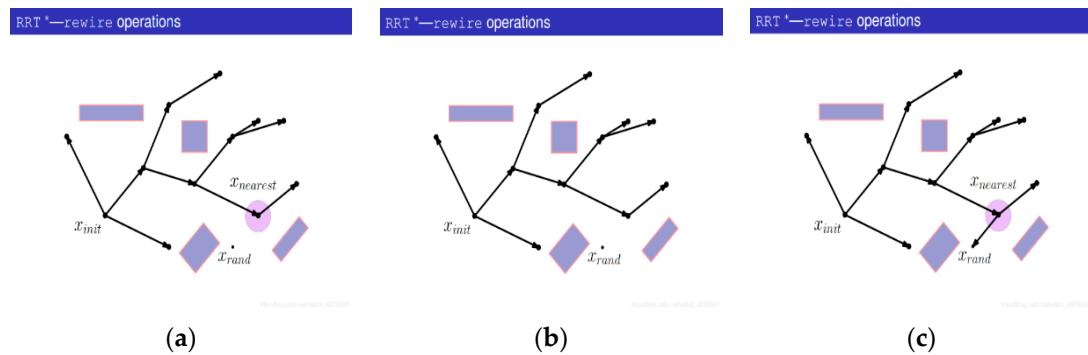
In RRT, the nearest point to Xrand is selected as the parent node in the extended node policy, but this choice is not necessarily optimal. The goal of planning is to make this point

as close as possible to the starting point. Many improvements have been achieved using RRT\* by drawing a small circle around the sampling point after it is added to the path tree and considering whether there are better parent nodes to connect to that point so that the distance from the starting point to the point is shorter (although those nodes may not be the closest points to the sampling point). If a more suitable parent is chosen, then connect them and remove the original wiring (rewiring).

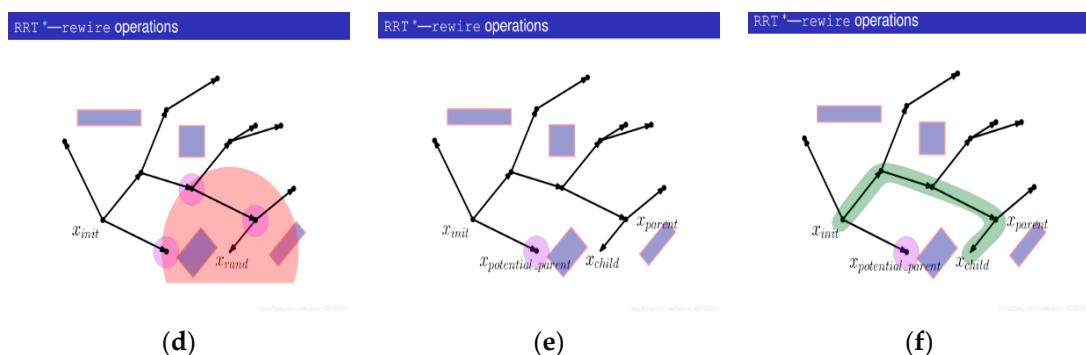
The RRT\* algorithm is asymptotically optimized, which means that the resulting path is more and more optimized with the increase of the number of iterations, and it is never possible to obtain the optimal path in limited time. In other words, it takes a certain amount of running time to get a relatively satisfactory and optimized path.

In the rewiring process, the tree structure is optimized by introducing the path length parameter to achieve the optimal path planning. The specific optimization process includes the following 15 main steps. The process and steps of rewiring are introduced as Figures 20–22:

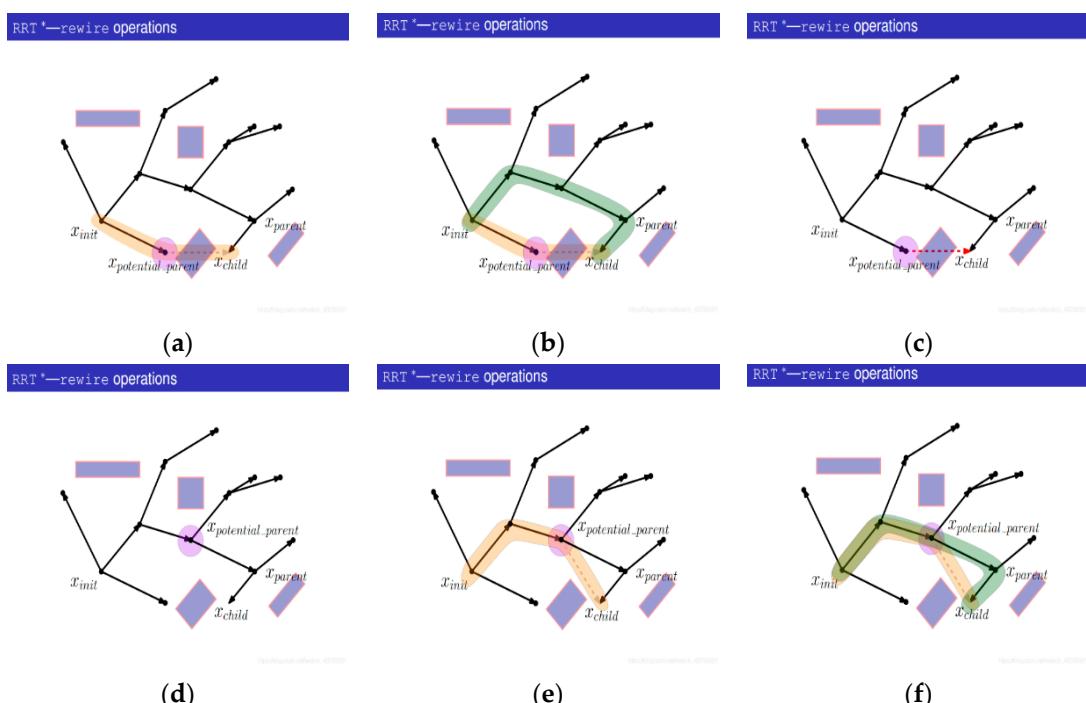
- (1) Generate a random point  $x_{rand}$ ;
- (2) Find the nearest node  $x_{nearest}$  from  $x_{rand}$  on the tree;
- (3) Connect  $x_{rand}$  with  $x_{nearest}$ ;
- (4) With  $x_{rand}$  as the center, search for nodes in the tree with a certain radius and find out the set of potential parent nodes  $\{x_{potential\_parent}\}$ . The purpose is to update  $x_{rand}$  and observe whether there is a better parent node;
- (5) Start with a potential parent,  $x_{potential\_parent}$ ;
- (6) Calculate the cost of  $x_{nearest}$  being the parent node;
- (7) Instead of performing collision detection, connect  $x_{potential\_parent}$  with  $x_{child}$  (that is,  $x_{rand}$ ) and calculate the path cost;
- (8) Compare the cost of the new path with that of the initial path. If the cost of the new path is smaller, the collision detection will be carried out; otherwise, the next potential parent node will be replaced;
- (9) If collision detection fails, the potential parent node will not act as the new parent node;
- (10) Turn to the next potential parent;
- (11) Connect the potential parent node to  $x_{child}$  (that is,  $x_{rand}$ ) and calculate the path cost;
- (12) Compare the cost of the new path with the cost of the original path. If the cost of the new path is smaller, the collision detection will be carried out; if the cost of the new path is larger, the next potential parent node will be replaced;
- (13) The collision detection passes;
- (14) Delete the previous edges from the tree;
- (15) Add a new edge to the tree, and take the current  $x_{potential\_parent}$  as the parent of  $x_{rand}$ .



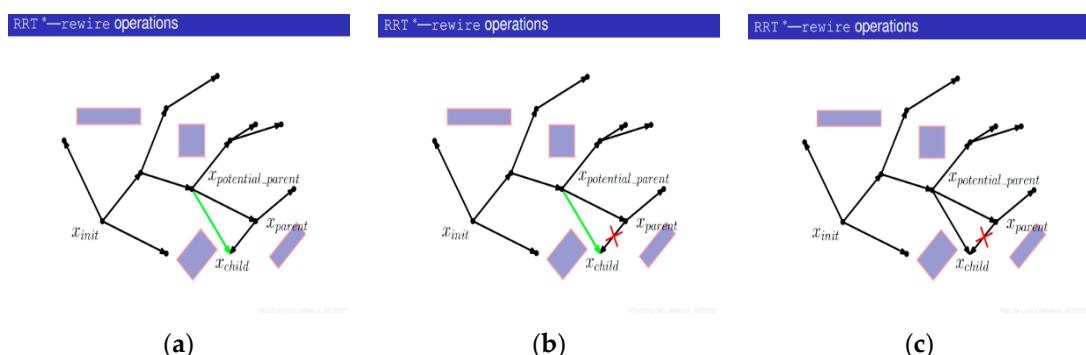
**Figure 20. Cont.**



**Figure 20.** Step (1) to (6). (a) Generate random point; (b) find the nearest node; (c) find initial parent node; (d) find potential parent nodes; (e) select potential parent node; (f) calculate the initial path cost.



**Figure 21.** Step (7) to (12). (a) Calculate the new path cost; (b) compare the cost of new path and initial; (c) failure of collision detection; (d) select new parent nodes; (e) calculate the new path cost; (f) the comparison of the cost of new and initial paths.



**Figure 22.** Collision detection passes. (a) The collision detection passes; (b) delete the previous path edges; (c) add new edges.

### 3.4. Smoothing the Interpolation of Third-Order $\beta$ Spline

Although the RRT\* algorithm improves the optimality and smoothness of the planned trajectory, it still has many sharp points and cannot be directly used for trajectory control. Here, third-order  $\beta$  spline interpolation is used to smoothen the solution of RRT\*, which can ensure the continuous acceleration control signal of the motion trajectory.

#### 3.4.1. Node Table

The node table is the key parameter to generating the basic function table, and it is strictly equal to the sum of the number of control points: the number of orders plus one. The parameters of the node table are set artificially. For  $\beta$  spline curve, there are two general ways to set it: sequential method and clamped method. The former is used to make standard  $\beta$  spline open and closed curves, and the latter is used to make a more practical  $\beta$  spline curve.

The order list only needs to be set linearly from 0 to 1, while the clamped list needs to set the nodes of each order plus 1 before and after as 0. Taking the third-order spline curve with six control points as an example, the size of its node table is  $6 + 3 + 1 = 10$ .

If it is a sequential list, we only need to set it in order:

$$0, \frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{4}{9}, \frac{5}{9}, \frac{6}{9}, \frac{7}{9}, \frac{8}{9} \quad (29)$$

If it is a clamped list, since it is the third order, the former  $3 + 1$  parameters are set as 0, the latter  $3 + 1$  parameters are set as 1, and the remaining parameters increase evenly:

$$0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1 \quad (30)$$

#### 3.4.2. Basic Function Tables

The basic function table is essentially a recursive equation, but it is also an intermediate parameter at the same time. The formula is as follows:

$$B_{i,\deg}(t) = \frac{t - knot_i}{knot_{i+\deg} - knot_i} B_{i,\deg-1}(t) + \frac{knot_{i+\deg+1} - t}{knot_{i+\deg+1} - knot_{i+1}} B_{i+1,\deg-1}(t) \quad (31)$$

where,  $t$  is the node to be interpolated;  $knot_i$  represents the  $i$ th element in the node table;  $B_{i,\deg}(t)$  is the parameter of the basic function table, whose structure is a two-dimensional array, and its meaning is the value of the  $i$ th element of the basic function table at the  $\deg$  order when the user input is  $t$ .

The recursive characteristics of the function table can be seen from (31). The current elements of the  $\deg$  order need to be calculated by two elements of the  $\deg - 1$  order. In addition,  $\beta$  spline curve algorithm requires that when the function table returns to order 0, it can be calculated according to the following formula:

$$B_{i,0} \begin{cases} 1 & knot_i \leq t \leq knot_{i+1} \\ 0 & knot_i > t \text{ or } knot_{i+1} < t \end{cases} \quad (32)$$

#### 3.4.3. Calculation

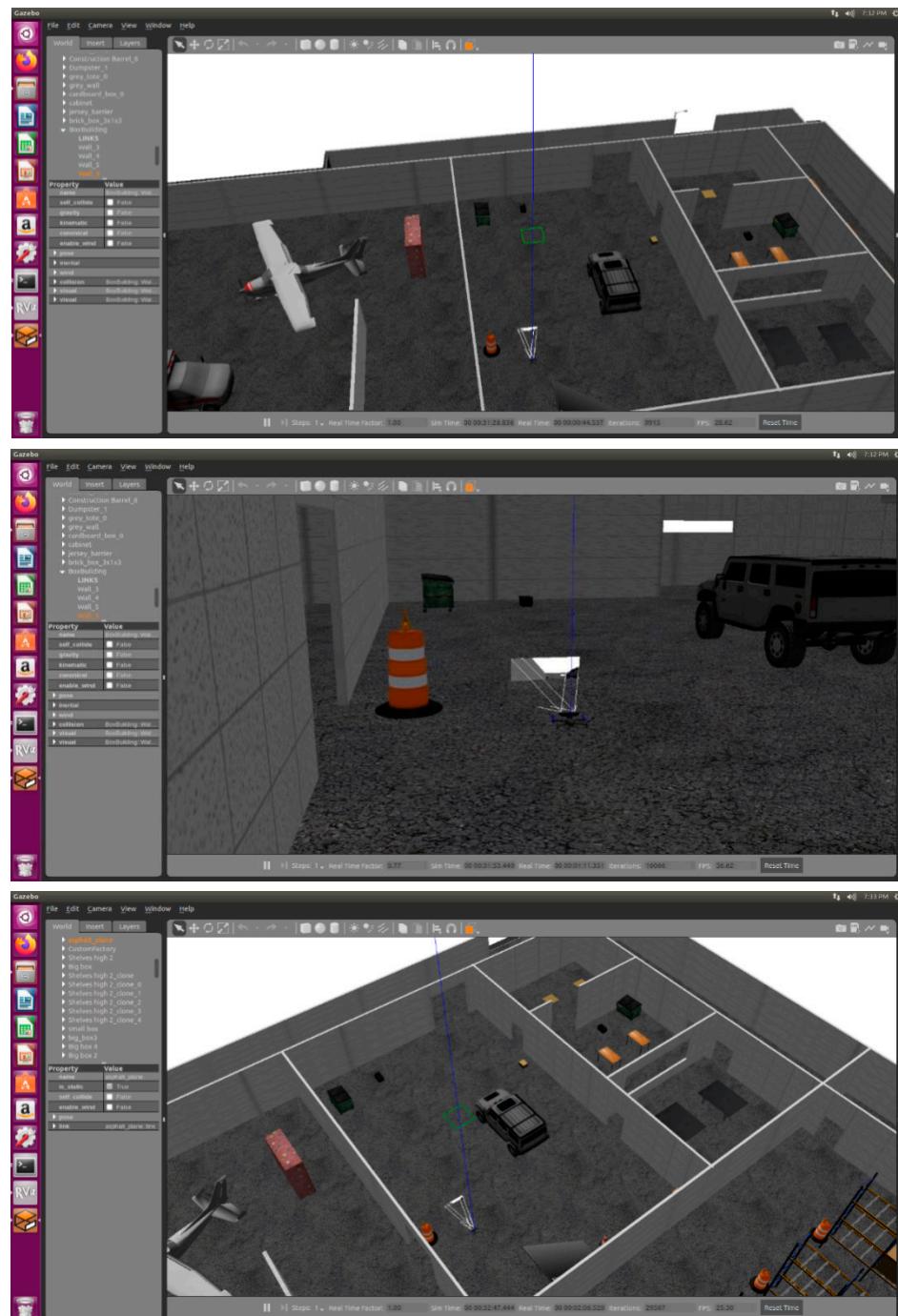
Assuming that the corresponding position of the  $t$  value in the  $\beta$  spline curve is  $C(t)$  finally, the calculation formula of the final  $\beta$  spline curve is:

$$C_t = \sum_{i=0}^{n-1} B_{i,\deg}(t) P_i \quad (33)$$

where  $B_{i,\deg}(t)$  is the value of the  $i$ th  $\deg$  order basic function table at  $t$ , and  $P_i$  is the  $i$ th interpolation control point.

### 3.5. Simulation Test and Analysis

The simulation was carried out in the Gazebo engine. A UAV PX4 with a built-in IRIS platform was selected and carried a RealSense D435I depth camera. The simulation environment is shown in Figure 23:



**Figure 23.** Screenshots of mapping simulation environment.

In the map building test, the UAV control system adopted the default parameters of the simulation system, and the VI-SLAM system constructed in Section 3 was adopted as the positioning system. The results of map building of the simulation environment are shown in Figure 24:

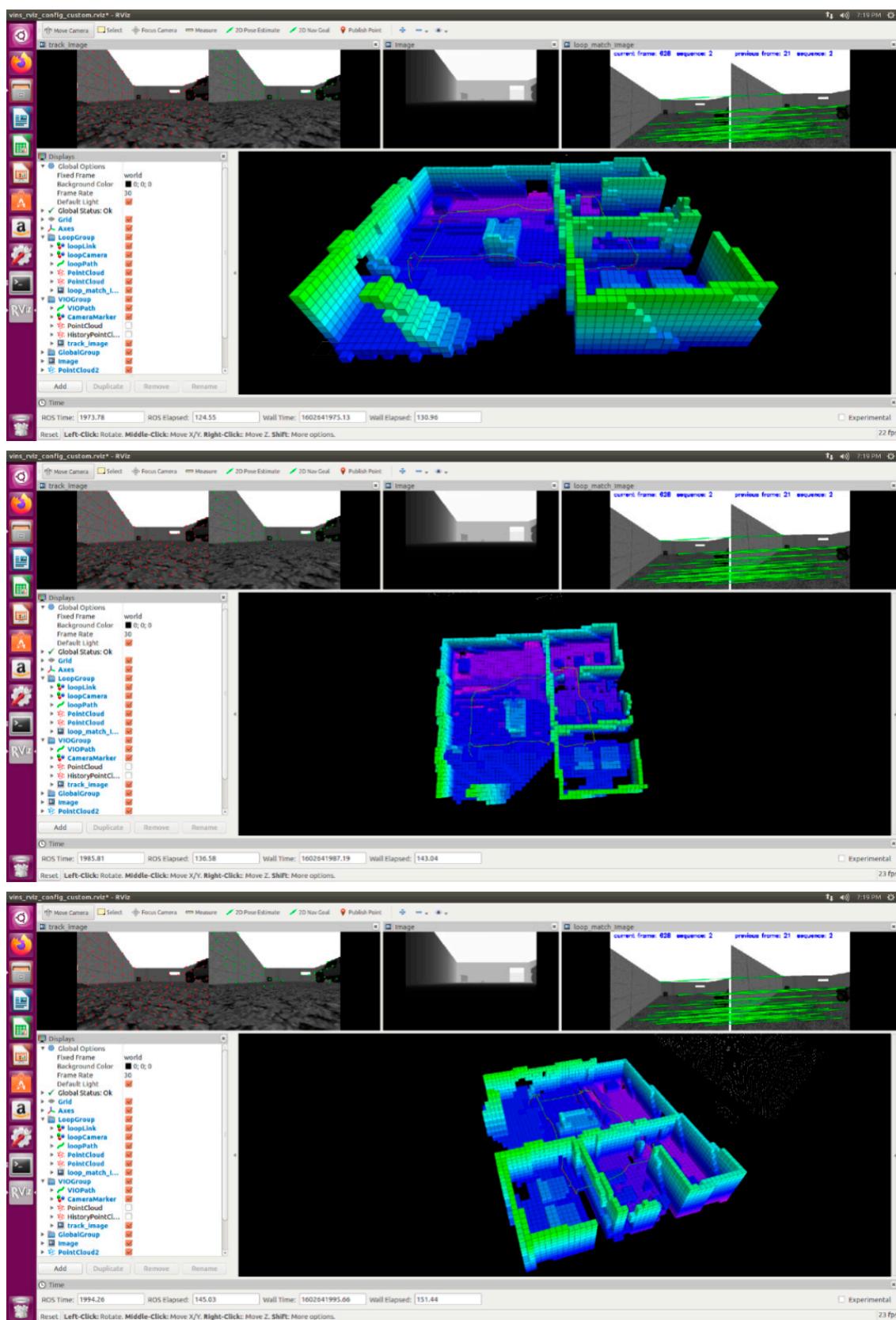
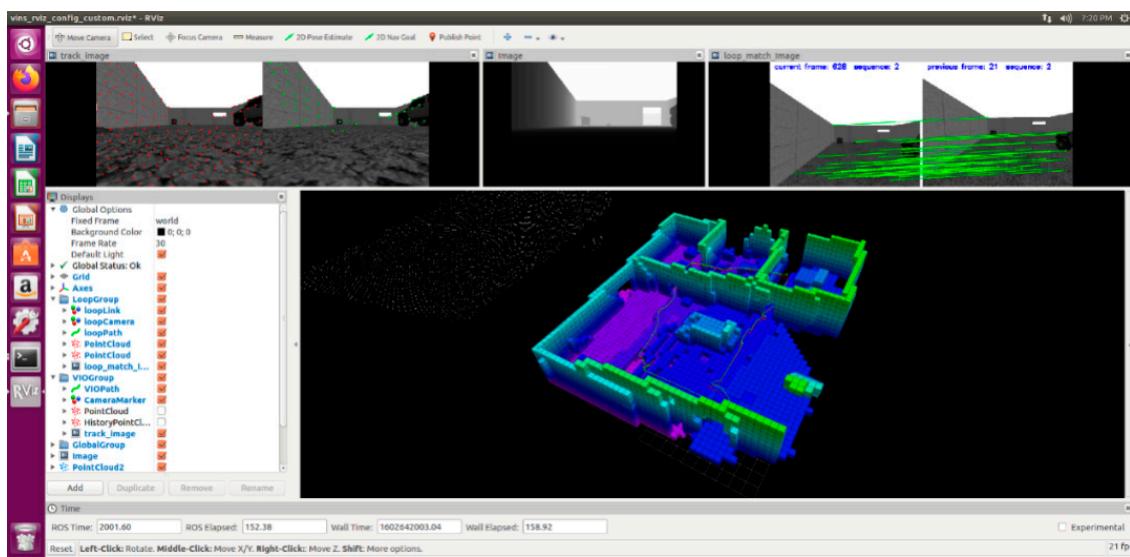
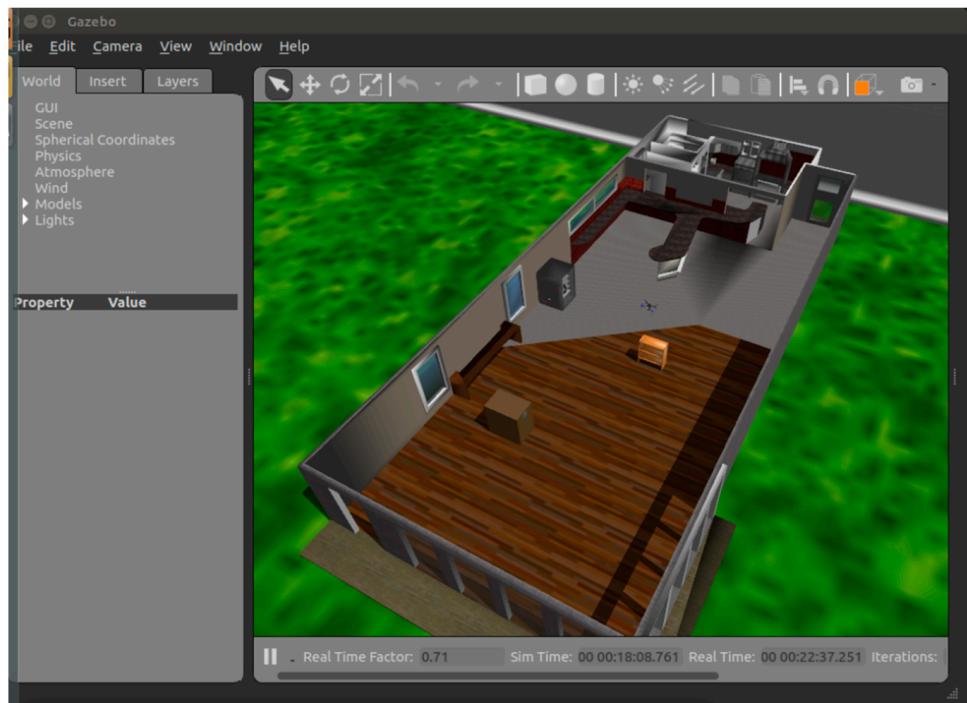


Figure 24. Cont.



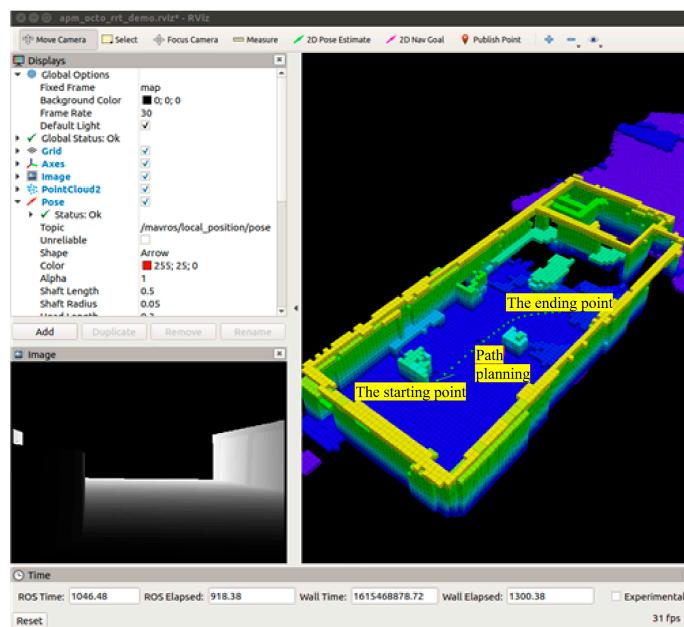
**Figure 24.** Map building results.

In the path planning test, the UAV control system adopted the default parameters of the simulation system, and the VI-SLAM system constructed in Section 3 was chosen as the positioning system. The path planning environment is shown in Figure 25:

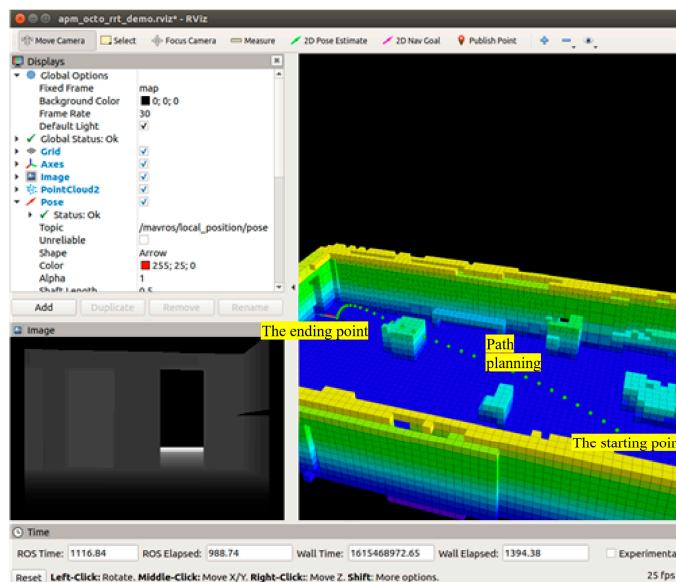


**Figure 25.** The path planning simulation environment.

Through path planning, the UAV can autonomously avoid obstacles in indoor environments and reach the target location. Parts of the path planning results are shown in Figures 26 and 27:



**Figure 26.** Path planning working condition 1.



**Figure 27.** Path planning working condition 2.

### 3.6. Section Conclusion

In this section, the map building and path planning algorithms were introduced in detail, and the above two algorithms were verified by the Gazebo engine. The results show that the proposed algorithm can fulfill the task requirements well.

## 4. The Detailed Design of the Target Detection and Recognition Algorithm

### 4.1. The Introduction of Target Detection and Recognition Module

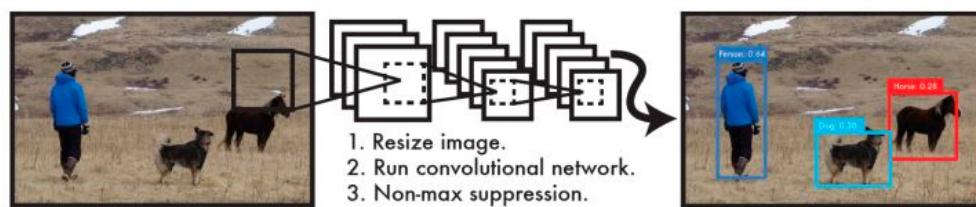
Since there is no specific cooperation target for detection, the recognition algorithm to be selected needs to be commonly applicable, transferable, and robust. At the same time, the algorithm should be optimized and accelerated under limited performance of the airborne processor to ensure that the high-speed UAV can accurately capture the target [29,30].

Therefore, the Jetson series GPU development board XavierNX was finally selected as the computing platform, the YOLOv3 network was used as the basic detection algorithm, and TensorRT architecture was introduced to achieve GPU inference acceleration.

#### 4.2. Target Detection Network

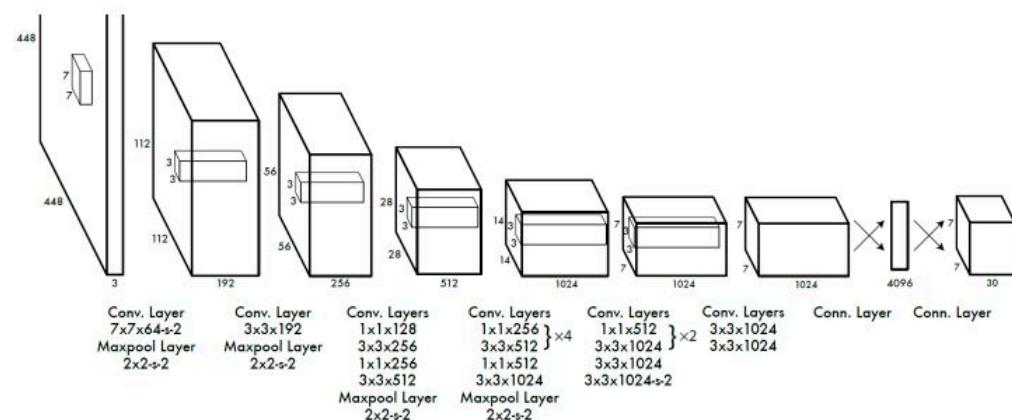
At present, target detection algorithms can be divided into two categories according to the process. One of them is the region-convolutional neural network (R-CNN) algorithm based on region proposals such as R-CNN, fast R-CNN, etc. These algorithms are two-stage methods, which require the use of heuristic methods or convolutional neural networks to generate candidate regions, and after that the regions need classification and regression. The other category is the one-stage algorithm, e.g., YOLO and SSD, which uses a unified convolutional neural network structure to perform regressive prediction for the location and categorization of targets at the same time. The first kind of method is characterized by high accuracy but slow speed, and the second kind of algorithm runs fast with low accuracy.

Overall, the YOLO algorithm is an end-to-end target recognition network, using a separate full-convolution neural network model, and its workflow is shown in Figure 28: Firstly, the resolution of the input images should be unified as  $448 \times 448$ , then put the images into the convolution neural network, finally these images are processed by the output part, and obtain the location and category information of the target. Compared with R-CNN and other two-stage algorithms, its structure is more concise and unified, with faster processing speed and easier hardware acceleration. At the same time, the YOLO training and processing courses are both end-to-end, and the available network can be directly obtained from the image training set.



**Figure 28.** The workflow of the YOLO target detection network.

In terms of network structure, YOLO uses a unified convolutional neural network sequence to process images and obtain feature sequences, and then uses a shallow convolutional neural network to perform position regression and category prediction. The specific network structure is shown in Figure 29.



**Figure 29.** YOLO network structure.

In terms of the internal structure of the network, a  $3 \times 3$  convolution kernel is mainly used for feature extraction and abstraction, a  $1 \times 1$  convolution kernel is used for cascade

cross-channel parameter pooling, and a LeakyReLU function is adopted as the activation function:  $\max(x, 0.1x)$ . Note that the activation function at the last layer of the network is replaced with a linear one. The final output of the network is a tensor whose size is  $7 \times 7 \times 30$ , where  $S = 7$  is the number of grids, the first 20 elements in the third dimension represent the degree of confidence of the 20 classifications, elements 21–22 are the degree of confidence of the bounding box  $B = 2$ , and the last 8 elements are the  $(x, y, w, h)$  of the bounding box  $B = 2$ .

The main features of the YOLO target detector are as follows:

#### (1) Features extraction network

Although the YOLOv1 network adopts the structure of the GoogLeNet classification network; it uses  $1 \times 1$  and  $3 \times 3$  CNN networks in feature extraction to lower the dimensionality of high-dimensional information and realize the information integration of high and low channels in the network. In the main part of feature extraction, YOLOv2 uses the multi-scale feature fusion method of the single shot multi-box detector (SSD) network and proposes to use the DarkNET-19 network to improve the fine-grained feature extraction in images. Since YOLOv2 only performs feature fusion in the latter layer and produces fixed-size feature maps, this method easily leads to the loss of most fine-grained information in the fusion process of high and low semantics. Thus, YOLOv2 has a poor detection effect for intensive small targets. While maintaining the detection speed, YOLOv3 adopts the simplified residual basic module to replace the  $1 \times 1$  and  $3 \times 3$  modules in the original CNN, and a deeper DarkNET-53 network is constructed as the feature extraction backbone network of YOLOv3.

#### (2) Residual mechanism

The YOLOv2 feature extraction in DarkNET-19 uses a straight tube network structure such as GoogLeNet or visual geometry group (VGG). Convolution is directly added in DarkNET-19 to deepen the network to realize the purpose of extracting more useful feature information by convolution network. This easily leads to the disappearance or explosion of the loss gradient in the network learning training process. For this reason, YOLOv3 in DarkNET-53, drawing on the concept of ResNet, uses a residual module to achieve the superposition of the output feature map of convolution with the input to solve the contradiction between network depth and gradient disappearance.

#### (3) Feature map

In the network, before YOLOv3 outputs the feature map, a method combining the feature pyramid network (FPN) and upsampling is proposed based on the FPN method in SSD, which improves the problem of the loss of fine-grained target feature information in the fusion of multiple high-level information and low-level information in the feature map. The basic idea of this method is: based on the current feature map, the upsampling method is used to concatenate the output features of a convolution layer into a new feature map. This structure can not only improve the feature richness of fine-grained targets, but also help the algorithm to improve the accuracy of target prediction.

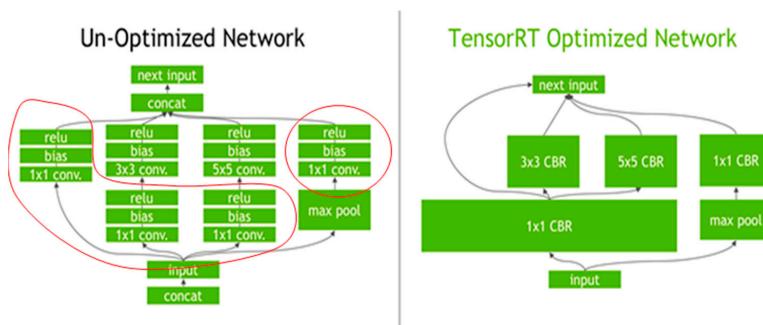
### 4.3. TensorRT Inference Acceleration

NvidiaTensorRT, formerly known as the graphics processing unit (GPU) inference engine (GIE), is a high-performance deep learning inference optimizer that can provide low-latency and high-throughput deployment inference for deep learning applications. TensorRT can be used to accelerate reasoning for exceedingly large-scale data centers, embedded platforms or autonomous driving platforms. TensorRT can now support almost all deep learning frameworks such as TensorFlow, Caffe, Mxnet, Pytorch and so on. Combining TensorRT with NVIDIA GPU, a fast and efficient deployment inference can be realized in almost all frameworks. TensorRT is currently the only programmable inference accelerator that can build and optimize customized network structures in addition to its on-premise network structure, so it can adapt to existing network structures and ones in the near future.

TensorRT has the following optimization methods, the most important of which are the first two kinds of adjustment to the network operation structure:

#### (1) Interlayer fusion and tensor fusion

Taking a GoogleNetInception calculation as an example, the left part of Figure 30 below is the calculation chart. There are many layers in this structure. During the deployment of model inference, the calculations of each layer are completed by the GPU, but in the actual computing process, the GPU starts different compute unified device architecture (CUDA) cores to complete the computation. The computing speed of the CUDA core tensor is very fast, and the operation time mainly consists of the slow core startup and read and write processes of the tensor, which cause a large amount of occupation of memory bandwidth and a waste of GPU computing resource. By the transverse and longitudinal merger between layers (the merged structure of convolution, bias and ReLU layers are fused to form a single layer called case-based reasoning (CBR)), the number of layers in the network is greatly reduced while maintaining the original functions. Lateral merging can combine convolution, offset and activation layers into a CBR structure. Vertical merging can combine layers with the same structure but different weights into a wider layer. Both operations can make the optimized structure occupy only one CUDA, and reduce the number of data transfers, memory bandwidth occupation and the time of core start and stop.



**Figure 30.** TensorRT optimization model.

The combined calculation is shown on the right side of Figure 30. Fewer computation layers lead to less occupation of CUDA cores, and the entire model structure is more compact and efficient.

#### (2) Data accuracy calibration

Most deep learning frameworks train neural networks with tensors at full 32-bit precision (FP32), and once the network is trained, the data length can be reduced to speed up the access and operation because backpropagation is not needed in the process of deploying inference. TensorRT supports FP16 and INT8 data compression acceleration modes. TensorRT provides a fully automatic calibration process to lower FP32 accuracy to INT8 accuracy with best matching performance and minimize performance loss.

#### (3) CUDA core optimization

In the inference calculation of the network model, the CUDA core of GPU is called for calculation. TensorRT can adjust CUDA kernel for different algorithms, different network models, and different GPU platforms to ensure the optimal calculation performance of the current model on a specific platform.

#### (4) Tensor memory management

Due to the characteristics of the neural network itself, the size of each feature map remains the same during its operation, so the video memory can be allocated in advance. TensorRT assigns certain video memory for every tensor during application, reducing memory usage and improving reuse efficiency.

#### (5) GPU multi-stream optimization

For bypass networks that cannot be merged, GPUs generally adopt multi-stream computing and then perform stream synchronization. TensorRT can optimize the flow operation from the hardware aspect to achieve the optimal synchronization effect.

#### 4.4. Analysis Test

The acceleration performance test was performed on a JetsonXavierNX computer. In 15 W working mode, the CPU (no optimization), GPU (CUDA optimization), and GPU (TensorRT optimization) were used for the test. The relative parameters of the device are shown in Table 2.

**Table 2.** JetsonXavierNX performance parameters.

Ability	10 W Mode	15 W Mode
AI performance	14 TOPS (INT8)	21 TOPS (INT8)
GPU	384-core NVIDIA Volta™ GPU with 48 Tensor cores	
GPU max freq	800 MHz	1100 MHz
CPU	6-core NVIDIA Carmel ARM® v8.2 64-bit CPU 6 MB L2 + 4 MB L3	
CPU max freq	2-core @ 1500 MHz 4-core @ 1200 MHz	2-core @ 1900 MHz 4/6-core @ 1400 Mhz
Memory	8 GB 128-bit LPDDR4x @ 1600 MHz	51.2 GB/s
Storage	16 GB eMMC 5.1	
Power	10 W   15 W	

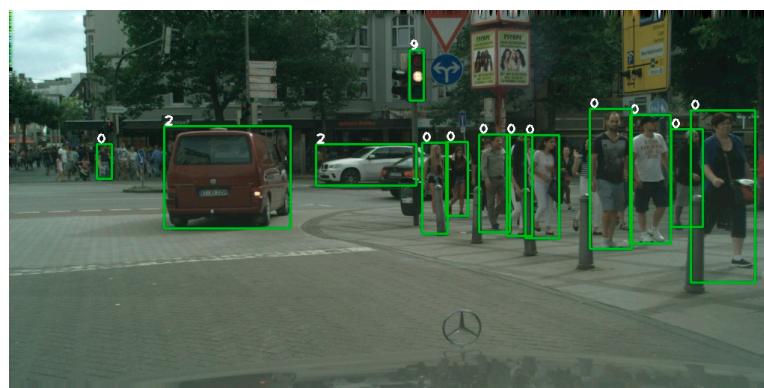
The test used  $1280 \times 720$  resolution images to identify 1000 groups and average the time. Since CUDA and TensorRT require pre-start of the CUDA core, the time in this section was recorded separately. The results of the speed test are shown in Table 3:

**Table 3.** Results of processing speed using different computing platforms.

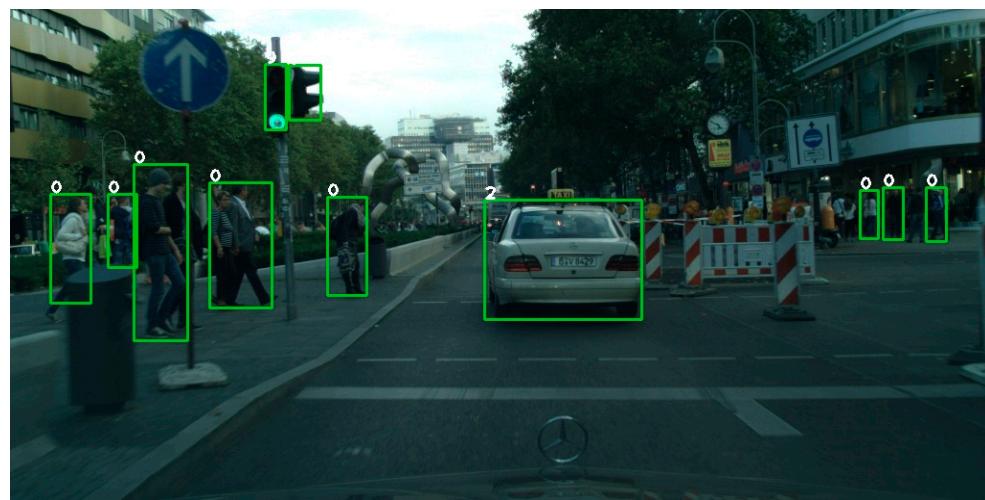
Computing Platforms	Initialization/ms	Average Time/ms
CUP	-	790
GPU(CUDA)	2310	85
GPU(TensorRT)	1103	12

After optimization, the network inference speed was greatly improved, about 66 times as fast as CPU inference, and about 7 times as fast as the CUDA inference, finally reaching about 83FPS. At the same time, due to the simplified network structure, the CUDA core startup process was accelerated by about two times after TensorRT optimization.

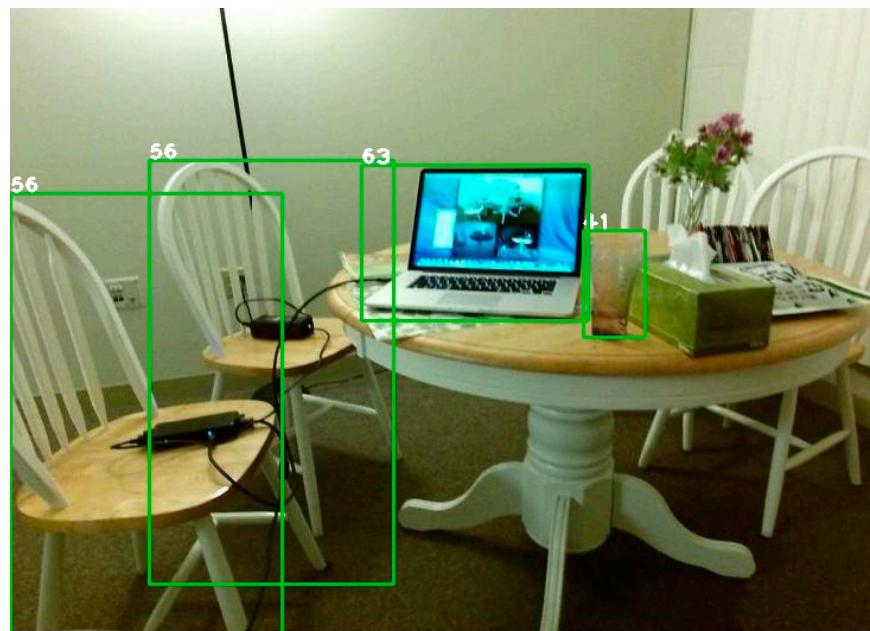
Part of the recognition results are shown in Figures 31–33:



**Figure 31.** Recognition result 1.



**Figure 32.** Recognition result 2.



**Figure 33.** Recognition result 3.

#### 4.5. Section Conclusion

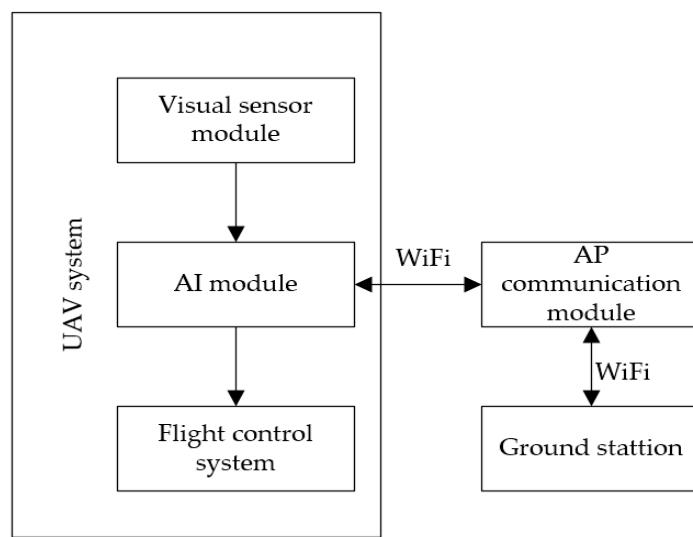
In this section, the target recognition algorithm was introduced in detail, and the algorithm was verified on the corresponding equipment. The results show that YOLO can realize the recognition of targets with high precision and accuracy.

### 5. Technical Validation

Based on the system mentioned above, an indoor UAV platform that is applicable for indoor environments was built for this paper, which adopted the following plan:

#### 5.1. Introduction of the Platform Plan

The platform consisted of four parts, including the body part, the autonomous navigation system, the ground control system, and the data transfer system. The overall structure is shown in Figure 34:



**Figure 34.** Diagram of overall structure.

According to the content in the figure, the aircraft platform built is shown in Figure 35:



**Figure 35.** Flight platform.

The subsystems are described as follows.

#### 5.1.1. The Fuselage Part

The body part was composed of power system, frame, and flight control system, which are described as follows.

##### (1) Power system

The power package was an Air Gear 450, manufactured by Tmotor, as shown in Figure 36:



**Figure 36.** Air Gear 450 power package.

(2) Frame design

The frame was assembled using carbon fiber with aluminum alloy CNC parts.

(3) Battery

The Leopard 4S-6000mah was adopted as the battery, and its discharge rate is 60 C, as shown in Figure 37.



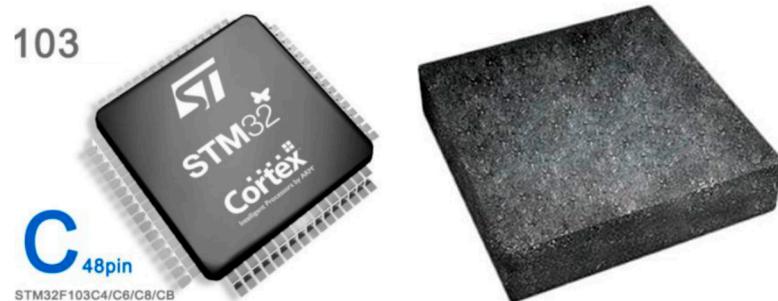
**Figure 37.** Leopard battery.

(4) Flight control system

The flight control system adopted the self-developed flight control module, as shown in Figure 38, whose detailed parameters are as follows:

Main control: STM32F103@72MHz frequency;

IMU: ICM20689\*2.



**Figure 38.** Flight control IC.

Since IMU was used as the underlying module, two IMU were installed face-to-face to suppress the gyro drift.

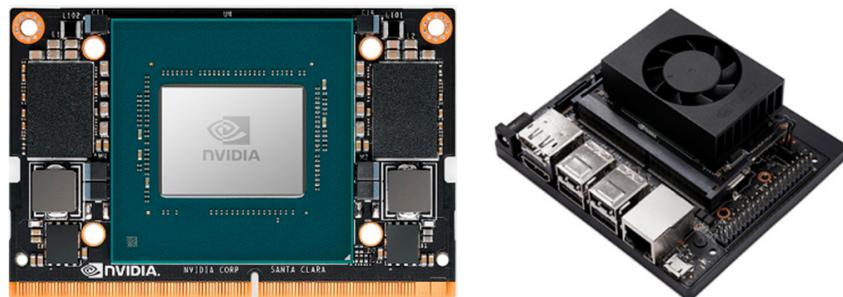
For detailed technical parameters, see the datasheet of IMU and the main control unit.

### 5.1.2. Autonomous Navigation System

The autonomous navigation system took an NVIDIA module as the processing core and an Intel camera as the sensor.

(1) Core processing unit

The NVIDIA Jetson Xavier NX was introduced as the processing core unit, as shown in Figure 39.



**Figure 39.** Jetson XAVIER NX module.

## (2) Sensor

The sensor used an Intel RealSense Camera D435i and T265 as the vision sensing module. D435i was used to provide depth information, and its performance is shown in Figure 40:



**Figure 40.** D435i camera.

T265 provides SLAM mapping information, as shown in Figure 41. The T265 contains two fisheye lens sensors, an IMU, and a Movidius Myriad 2 VPU. The camera enjoys low delay and very efficient power consumption. Through extensive performance tests and validation, under expected application conditions, the closed-loop offset was less than 1%. The delay between the pose action and the action reflection was less than 6 milliseconds.



**Figure 41.** T265 camera.

### 5.1.3. Data Transfer System

The Huawei AP6750-10T was adopted as data transfer system, as shown in Figure 42:



**Figure 42.** AP data transfer system.

Its performance indicators are as Table 4.

**Table 4.** Autonomous navigation system.

Model	AP6750-10T
Type	Distributed wireless router
Wireless standard	IEEE 802.11 a/b/g/ac/ac wave2, support 2×2MIMO
Wireless rate	3000 Mbps
Working frequency range	2.4 GHz, 5 GHz
Support agreement	802.11a/b/g/n/ac/ax
Software parameters	
WPS support	Supports WPS one-click encryption
Safety performance	
Support Open System authentication Support WEP authentication, and support 64-bit, 128-bit, 152-bit, and 192-bit encryption bytes Support wap2-psk Support wpa2-802.1x Support wpa3-sae Support wap3-802.1x Support wap-wpa2 Support wap-wpa3	
Internet management	
Support IEEE 802.3ab standard Support sub-negotiation of rate and duplex mode Compatible with the IEEE 802.1 q Support NAT	
Qos support	
Based on the WMM, it supports the WMM power saving mode, uplink packet priority mapping, queue mapping, queue mapping, VR/ mobile game application acceleration, and hierarchical HQoS scheduling for airports.	
Hardware parameters	
Local network interfaces	2 × 10 GE electrical interface, 1 × 10 GE SFP+
Other interfaces	one
Type of antenna	Built-in type
Working environment	Temperature: −10~+50 °C

#### 5.1.4. Ground Control System

The ground control system was coded by Qt software, as shown in Figure 43.



**Figure 43.** Qt software for coding ground control system.

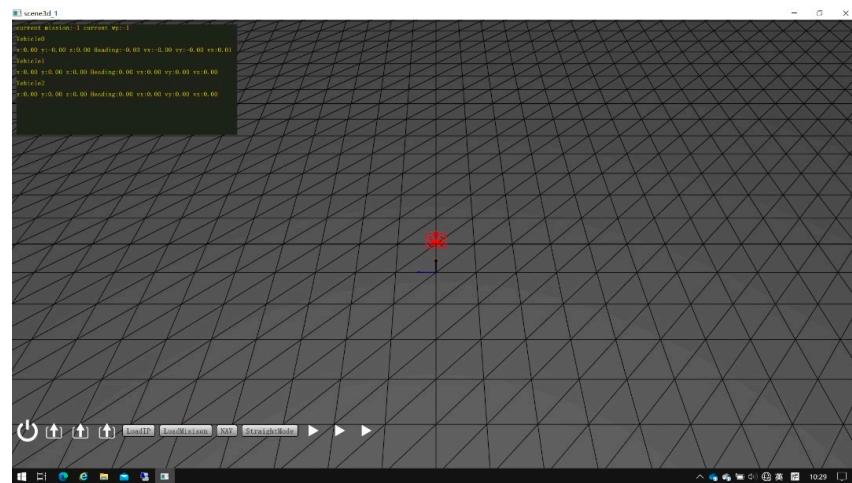
## 5.2. Flight Test

### 5.2.1. Performance Test

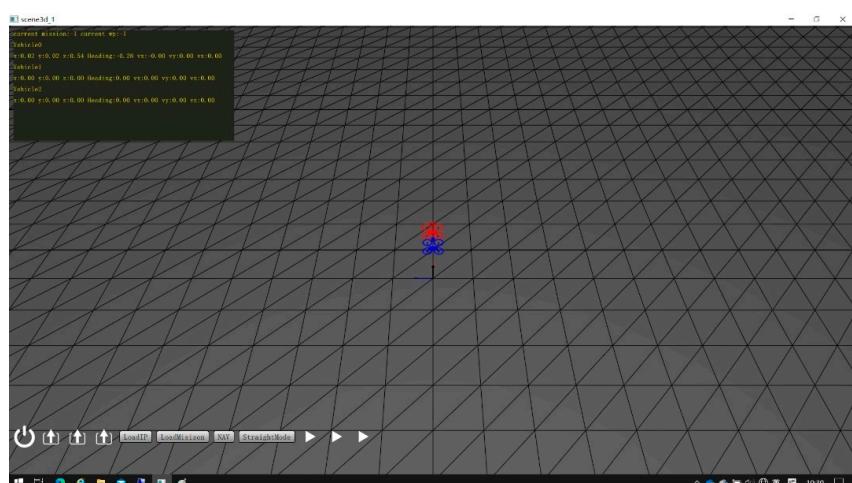
#### (1) Positioning accuracy of integrated navigation

Four reference points were used for accuracy comparison, which were  $(0, 0, 0)$ ,  $(0, 0, 0.51)$ ,  $(2, 0, 0.51)$  and  $(2, -1, 0.51)$ .

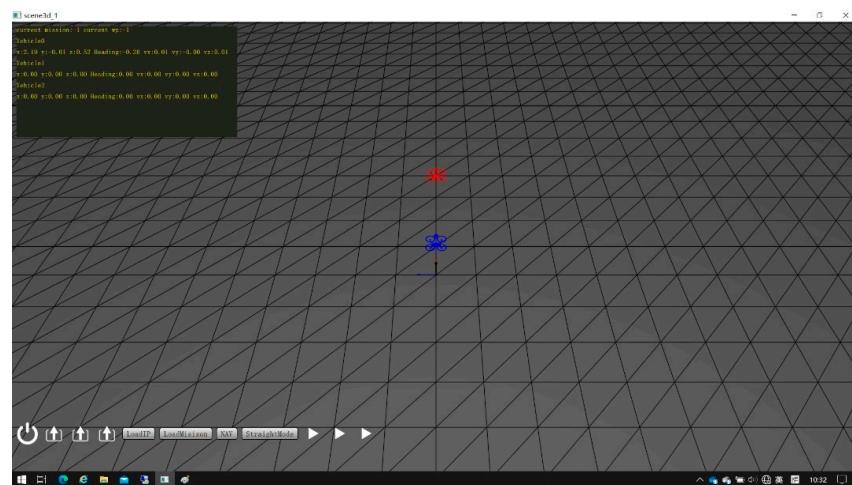
When the UAV was placed at the above four points, the corresponding navigation position was obtained, as shown in Figures 44–47:



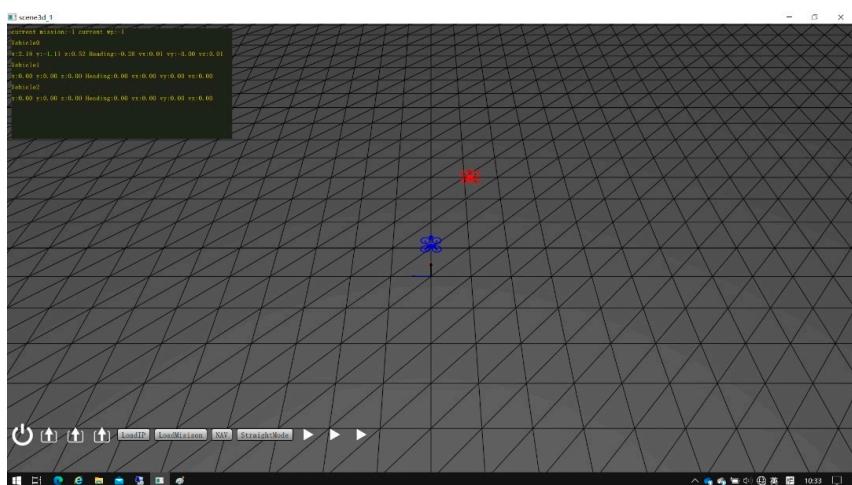
**Figure 44.**  $(0, 0, 0)$  position navigation data map.



**Figure 45.**  $(0, 0, 0.5)$  position navigation data map.



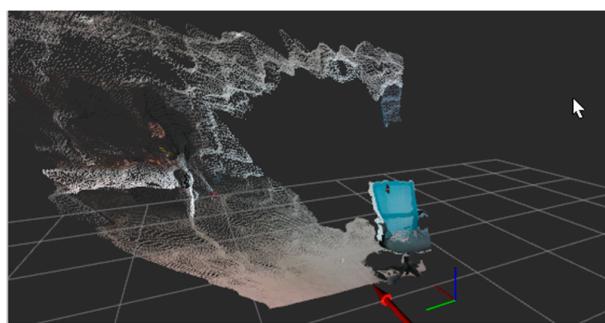
**Figure 46.** (2, 0, 0.5) position navigation data map.



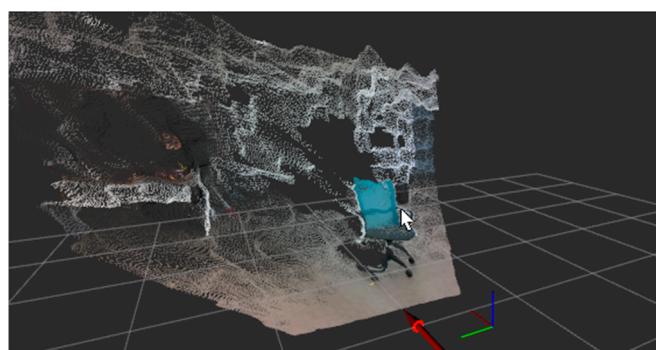
**Figure 47.** (2, −1, 0.5) position navigation data map.

The actual four-point navigation positions were (0, 0, 0), (0.02, 0.02, 0.54), (2.19, −0.01, 0.52), (2.18, −1.11, 0.52). According to the above results, it can be seen that this meets the requirements of the actual conditions, namely:

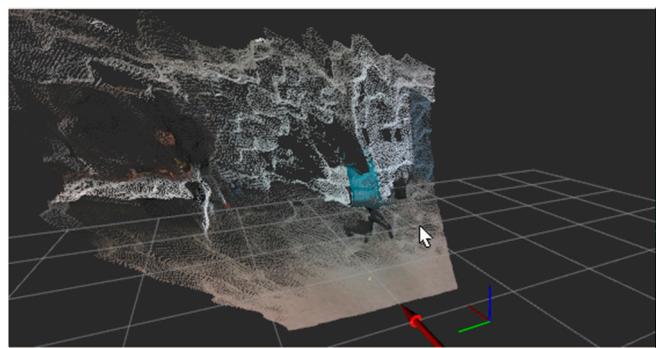
- Combined navigation and positioning accuracy (CEP):  $\leq 0.2$  m;
  - Fixed-altitude, fixed-point flight accuracy (CEP):  $\leq 0.5$  m (RMS).
- (2) Obstacles (stools) were directly placed 1 m, 2 m, 3 m, 4 m and 5 m away in front of the UAV for evaluation, respectively. The measurement results are shown in Figures 48–52:



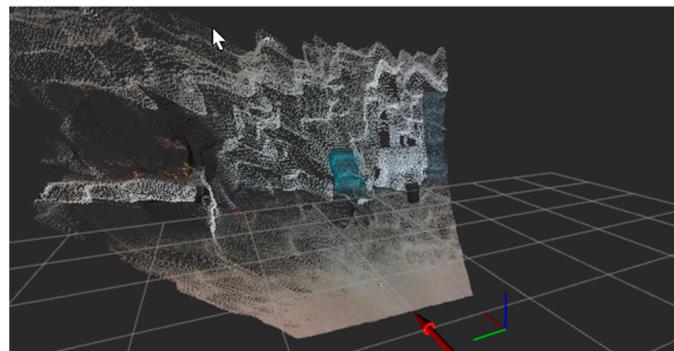
**Figure 48.** Resultant figure when obstacle was 1 m away.



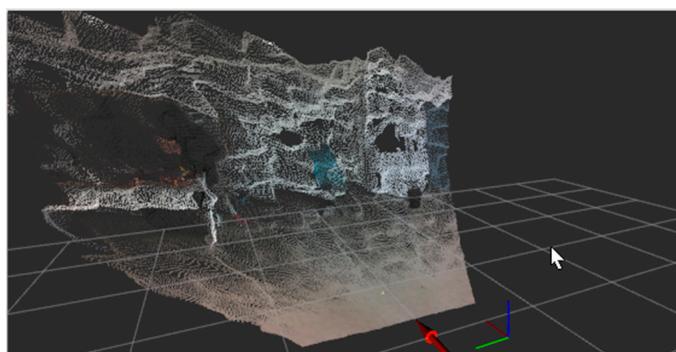
**Figure 49.** Resultant figure when obstacle was 2 m away.



**Figure 50.** Resultant figure when obstacle was 3 m away.



**Figure 51.** Resultant figure when obstacle was 4 m away.



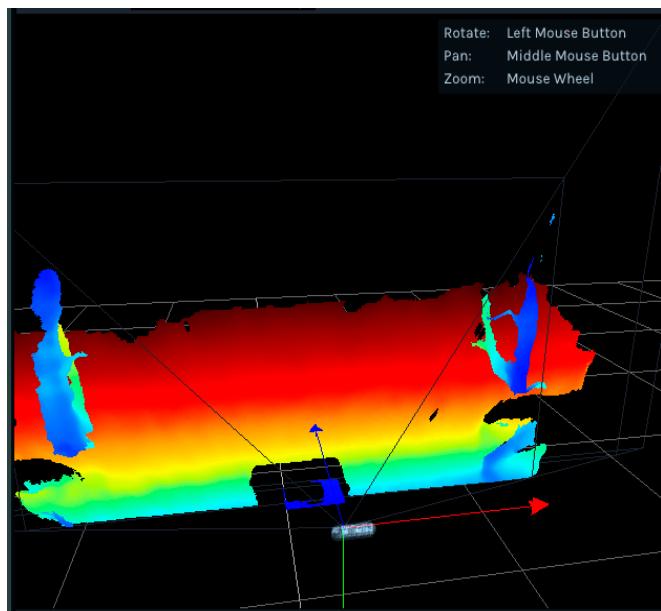
**Figure 52.** Resultant figure when obstacle was 5 m away.

From the above results, the system meets the requirements of the technical requirements, namely:

Obstacle detection distance:  $\geq 5$  m.

### (3) Obstacle detection channel and range

Because T265 was used as the obstacle measurement equipment, its field of view (FOV) range was  $163 \pm 5^\circ$ , and the effect is shown in Figure 53:



**Figure 53.** Results of obstacle measurement range.

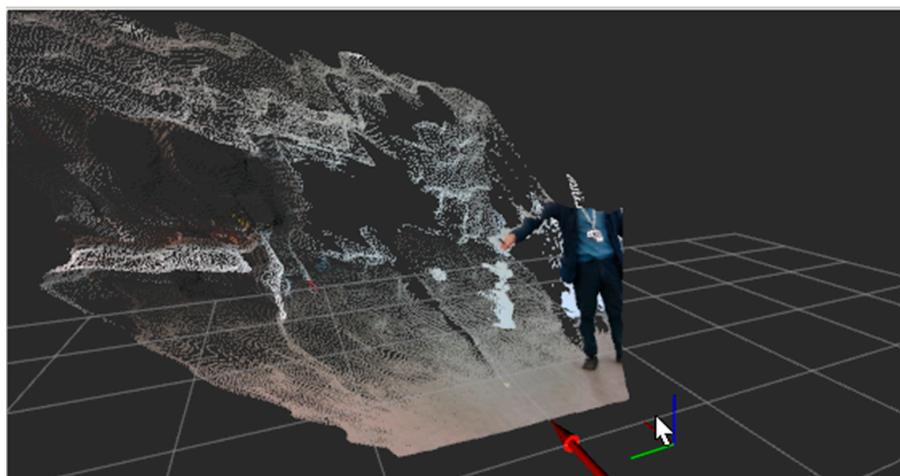
It can be seen from the results that the system meets the obstacle detection channel and range requirements, namely:

The system possesses detection ability in at least three channels; namely, the front, the top and the bottom. The detection range in each channel ( $\geq \pm 45^\circ$  horizontally,  $\geq \pm 45^\circ$  vertically).

### (4) Minimum size of detectable obstacle

The ruler was placed 5 m away from the camera to measure the recognition of obstacle size.

It can be seen from Figure 54 that the system can complete the recognition of obstacles. That is, at 2 m distance, to achieve the recognition of objects with a size of  $100 \times 5$  mm.



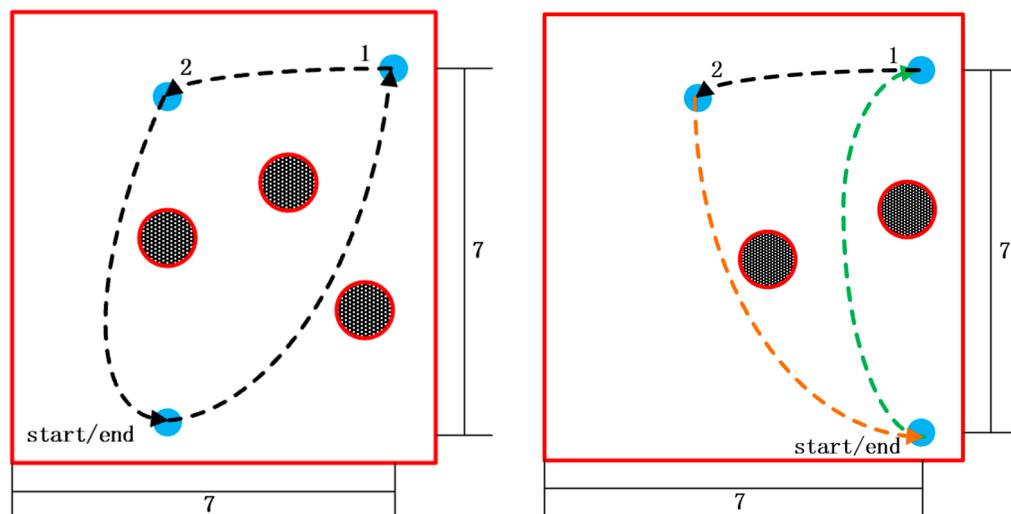
**Figure 54.** Obstacle size recognition.

(5) Obstacle detection rate

This test was placed in the flight test.

#### 5.2.2. Single-Machine Indoor Autonomous Obstacle Avoidance and Navigation

Scene tests under two conditions were performed in this paper. The maps of the two sites are shown in Figure 55:

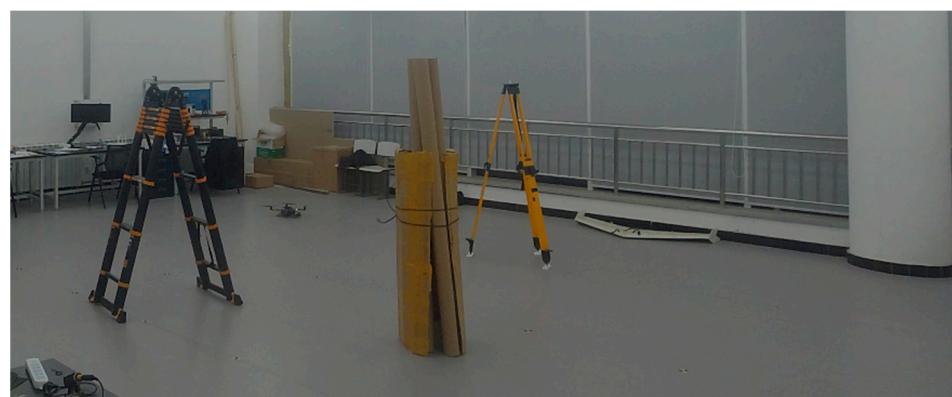


**Figure 55.** Maps of test sites.

In order to test the indoor autonomous obstacle avoidance and navigation algorithm, a  $7 \times 7$  m field was built in the room, in which two obstacles were placed. The UAV started from the start point in Figure 55, then ran to the first point, the second point, and finally returned to the end point. During this period, the system automatically recognized and avoided the two obstacles in the picture.

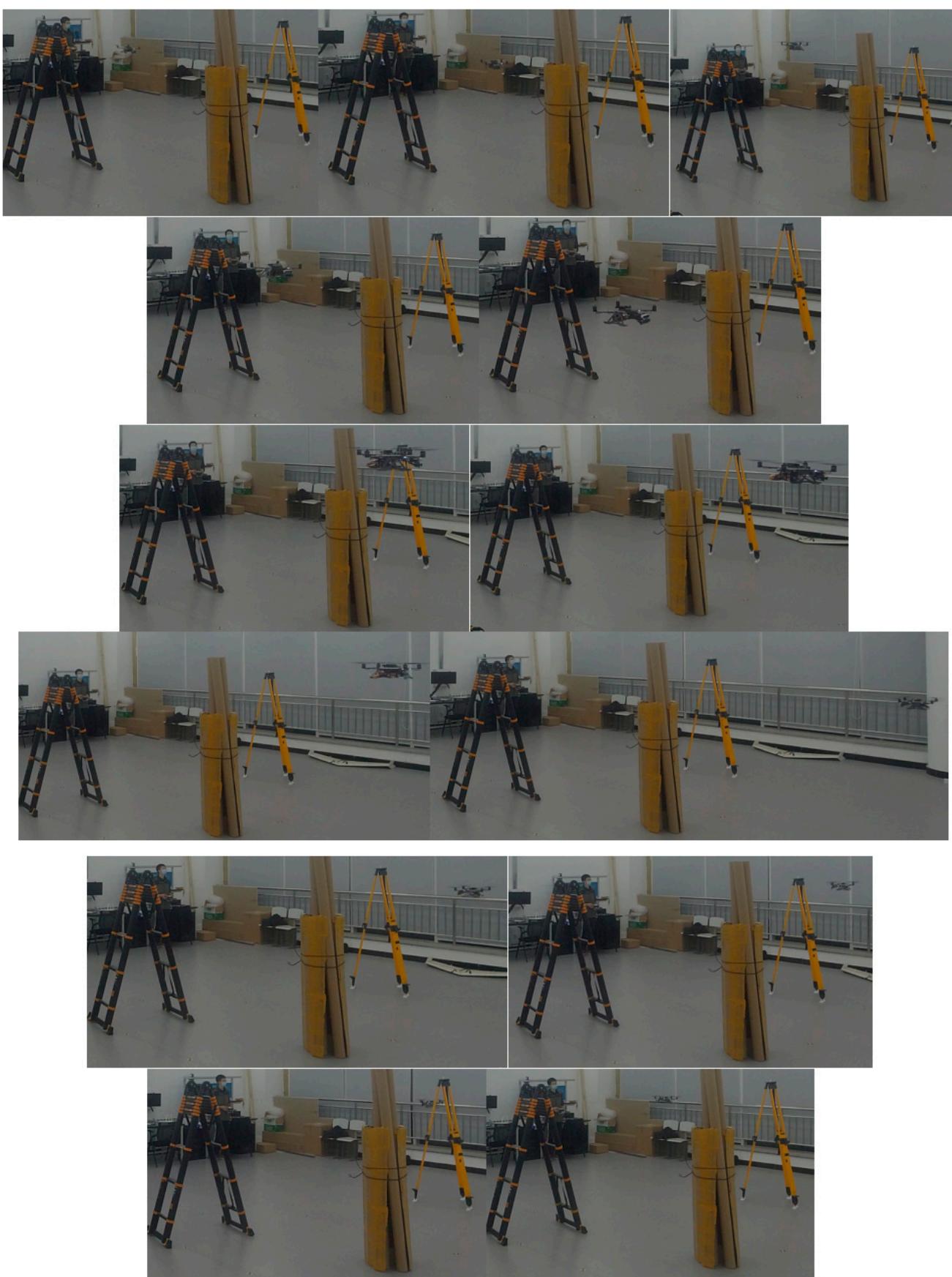
(1) Scene experiment with three obstacles

The scene diagram of the three-obstacle experiment is shown in Figure 56:



**Figure 56.** Three-obstacle scene.

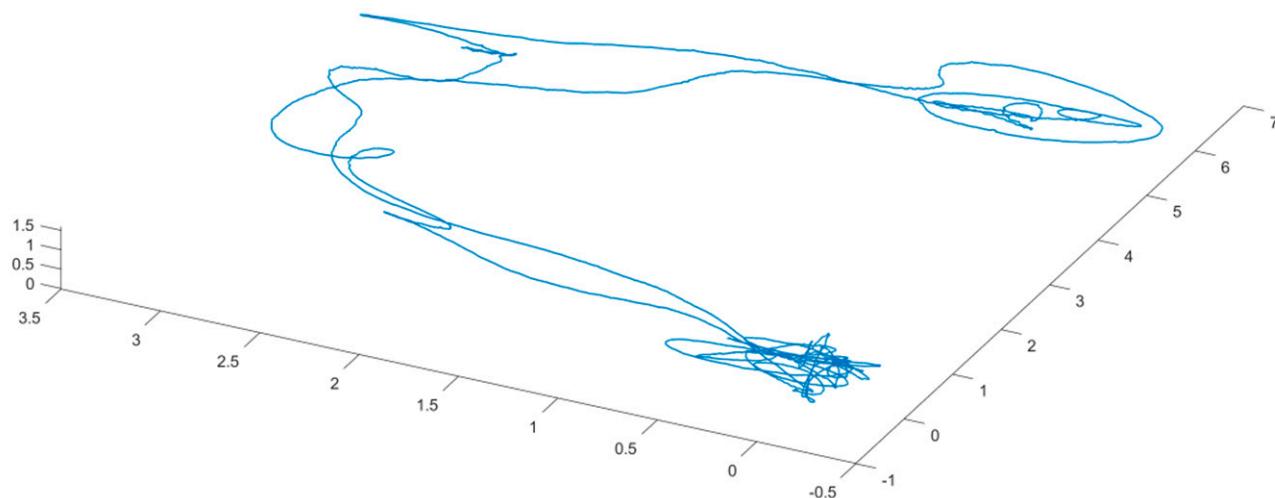
In this scenario, the UAV completed the flight test process from the starting point to the end point well, as shown in Figure 57.



**Figure 57.** Flight record of three-obstacle scene.

## (2) Scene experiment with two obstacles

In the two-obstacle scenario experiments, the flight data of the UAV were recorded as shown in Figure 58:



**Figure 58.** Flight record of two-obstacle scene.

## 6. Conclusions

This paper completed the following work: Firstly, an integrated navigation algorithm based on machine vision/close-range detection/inertial measurement unit (IMU) was designed and realized. Then, an indoor simultaneous localization and mapping (SLAM) algorithm was designed and realized. Moreover, a method for obstacle detection, obstacle avoidance motion decision and motion planning was designed and realized. At last, an autonomous navigation and obstacle avoidance simulation system was built. In the meantime, the positioning and navigation system in an unknown environment as well as the indoor obstacle avoidance flight was also demonstrated and verified. There are some advantages and weaknesses listed as follows:

- Advantages: (1) Uses distributed hardware solution to realize obstacle observation (D435i) and SLAM (T265) functions, which greatly reduces the computational power requirements of the airborne computer; (2) optimizes the YOLO network using TensorRT so it can run in real time on the onboard computer; (3) OCTomAP mapping, RRT\* and  $\beta$  spline curve fitting are finished mainly by CPU, target recognition mainly by GPU, making full use of onboard computer resources.
- Weaknesses: (1) The basic assumption of a l-SLAM system is that the environment remains static. If the environment moves in whole or part, the localization results will be disturbed; (2) OctomAP requires a cumulative period of stable observations to effectively identify obstacles, which makes the UAV unable to effectively respond to obstacles that suddenly appear; (3) the motion trajectory generated by 3-RRT\* and  $\beta$  spline curve only has position command, but no speed and acceleration command, and so cannot guide the UAV to fly at high speed.

There are also some weaknesses that should be noted and explored such as whether the detection of the obstacle and the path computation are influenced by the changing environments and so on.

**Author Contributions:** Methodology, C.C.; formal analysis, Z.W. and Z.G.; data curation, P.C.; writing—original draft preparation, Z.W. and C.Z.; writing—review and editing, Y.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

### Nomenclature

UAV	Unmanned aerial vehicle
GNSS	Global navigation satellite system
SLAM	Simultaneous location and mapping
USV	Unmanned surface vehicle
UAM	Urban air mobility
SAFDAN	Solar atomic frequency discriminator for autonomous navigation
VPC	Visual predictive control
MPC	Model predictive control
UUV	Unmanned underwater vehicle
VR	Virtual reality
GPS	Global positioning system
VI-SLAM	Visual–inertial simultaneous localization and mapping
IMU	Inertial measurement unit
VIO	Visual–inertial odometry
KLT tracking	Kanade–Lucas–Tomasi tracking
RRT	Rapidly exploring random trees
CNN	Convolutional neural network
RCNN	Region-convolutional neural network
SSD	Single shot multi-box detector
VGG	Visual geometry group
FPN	Feature pyramid network
GPU	Graphics processing unit
GIE	GPU inference engine
CUDA	Compute unified device architecture
CBR	Case-based reasoning
FP32	Full 32-bit precision
FOV	Field of view

### References

- Li, Z.; Lu, Y.; Shi, Y.; Wang, Z.; Qiao, W.; Liu, Y. A Dyna-Q-based solution for UAV networks against smart jamming attacks. *Symmetry* **2019**, *11*, 617. [[CrossRef](#)]
- Kuriki, Y.; Namerikawa, T. Consensus-based cooperative formation control with collision avoidance for a multi-UAV system. In Proceedings of the 2014 American Control Conference, Portland, OR, USA, 4–6 June 2014; IEEE: New York, NY, USA, 2014; pp. 2077–2082.
- Kwak, J.; Park, J.H.; Sung, Y. Unmanned aerial vehicle flight point classification algorithm based on symmetric big data. *Symmetry* **2016**, *9*, 1. [[CrossRef](#)]
- Turan, E.; Speretta, S.; Gill, E. Autonomous navigation for deep space small satellites: Scientific and technological advances. *Acta Astronaut.* **2022**, *193*, 56–74. [[CrossRef](#)]
- Kayhani, N.; Zhao, W.; McCabe, B.; Schoellig, A.P. Tag-based visual–inertial localization of unmanned aerial vehicles in indoor construction environments using an on-manifold extended Kalman filter. *Autom. Constr.* **2022**, *135*, 104112. [[CrossRef](#)]
- Li, Z.; Zhang, Y. Constrained ESKF for UAV Positioning in Indoor Corridor Environment Based on IMU and WiFi. *Sensors* **2022**, *22*, 391. [[CrossRef](#)]
- Aldao, E.; González-de Santos, L.M.; González-Jorge, H. LiDAR Based Detect and Avoid System for UAV Navigation in UAM Corridors. *Drones* **2022**, *6*, 185. [[CrossRef](#)]
- Zhang, W.; Yang, Y.; You, W.; Zheng, J.; Ye, H.; Ji, K.; Chen, X.; Lin, X.; Huang, Q.; Cheng, X.; et al. Autonomous navigation method and technology implementation of high-precision solar spectral velocity measurement. *Sci. China Phys. Mech. Astron.* **2022**, *65*, 289606. [[CrossRef](#)]
- Ramezani Dooraki, A.; Lee, D.J. A Multi-Objective Reinforcement Learning Based Controller for Autonomous Navigation in Challenging Environments. *Machines* **2022**, *10*, 500. [[CrossRef](#)]
- Durand Petiteville, A.; Cadenat, V. Advanced Visual Predictive Control Scheme for the Navigation Problem. *J. Intell. Robot. Syst.* **2022**, *105*, 35. [[CrossRef](#)]
- Specht, C.; Świdłaski, E.; Specht, M. Application of an autonomous/unmanned survey vessel (ASV/USV) in bathymetric measurements. *Pol. Marit. Res.* **2017**, *nr 3*, 36–44. [[CrossRef](#)]

12. Xie, S.; Wu, P.; Peng, Y.; Luo, J.; Qu, D.; Li, Q.; Gu, J. The obstacle avoidance planning of USV based on improved artificial potential field. In Proceedings of the 2014 IEEE International Conference on Information and Automation (ICIA), Hailar, China, 28–30 July 2014; IEEE: New York, NY, USA, 2014; pp. 746–751.
13. Wang, X.; Yadav, V.; Balakrishnan, S.N. Cooperative UAV formation flying with obstacle/collision avoidance. *IEEE Trans. Control Syst. Technol.* **2007**, *15*, 672–679. [[CrossRef](#)]
14. Manhães, M.M.M.; Scherer, S.A.; Voss, M.; Douat, L.R.; Rauschenbach, T. UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation. In Proceedings of the OCEANS 2016 MTS/IEEE Monterey, Monterey, CA, USA, 19–23 September 2016; pp. 1–8.
15. Liu, L.; Wu, Y.; Fu, G.; Zhou, C. An Improved Four-Rotor UAV Autonomous Navigation Multisensor Fusion Depth Learning. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 2701359. [[CrossRef](#)]
16. Sajjadi, S.; Mehrandezh, M.; Janabi-Sharifi, F. A Cascaded and Adaptive Visual Predictive Control Approach for Real-Time Dynamic Visual Servoing. *Drones* **2022**, *6*, 127. [[CrossRef](#)]
17. Nascimento, T.; Saska, M. Embedded Fast Nonlinear Model Predictive Control for Micro Aerial Vehicles. *J. Intell. Robot. Syst.* **2021**, *103*, 74. [[CrossRef](#)]
18. Ambroziak, L.; Ciężkowski, M.; Wolniakowski, A.; Romaniuk, S.; Bożko, A.; Ołdziej, D.; Kownacki, C. Experimental tests of hybrid VTOL unmanned aerial vehicle designed for surveillance missions and operations in maritime conditions from ship-based helipads. *J. Field Robot.* **2021**, *39*, 203–217. [[CrossRef](#)]
19. Hassan, S.A.; Rahim, T.; Shin, S.Y. An Improved Deep Convolutional Neural Network-Based Autonomous Road Inspection Scheme Using Unmanned Aerial Vehicles. *Electronics* **2021**, *10*, 2764. [[CrossRef](#)]
20. Zhu, H.; Liu, C.; Li, M.; Shang, B.; Liu, M. Unmanned aerial vehicle passive detection for Internet of Space Things. *Phys. Commun.* **2021**, *49*, 101474. [[CrossRef](#)]
21. He, L.; Aouf, N.; Song, B. Explainable Deep Reinforcement Learning for UAV autonomous path planning. *Aerospace Sci. Technol.* **2021**, *118*, 107052. [[CrossRef](#)]
22. Miranda, V.R.; Rezende, A.; Rocha, T.L.; Azpírua, H.; Pimenta, L.C.; Freitas, G.M. Autonomous Navigation System for a Delivery Drone. *J. Control. Autom. Electr. Syst.* **2022**, *33*, 141–155. [[CrossRef](#)]
23. Zhao, X.; Chong, J.; Qi, X.; Yang, Z. Vision Object-Oriented Augmented Sampling-Based Autonomous Navigation for Micro Aerial Vehicles. *Drones* **2021**, *5*, 107. [[CrossRef](#)]
24. Ren, Y.; Liu, X.; Liu, W. DBCAMM: A novel density based clustering algorithm via using the Mahalanobis metric. *Appl. Soft Comput.* **2012**, *12*, 1542–1554. [[CrossRef](#)]
25. Guitton, A.; Symes, W.W. Robust inversion of seismic data using the Huber norm. *Geophysics* **2003**, *68*, 1310–1319. [[CrossRef](#)]
26. Elmokadem, T.; Savkin, A.V. Towards Fully Autonomous UAVs: A Survey. *Sensors* **2021**, *21*, 6223. [[CrossRef](#)] [[PubMed](#)]
27. Zammit, C.; Van Kampen, E.J. Comparison between A\* and RRT algorithms for UAV path planning. In Proceedings of the 2018 AIAA Guidance Navigation, and Control Conference, Kissimmee, FL, USA, 8–12 January 2018; p. 1846.
28. Aguilar, W.G.; Morales, S.; Ruiz, H.; Abad, V. RRT\* GL based optimal path planning for real-time navigation of UAVs. In Proceedings of the International Work-Conference on Artificial Neural Networks, Cádiz, Spain, 14–16 June 2017; Springer: Cham, Switzerland, 2017; pp. 585–595.
29. Wang, C.; Meng, M.Q.H. Variant step size RRT: An efficient path planner for UAV in complex environments. In Proceedings of the 2016 IEEE International Conference on Real-Time Computing and Robotics (RCAR), Angkor Wat, Cambodia, 6–10 June 2016; IEEE: New York, NY, USA, 2016; pp. 555–560.
30. Kang, Z.; Zou, W. Improving accuracy of VI-SLAM with fish-eye camera based on biases of map points. *Adv. Robot.* **2020**, *34*, 1272–1278. [[CrossRef](#)]