IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 20, 2019, accepted November 2, 2019, date of publication November 7, 2019, date of current version November 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2952173

The Obstacle Detection Method of UAV Based on 2D Lidar

LANXIANG ZHENG, PING ZHANG, JIA TAN, AND FANG LI
School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China
Corresponding author: Ping Zhang (pzhang@scut.edu.cn)

This work was supported by the Science and Technology Planning Project of Guangdong Province, China, under Grant 2017B010116001.

ABSTRACT With the widespread use of UAVs in daily life, there are many sensors and algorithms used to ensure flight safety. Among these sensors, lidar has been gradually applied to UAVs due to its stability and portability. However, in the actual application, lidar changes its position with the movement of the UAV, resulting in an offset in the detected point cloud. What's more, when the lidar works, it scatters laser light from the center to the surroundings, which causes the detected point cloud to be externally sparse and dense inside. This point cloud with uneven density is difficult to cluster using common clustering algorithms. In this paper, a velocity estimation method based on the polynomial fit is used to estimate the position of the lidar as it scans each point and then corrects the twisted point cloud. Besides, the clustering algorithm based on relative distance and density (CBRDD) is used to cluster the point cloud with uneven density. To prove the effectiveness of the obstacle detection method, the simulation experiment and actual experiment were carried out. The results show that the method has a good effect on obstacle detection.

INDEX TERMS UAV, obstacle, lidar, point cloud, correction, clustering, CBRDD.

I. INTRODUCTION

In the last decade, Unmanned Aerial Vehicles (UAVs) received increasing attention from academia. UAV is an autonomous, semi-autonomous or remote-controlled unmanned aerial vehicle with lightweight, small size, high maneuverability, good concealment and adaptability. UAVs can perform dangerous, difficult-to-operate air applications and reduce costs [1]. It has been widely used in search and rescue [2], aerial mapping [3], target tracking [4], autonomous formation flight [5], collaborative search [6]. Among them, obstacle detection is one of the most important guarantees for the success of applications.

Obstacle detection is essential for autonomous safe flight of UAVs. Having obstacle information along with their location allows the generation of accurate path for the application. Obstacle information can be reflected in two ways: global obstacle information and local obstacle information. Global information is typically performed using grids, such as occupancy maps [7]. In each grid cell, the probability of the presence of an obstacle is calculated based on the range sensor and the position of the sensor itself. Especially, This method is used in conjunction with uncertainty probability sensor models to handle uncertainty in sensor measurements [8]. This method has been widely used in UAVs but comes with a computational cost and memory usage. The local obstacle information is different from the global obstacle information, and it is more concerned with the relationship between the obstacle and the current position of the UAV. This method does not take into account the current position of the UAV, using sensors to detect obstacle information [9]. Local obstacle information is more like local optimization while global obstacle information is more inclined to global optimization, and both methods have better effects. The two can be converted to each other through the position of the UAV (the position of the sensor).

The obstacle detection system can detect obstacles in the environment in real-time during the flight to ensure that the UAV has a clear understanding of its surroundings, and it is also a prerequisite for UAVs to automatically avoid obstacles [10]. In the course of flight, UAV relies on the onboard environment detection sensors to perceive the environment. The sensors should be real-time and effective, and mainly have the following types [11]: vision sensors [12], [13], lidar sensors [14], [15], and ultrasound sensors [16], [17]. Comparing with other kinds of sensors, lidar sensors bears several advantages such as being less impacted by the environment (weather, cloud cover) and flying conditions, more precision

The associate editor coordinating the review of this manuscript and approving it for publication was Noor Zaman.

and density data, and high flexibility. Thus, it has been widely used in obstacle detection systems.

In the case of obstacle detection, attention was often given to the surveying accuracy of the point cloud from lidar, which is more important for improving detection accuracy. The cause of the error mainly comes from the fixed position [18], sensor measurement error [19], and motion error [20]. Motion error is mainly considered in this paper. If the only motion of the lidar is to rotate the laser beam, the acquired point cloud can accurately reflect the obstacle distribution in the environment. However, if the lidar itself is moving with the UAV, accurate mapping requires knowledge of the lidar attitude during continuous laser ranging. There are two ways to solve this problem [21], one is to speed up the scanning speed of the lidar, and the other is to correct the point cloud. The former is limited by the hardware of the lidar and is difficult to implement. The latter uses position estimation (via GPS/INS) to map the laser points into a desired coordinate system. Position estimation can be implemented by GPS/INS [22], or visual odometry systems [23]. GPS/INS is more flexible, less affected by the environment and less computation, widely used in position estimation. Specifically, comparing with GPS, INS has higher instantaneous accuracy and is enable to estimate UAV heading, so it is more suitable for the position estimation. Once the position of the lidar during continuous laser ranging is obtained, the core process of point cloud correction is to construct an observation error equation and solve this equation.

In this paper, the primary goal is to develop a simple, cheap and utilizable obstacle detection system that provides effective obstacle information for UAVs or robots. Although many studies use lidar to detect obstacles, the effects of movement on the point cloud are ignored, which affects the accuracy of detecting obstacles. To this end, a novel and effective method to correct the point cloud obtained by lidar is proposed. This method estimates the position of the UAV when the lidar scans each point and then corrects the point cloud through relationship transformation. After analyzing the point cloud characteristics obtained by laser radar, a clustering algorithm based on relative distance and density(CBRDD) is proposed to cluster the poind cloud with uneven density.

To summarize, the contributions of our work include the following.

- Proposing a cost-effective obstacle detection system. The hardware of this system is simple and easy to fix. The cost depends on the price of the lidar, therefore, it is suitable for large-scale swarm applications.
- A point cloud correction method is proposed to estimate the position of the UAV when the lidar scans each point. This method can correct the distortion caused by the UAVs movement.
- Proposing a clustering algorithm for point clouds with uneven density obtained by lidar, which can be extended to apply to a 3D point cloud if needed.
- The obstacle detection system has a simple hardware structure, has good robustness and versatility, and can be applied to various mobile robots and even 3D motion robots.

The rest of this article is organized as follows: related work is covered in Section II. In Section III, the framework of the obstacle detection system and the experimental platform of UAV are introduced. Section IV describes the point cloud correction method and related conversion relationships in detail. Section V describes CBRDD and shows the pseudo-code for the method. We present our experimental results in Section VI. Finally, Section VII concludes the paper.

II. RELATED WORK

The steps of obstacle detection generally contain three parts [24]: obtaining obstacle information, point cloud preprocessing and point cloud clustering. Among them, obstacle information is collected by lidar, and stored in point clouds. Point cloud preprocessing mainly includes correction and filtering, in which point cloud correction is to restore the distorted point cloud, and filtering is to remove unnecessary points. Point cloud clustering extracts ordered obstacle information from cluttered point clouds and forms intuitive obstacle information.

The reason of the distorted point cloud is that the lidar has extrinsic motion during continuous laser ranging. In this case, a method is proposed to correct the distorted point cloud, using the position compensation method. Such as using the ICP (Iterative Closest Point) algorithm to obtain the motion of the lidar when two adjacent point clouds are obtained, and establishes a compensation function to correct the point cloud [25]. Although this method achieves good results, it requires a large amount of computation. Another way is to use the INS to estimate the position of the lidar when scanning each point, and then correct the point cloud to the real coordinate system [26]. However, the operating frequency of the INS is difficult to match the frequency of the lidar's scanning of each point, so it is needed to estimate the position of the lidar. Linear interpolation is a commonly used method for estimating the position of lidar. This method estimated the position of the lidar when it scans each point by calculating periodic IMU/INS data, and then constructs a transformation equation between the estimated position and the position at which the sequence is returned, mapping the point to the actual position [20], [27].

Clustering obstacle points can obtain information such as the shape, size, angle, and position of the obstacles, so that the UAV can fully recognize the obstacles and adopt effective strategies to avoid obstacles. Several algorithms have good results, for example, the algorithm for feature extraction and matching on the point cloud obtained by multi-point clouds fusion is used to identify obstacles [28], [29], which can extract the shape of obstacles and match the point cloud models in the library to classify types of obstacles. Convolutional neural networks [30], [31] are also used for obstacle recognition. However, these methods usually take more time consuming and are not conducive to UAV with high-speed

and weak computing power. For most autonomous robots, the category of obstacles is not important, and knowing the size and position of obstacles is enough to avoidance. In this case, K-MEANS [32], DBSCAN [34], Euclidean clustering method [33] are widely used. The iterative DBSCAN algorithm divides the points in the sequence into different clusters to identify obstacles in the environment [34]. DBSCAN algorithm can't accurately segment obstacles with similar density, so the DBSCAN algorithm can be combined with K-MEANS algorithm [35] to improve the obstacle clustering effect. A simple range difference of neighboring pixels (in scan angle) would perform the same effects as well [36]. This method is efficient and has a small amount of calculation, but the threshold in this method needs to change as the distance from the obstacle to the lidar changes. Besides, obstacles can be divided into circular, linear and rectangular clusters according to the number of points in the cluster and the distribution of points [37].

In conclusion, the clustering algorithm can effectively divide the disordered point cloud into classes, which makes it easier for the UAV to perceive the environment. The clustering algorithm should have a better clustering effect, and also requires a lower amount of computation so that it can be operated on an onboard microprocessor with weak computing power.

In this paper, a low-cost obstacle detection system based on a 2D lidar is proposed, which can detect obstacles in the environment during the movement of UAV. To improve the accuracy of obstacle clustering, motion state estimation is used to estimate the position of lidar, and then to solve the point cloud distortion caused by motion. Besides, the traditional clustering algorithm is less robust when clustering on point clouds with uneven density. In this case, the clustering algorithm based on relative distance and density is used to solve the problem which has a better effect than the traditional clustering algorithm.

III. SYSTEM FRAMEWORK

The UAV system framework used in this paper is shown in Fig. 1. It is mainly divided into three parts: obstacle detection sensor, microprocessor, and flight platform. In this system, the DJI Matrice 100 (M100) is used as a flight platform, a fixed 2D lidar is used as an obstacle detection sensor, and Raspberry Pi 3B (RPi) as a microprocessor.

M100 is one of the most commonly used flight platforms, equipped with an autopilot called the N1 flight controller that controls its stability while feeding back the flight data (velocity, heading, acceleration, position, etc.) of the UAV to the user through the UART. The obstacle detection system uses a low-cost 360° 2D lidar which center is fixed as close as possible to the center of the UAV, and the lidar initializes its direction towards the heading of the UAV.

As shown in Fig. 1, the flight data acquired from the M100 and the point cloud returned by the lidar are sent to the RPi via the UART. After that, a series of processing will

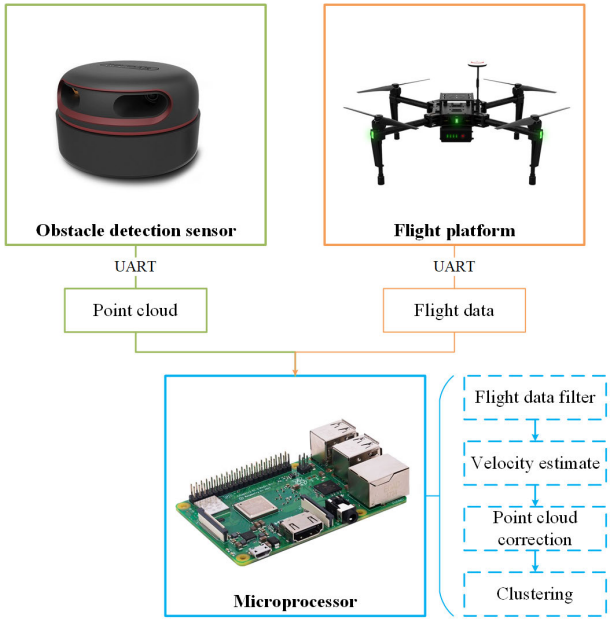


FIGURE 1. Lightweight obstacle detection system of UAVs: obstacle detection sensor (green boxes), flight platform (orange boxes), microprocessor (blue boxes).

be performed on the RPi, displayed in blue color dotted boxes in Fig. 1, so that obstacle information can be obtained.

IV. POINT CLOUD CORRECTION

In this paper, lidar consists of a fixed part and a rotational part: the former is for fixing on the UAV, while the latter realizes 360° environment scanning with the utilization of rotated measuring units, to obtain environmental point cloud of the whole plane. The lidar packs and transmits obstacle information after a work cycle, and angle and distance of each point in the returned sequence is referenced to the position at which the lidar starts this duty cycle. However, in practical applications, the lidar moves as the UAV moves, which causes the position of the lidar to change all the time. Therefore, the influence of the movement on the point cloud obtained by lidar cannot be ignored. Fig. 2 shows a simple linear

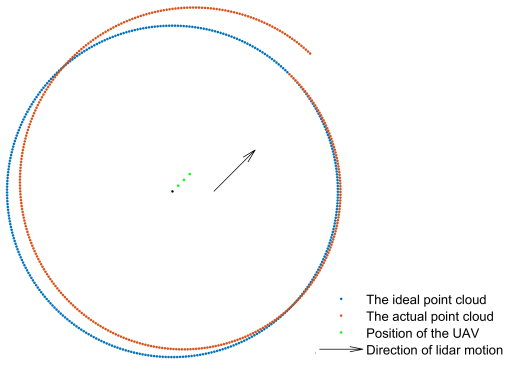


FIGURE 2. Illustration of the difference between the ideal point cloud and the actual point cloud obtained by a motion lidar.

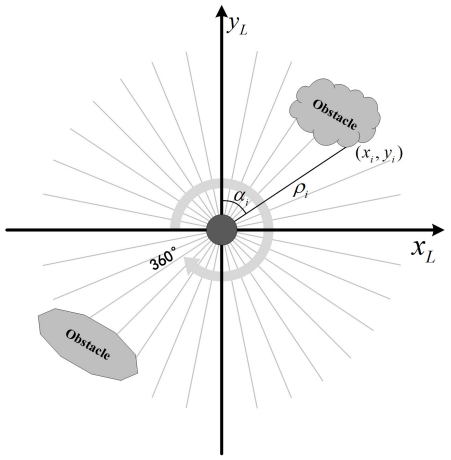


FIGURE 3. Illustration of lidar detection obstacles.

relationship between the ideal point cloud and the actual point cloud for lidar scanning. Where the direction of UAV is indicated by the arrow, and lidar scans clockwise. UAV starts at the black point and moves along green points. The blue point cloud is based on the coordinate system where the lidar starts working, and the red point cloud is actually scanned points by lidar. The offset error between two point clouds can not be ignored, and the faster the UAV moves, the larger the offset error is. Therefore, it is necessary to correct the point cloud before it sets on practice.

A. COORDINATE SYSTEM CONVERSION

The point clouds obtained by lidar is in the polar coordinates, as shown in Fig. 3, where α_i represents the angle of the i-th point relative to the initial direction of the lidar, and ρ_i represents the distance between the lidar and this point. To facilitate the calculation, it is necessary to convert sequence from the polar coordinate system to the Cartesian coordinate system. Assume the set S represents the data in the Cartesian coordinate system, $S = \{s_i | i = 1, 2, \ldots, N\}$, $s_i = (x_i, y_i)$. The coordinate conversion formula is as follows.

$$\begin{cases} x_i = \rho_i \cos\theta_i \\ y_i = \rho_i \sin\theta_i \end{cases} \tag{1}$$

B. VELOCITY ESTIMATION MODEL

Correcting the point cloud requires knowing the position of the lidar (the UAV) when the lidar scans each point. In general, the position of the UAV can be obtained by its GPS or IMU, but the frequency at which the position is obtained cannot correspond to the frequency at which each point scanned. Therefore, it is necessary to estimate the position of the UAV when the lidar scans each point. Comparing with GPS, IMU has higher instantaneous accuracy and is enable to estimate UAV heading, so it is used for the position estimation. The velocity we used comes from the feedback data from the M100, which is accurate compared to the velocity we calculated from the IMU's original data.

In this paper, the polynomial fitting algorithm is used to fit the velocity and estimate the velocity curve, and then calculate the position of lidar at each scanned point. It is well known that using the speed of the UAV closer to the target time for fitting, the estimated speed is more accurate. Therefore, it is necessary to know that the correspondence between the time of lidar scans the i-th point and the time of the latest UAV speed is acquired. Assuming the time that the lidar start working before the UAV is t_{offset}. The lidar takes time T_L to complete a work cycle, and it will package the scanned N points of obstacles to the user after at the end of the work cycle. The velocity information is obtained from IMU with a period T_U. j indicates j-th speed obtained by the UAV before the i-th point, then the following relationship exists between i and j.

$$j = \left\lfloor \frac{(cN + i)T_L - Nt_{offset}}{NT_U} \right\rfloor \tag{2}$$

In the above formula, c is the number of cycles that the lidar has scanned, and N is the number of points in the point cloud. $\lfloor x \rfloor$ means rounding down x.

Then, the polynomial fitting algorithm is used to estimate the velocity v_{Li} of UAV corresponding to each point i in the sequence. Assuming that $V_j = \{v_{j-k}, v_{j-k+1}, \cdots, v_{j-1}, v_j\}$ represents speed acquired before t_{Uj}, and k represents speed corresponding to the previous k times. Then v_{Li} can be expressed as

$$v_{Li} = p_{i,0} + p_{i,1}t + \ldots + p_{i,n}t^n \tag{3}$$

where $P_i = [p_{i,0}, p_{i,1}t, \ldots, p_{i,n}]^T$ is a vector consisting of coefficients of a polynomial fitting algorithm.

After obtaining the velocity v_{Li}, the distance of the UAV moves at the i-th point can be expressed as.

$$\begin{aligned} \Delta x &= \sum_{i=1}^{n} v_{Li}^x \Delta t_{Li} \\ \Delta y &= \sum_{i=1}^{n} v_{Li}^y \Delta t_{Li} \end{aligned} \tag{4}$$

where Δt_{Li} represents the time interval between the i-th point and the previous point. v_{Li}^x and v_{Li}^y are the velocity components along the heading and perpendicular to the heading, respectively.

C. POINT CLOUD CORRECTION

Define the world coordinate system $\{W\}$, the UAV coordinate system $\{U\}$, and the lidar coordinate system $\{L\}$. U' and L' indicate the position of UAV and lidar at the next moment. The movement of UAV when the lidar works is shown in Fig. 4. The deviation between the center of lidar and the center of UAV and the deviation between the positive direction of lidar and the positive direction of UAV are indicated by $(x_{offset}, y_{offset}, \theta_{offset})$ respectively. Therefore, the conversion between the lidar coordinate system and the

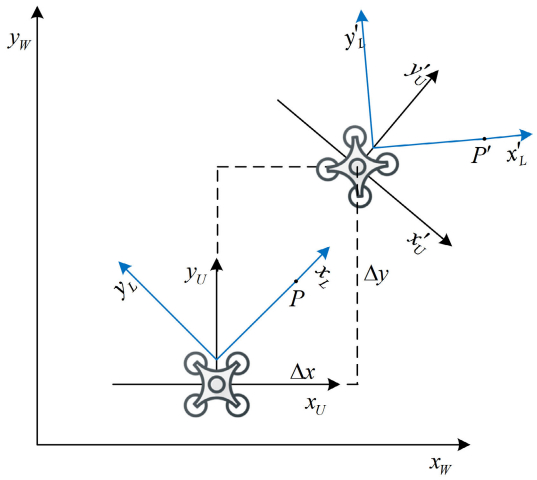


FIGURE 4. UAV's motion model during one week of lidar scanning.

UAV coordinate system is as follows.

$$T_U^L = (T_L^U)^{-1} = \begin{pmatrix} \cos\theta_{offset} & -\sin\theta_{offset} & x_{offset} \\ \sin\theta_{offset} & \cos\theta_{offset} & y_{offset} \\ 0 & 0 & 1 \end{pmatrix}^{-1} \quad (5)$$

where T_B^A represents the homogeneous transformation matrix from the coordinate system $\{A\}$ to the coordinate system $\{B\}$, and $T_B^A = (T_A^B)^{-1}$. In addition, Δx, Δy, are calculated by equation 4, representing the displacement of the UAV on the x-axis and y-axis during the Δt time period. $\Delta\theta$ represent the deflection angle of the UAV. Then the position of UAV is changed from U to U' after time Δt, expressed as $U \rightarrow U' = (\Delta x, \Delta y, \Delta\theta)$, and the transformation matrix can be defined as.

$$T_{U'}^U = (T_U^{U'})^{-1} = \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta & \Delta x \\ \sin\Delta\theta & \cos\Delta\theta & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \quad (6)$$

As shown in Fig. 4, P is a point obtain by lidar and the measured correction of the point is P', and the homogeneous conversion between them is as follows.

$$\tilde{P}_L = T_U^L \cdot T_{U'}^U \cdot T_{L'}^{U'} \cdot P_{L'} \quad (7)$$

V. CBRDD

When the lidar is working, it emits a laser beam to the surroundings. The laser beam will reflect if it encounters an obstacle, which in turn can assess the distance from the lidar to the obstacle. Due to the special working mode of the lidar, the point cloud of the obstacle returned is unevenly distributed. For example, Fig. 5(a) shows the point cloud of lidar scanning the ground and lidar is at the origin of the coordinates, and points detected near the y-axis are denser and points far from the y-axis are sparse. To visualize the detection characteristics of lidar more intuitively, the processed scanning results are shown in Fig. 5(b) and Fig. 5(c), where Fig. 5(b) shows the distance between two adjacent points and Fig. 5(c) shows the density distribution of points in units of $0.2 \times 0.2\ m^2$. Through Fig. 5, the following features of the lidar can be obtained.

Feature 1: The distance between two adjacent points is proportional to the distance from the lidar, and it is small in which points near the lidar and is big where points far away from the lidar.

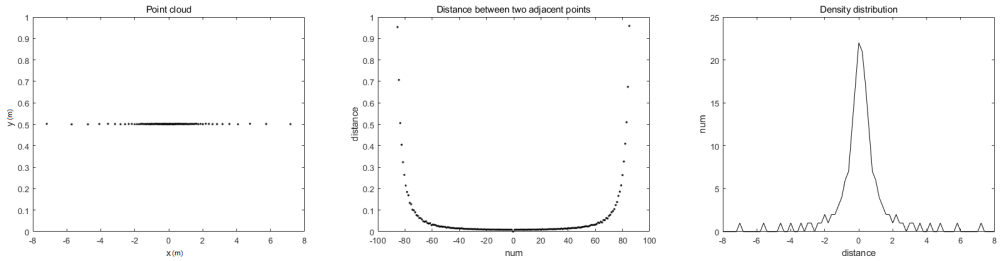
Feature 2: Detection points close to lidar are dense and the peripheral points are sparse, as shown in Fig. 5(c).

Due to the above features, the general clustering algorithm does not apply to this kind of point clouds. Therefore, clustering based on relative distance and density (CBRDD) algorithm is used here, which has strong versatility to handle both dense and sparse cloud points.

A. RELATIVE DISTANCE

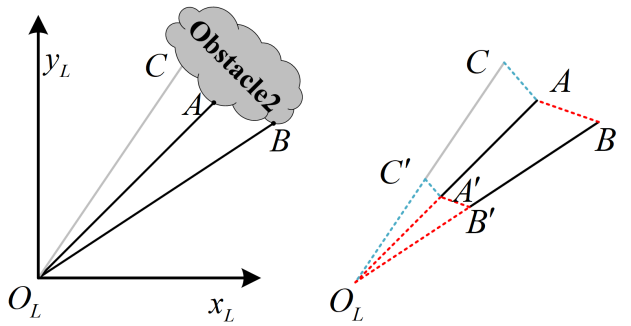
Generally, judging whether two points belong to the same cluster needs to compare the distance between the two points and the threshold. Such as, in Fig. 6(a), the lidar detects an obstacle, where A, B, and C are the corresponding points of the laser beam on the obstacle, respectively. A and B belong to the same cluster, if and only if the distance between these two points is less than the distance threshold. And the distance thresholds need real-time changes so that clustering algorithms can be applied to a wider range of detections [36], which is a trouble. In this case, we convert the Euclidean distance into the relative distance so that a distance threshold can be applied to each point.

This transformation is similar to the scaling transformation of a triangle, as shown in Fig. 6(b). First, connect the point AB



(a) Ground point cloud (b) Distance between adjacent points (c) Density distribution

FIGURE 5. Lidar detection characteristics.



(a) Euclidean distance between two points (b) Illustration of conversion between relative distance and Euclidean distance

FIGURE 6. The shortest distance between two adjacent points in the point cloud.

to construct a triangle ABO_L (ΔABO_L), then shorten the edge OA to the unit length, while shortening the sides AB and BO_L at the same ratio. After that, a similar $\Delta A'B'O_L$ of ΔABO_L is constructed, as shown as the red triangle in Fig. 6(b). $\Delta C'A'O_L$ can be acquired after similar treatments, shown in the blue triangle in Fig. 6(b). In this way, we can map each point in the point cloud to the relative coordinate system of i-th point, and then we can select the appropriate threshold for clustering.

When calculating the relative distance between points and the i-th point, the reference distance bd_i can be defined as

$$bd_i = \sqrt{x_i^2 + y_i^2} \tag{8}$$

where (x_i, y_i) is the position of i-th point. Then the relative distance between i-th point and j-th point can be defined as

$$rd_{i,j} = \frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{bd_i} \tag{9}$$

Referring to the definition of the distance matrix, the relative distance matrix can be defined as follows, using the reference data as a row and other data as a column.

$$RD = \begin{bmatrix} rd_{11} & \cdots & rd_{1N} \\ \vdots & \ddots & \vdots \\ rd_{N1} & \cdots & rd_{NN} \end{bmatrix} \tag{10}$$

Although the relative distance method is effective, there is also a disadvantage that when the relative distance of i to j satisfies the threshold, the relative distance of j to i may not be satisfied. As shown in Fig. 7, A and B are two points detected by lidar, and $\Delta A'B'O_L$ is a triangle based on the reference distance O_LA while $\Delta A'B'O_L$ is based on reference distance O_LB. When rd_{ba} just meets the threshold, it is obvious that rd_{ab} is not satisfied. But this does not affect our clustering algorithm, because in this method, as long as any relative distance is less than the threshold, the two points will be attributed to the same cluster.

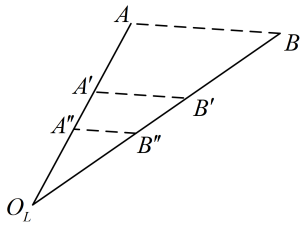


FIGURE 7. Relative distances between A and B based on different reference distances.

B. POINT CLOUD CLUSTERING ALGORITHM

The clustering algorithm used has some simplifications and improvements on the DBSCAN algorithm [39], mainly based on core points and edge points. All points can be divided into core points and edge points according to the relative distance density. The core point is the point that is closer to the cluster center, and the edge point is the point that is far from the center and close to the cluster boundary. The definition of these two types of points is as follows.

Refer to the definition in CFDP [38], using σ_i for local relative density, σ_c for relative density threshold, rd_c for relative distance threshold. σ_i represents the number of points in the i-th row of the relative distance matrix whose relative distance is less than the relative distance threshold, that is, the number of points in the point cloud whose relative distance to the i-th point is less than the threshold, defining a binary function as follows.

$$\chi(x) = \begin{cases} 1 & x \le 0 \\ 0 & x > 0 \end{cases} \tag{11}$$

Then the definition of σ_i is as follows.

$$\sigma_i = \sum_{i,j \in S, i \neq j} \chi(rd_{ij} - rd_c) \tag{12}$$

The local relative density of all points in the point cloud constitutes a set σ, $\sigma = (\sigma_1, \sigma_2, \cdots, \sigma_N)$.

Algorithm 1 Calculate the Relative Distance Density of Point Clouds

Input: The corrected point cloud $S = \{s_1, s_2, \ldots, s_n\}$; relative distance threshold rd_c
Output: Relative distance density den

```
for i=1 to n do
    selfDist[i] = distance_between(s[i], 0)
    for j!=i and j<n do
        RD[i,j] = distance_between(s[i], s[j]) / selfDist[i]
    end for
end for
for i=1 to n do
    DEN[i] = the num of RD[i]<rd_c
end for
return DEN
```

A point that satisfies the core point decision condition $\chi(\sigma_i - \sigma_c) > 0$, is a core point, otherwise, it is an edge point.

Algorithm 2 CBRDD Algorithm Clustering

Input: All points in set S; relative density threshold den_c and relative distance threshold rd_c; the relative density matrix DEN.

Output: The cluster result.

```
Define CLUSTERS to store points, core points set CORESET and edge points set EDGESET;.
for i=1 to sizeof(DEN) do
    if DEN[i] > den_c+1 then add point s[i] to CORESET
    elseadd point s[i] to EDGESET
    end if
end for
while CORESET is not Empty do
    clusterNum = clusterNum + 1
    Establish newCluster
    add CORESET[i] to newCluster
    remove CORESET[i] form CORESET
    while node[j] satisfy the condition: RD[i,j] < rd_c or RD[j,i] < rd_c do
        add node[j] to newCluster
        remove node[i] if node[i] in CORESET
        remove node[j] if node[j] in CORESET
        remove node[j] if node[j] in EDGESET
    end while
    for node[i] in newCluster do
        while RD[i,j]<den_c do
            add j to newCluster
            RD[i,j] = Inf
            RD[j,i] = Inf
        end while
    end for
    CLUSTERS[clusterNum] = clusterNew
end while
for node[i] in EDGESET do
    clusterNum = clusterNum + 1
    Establish newCluster
    add point to newCluster
    CLUSTERS[clusterNum] = clusterNew
end for
return CLUSTERS
```

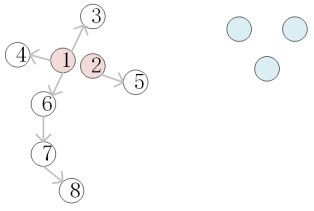


FIGURE 8. Illustration of point cloud clustering.

Then the connectivity of the point cloud is used to classify the point cloud. As shown in Fig. 8, red dots and blue dots are core points, and the other points are the edge points. Starting from the red points 1 and 2, the clustering algorithm sequentially traverses all the points that satisfy the relative distance threshold to generate a set. The blue points in Fig. 8 are in a special cluster with only core points and no edge points. The detailed clustering algorithm is as follows.

Assume U and E are core point set and edge point set respectively, and sets $C_1, C_2, \cdots, C_k$ store clustered points. The clustering criteria for the points are as follows.

Step 1. Start with the first point u_1 in the core point set and put the point into the new set C_k. Move all points in the relative distance matrix whose relative distance is less than the relative distance threshold rd_c into the set C_k. Then remove the elements that exist both in the set U and set C_k from the core point set U, and set the corresponding rd_{ij} and rd_{ji} in the relative distance matrix to infinity. Remove the elements that exist both in the set E and set C_k from the edge point set E.

Step 2. Traversing set C_k, moving all points in the relative distance matrix whose rd_{ij} is less than the relative distance threshold into the set C_k. Then remove the elements that exist both in the set U and set C_k from the core point set U, and set the corresponding rd_{ij} and rd_{ji} in the relative distance matrix to infinity. Remove the elements that exist both in the set E and set C_k from the edge point set E.

Step 3. Repeat step 2 until there are no points that satisfy the criteria. So far, the division of points belonging to the same cluster with the core point u_1 is ended.

Step 4. The above steps are repeated for the first point in the core point set U, and ends when the core point set U is empty. Points that are not yet assigned to each cluster at the end are noise points, which can be divided into a new set or discarded according to system requirements.

VI. EXPERIMENTAL RESULT

A. EXPERIMENTAL ENVIRONMENT

The algorithm is tested on the computer and UAV system respectively. The computer is used to conduct the simulation, and the UAV system performs practical feasibility. On the computer, the simulation is constructed using Gazebo 7 and ROS Kinetic. As shown in Fig. 9, the platform of UAV system is DJI's M100 UAV, equipped with Raspberry Pi 3B and a 2D lidar. The lidar is RPLIDAR A2, which has a detection range of 8 m, a weight of 190 g, a scanning frequency of 10 Hz, a resolution angle of 1°, and returns a sequence containing 360 points after once scan. Raspberry Pi 3B's hardware are 4 core Broadcom BCM2837 64-bit ARMv8 processor 1.2GHz and 1GB RAM, and its operating system is Ubuntu MATE 16.04. M100 weighs 2355g and has a maximum load of 1169g.

B. VELOCITY ESTIMATION ALGORITHM EXPERIMENT

The parameters in the polynomial fitting algorithm will affect the results. To select the appropriate parameters, we randomly select 3000 data in the speed log file of the UAV's outdoor flight, changed the parameters, and estimating the UAV speed of each point in the sequence. The above experiment was

TABLE 1. Standard deviation and time consumption of the velocity estimation algorithm.

	Standard deviation($\times 10^{-2}$)							Time consumption(ms)						
Degree	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=3	S=4	S=5	S=6	S=7	S=8	S=9
n=2	1.880	1.422	1.249	**1.212**	1.248	1.328	1.448	2.993	2.932	2.887	**2.884**	2.948	2.938	2.887
n=3	1.881	3.434	2.303	1.819	1.588	1.480	1.429	2.368	3.929	3.878	3.749	3.929	3.676	3.823
n=4	1.882	1.427	6.432	3.936	2.890	2.354	2.065	2.366	2.677	5.858	5.181	5.326	6.174	5.845
n=5	1.884	1.426	1.257	24.020	6.536	4.843	3.481	2.360	2.661	2.602	5.790	6.150	5.774	5.276
n=6	1.885	1.426	1.258	1.215	12.601	7.609	4.507	2.293	2.656	2.656	2.692	5.474	6.229	5.848

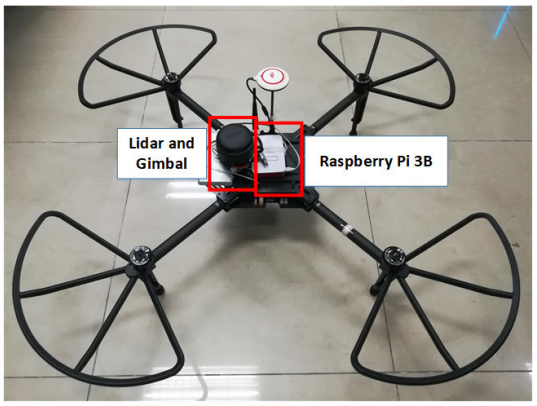
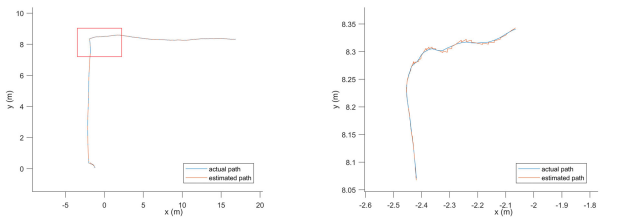


FIGURE 9. The platform of UAV system and environmental detection system used in the experiment.

carried out 10 times, and the standard deviation and the time consumption of the velocity estimation algorithm under different parameters are shown in Table 1, where the order is represented by n and the number of fitted data is represented by S.

As can be seen from Table 1, at the same order, the standard deviation does not decrease as the number of fitted data increases, but gradually increasing after decreasing. This means that there is an overfitting phenomenon, so the fitting data cannot be too much. We noticed that there is a minimum standard deviation at $S = 6, n = 2$, shown by the bold values in Table 1, and the time consumption is small at the same time. Therefore, s=6, n=2 is selected as the parameter of the polynomial fit.

Then we compare the estimated path with the original path, as shown in Fig 10. The original path is one of the flight data fed back by M100, having a high accuracy [40]. The estimated path is generated by estimating the velocity of the UAV corresponding to each point of the sequence obtained by lidar between the two adjacent feedback data, and will be calibrated when new feedback data is acquired. Fig. 10 are comparisons of estimated and actual paths, and Fig. 10(b) is a drawing of partial enlargement in the red box in Fig. 10(a). As can be seen, the estimated path can fit the actual path and can be used to correct the point cloud.



(a) Actual path and estimated path. (b) Drawing of partial enlargement

FIGURE 10. The result of path estimation algorithm.

C. POINT CLOUD CORRECTION ALGORITHM SIMULATION EXPERIMENT

To verify the feasibility and effectiveness of the point cloud correction algorithm, we designed a simulation experiment and built a 3D scene on Gazebo as shown in Fig 11. In this scene, the red rectangular blocks are impenetrable obstacles which are randomly distributed. The UAV flies at a speed of $2m/s$ with fixed altitude. The lidar scanning range is $R = 8m$, the working frequency is $10Hz$, the adjacent scanning interval is $1°$, and the number of sequence points obtained by scanning is $N = 360$.

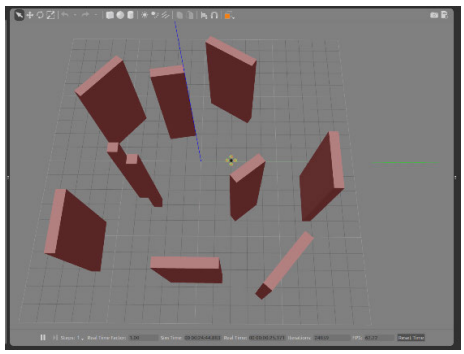
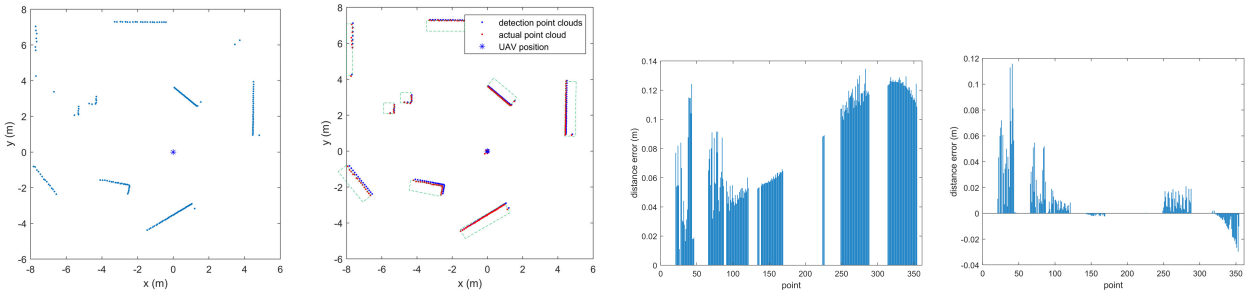


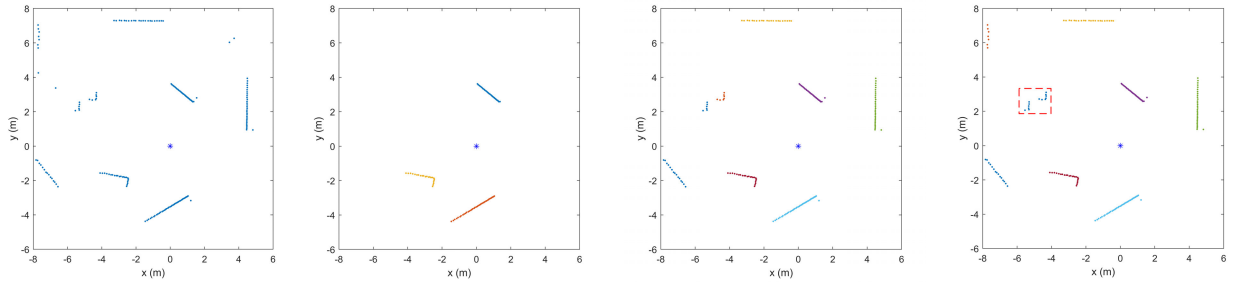
FIGURE 11. Gazebo 3D simulation scene.

The single-scan results of the lidar are shown in Fig. 12(a), and dots in the figure represent the point cloud of obstacles. The original point cloud and the corrected point cloud are shown in Fig. 12(b), and are represented by blue and red respectively. Also, the actual position of obstacles is represented by green rectangular dotted boxes in Fig 12(b). The error of each point between the original point cloud and the actual point cloud before and after point cloud correction is shown in Fig. 12(c) and Fig. 12(d), respectively. Comparing Fig. 12(c) and Fig. 12(d), it can be found that after the point cloud correction, the error is mainly the random error of the lidar detection.



(a) Original point cloud (b) Original point cloud and corrected point cloud (c) Error before correction (d) Error after correction

FIGURE 12. Point cloud correction.

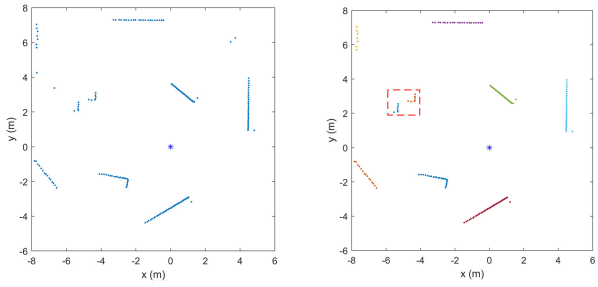


(a) Corrected point cloud (b) $Eps = 0.2, k = 5$ (c) $Eps = 0.5, k = 5$ (d) $Eps = 0.8, k = 5$

FIGURE 13. BNSCAN clustering results.

D. CLUSTERING ALGORITHM SIMULATION EXPERIMENT

After correction, the clustering algorithm is used to obtain the distribution information of obstacles. In order to verify the clustering effect of the CBRDD algorithm, we compare it with the DBSCAN algorithm, and the result is shown in Fig. 13 and Fig. 14.



(a) Corrected point cloud (b) $d_s = 0.1, k = 5$

FIGURE 14. CBRDD clustering results.

Fig. 13 are results of the DBSCAN algorithm, and different clusters are in a different color. Differences between those figures are due to the difference in the parameters *Eps* and k. *Eps* refers to the core point *Eps* domain radius, and k refers to the core point *Eps* neighborhood density threshold [39]. The corrected point cloud is shown in Fig. 13(a), and the position of the UAV is indicated by $*$. In Fig. 13(b), when $Eps = 0.2$ is small, the DBSCAN algorithm can cluster high-density point clouds near the lidar, but cannot cluster the low-density point clouds away from the lidar. In Fig. 13(c), $Eps = 0.5$, the algorithm is more effective than that in Fig. 13(b), and can effective to cluster the majority of point clouds except peripheral point clouds. In Fig. 13(d), $Eps = 0.8$, all points can be clustered, but since the *Eps* is large, it is easy to divide points closer to the lidar into the same cluster, such as clustering result inside the red dotted box shown in the Fig. 13(d).

The clustering results of the CBRDD algorithm is shown in Fig. 14(b) with the parameter $d_s = 0.1$ and $k = 5$. It can be seen from the figure that different clusters have been identified by different colors, and two clusters that are closer together within the red dotted box are also distinguished. It indicates that the CBRDD algorithm can effectively cluster point clouds with different densities.

E. POINT CLOUD CORRECTION ALGORITHM ACTUAL EXPERIMENT

To further verify the method proposed in this paper, we use the UAV system to detect obstacle information in the environment. We select a more representative scene, as shown in Fig. 15, the scene contains an arch bridge and four pillars. The UAV automatically flies under the bridge to detect obstacle information with a fixed altitude.

In this experiment, the velocity of the UAV is 2m/s, and the flight state diagram of UAV during the experiment is shown

FIGURE 15. Experimental scene.

in Fig. 16. We randomly select one of the obstacle information sequences returned by the lidar, as shown in Fig. 17, in which, the original point cloud is in blue, the corrected point cloud is in red, and green dotted boxes indicate the actual obstacles. To see more clearly, the point cloud in the red box is magnified and displayed on the right side of Fig. 17. As can be seen from Fig. 17, the corrected point cloud more accurately reflects the information of real obstacles than the original point cloud.

FIGURE 16. Several moments of flight.

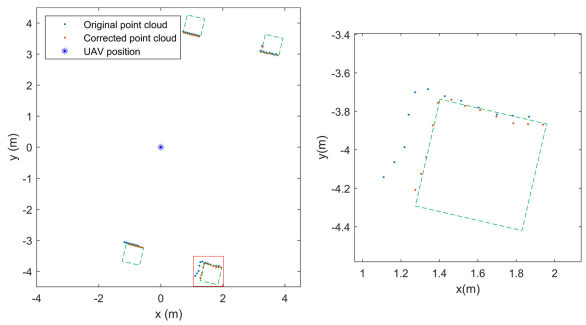
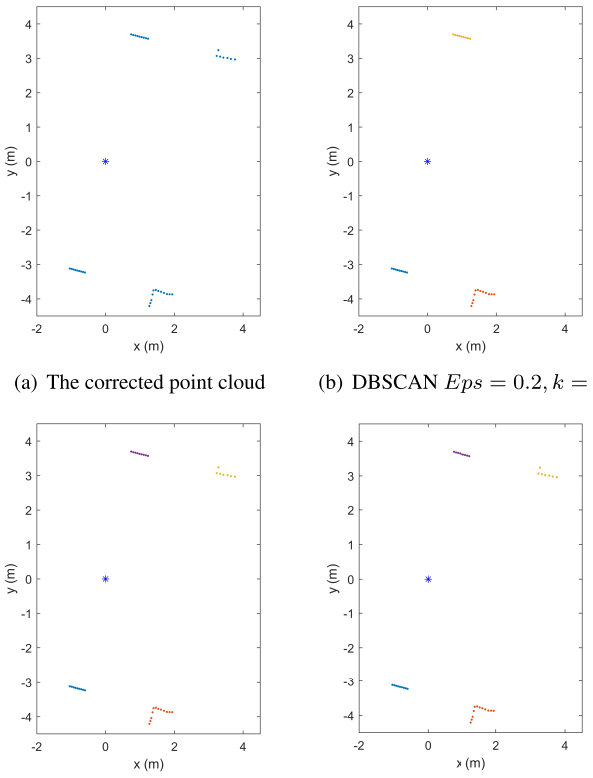


FIGURE 17. Original point cloud and corrected point cloud.

F. CLUSTERING ALGORITHM ACTUAL EXPERIMENT

After correction, the point cloud is clustered by DBSCAN and CBRDD respectively, and the clustering results are shown in Fig. 18. Fig. 18(b) and Fig. 18(c) are the clustering result of the DBSCAN algorithm with parameters $Eps = 0.2, k = 5$ and $Eps = 0.5, k = 5$. When the parameter $Eps = 0.2$, $k = 5$, the DBSCAN algorithm can cluster the three obstacles in the point cloud, ignoring the obstacles in the upper right corner. That is mainly because that this neglected obstacle point cloud is far from the lidar, causing the distance between any two adjacent points in the point cloud to exceed the parameter *Eps* of the DBCSAN. Therefore, the large parameter $Eps = 0.5$ can better cluster the point cloud, as shown in Fig. 18(c). From the result of the CBRDD algorithm, as shown in Fig. 18(d), we can see that the effect of clustering is the same as DBSCAN at parameter $Eps = 0.5$. It can be concluded from the above experiments that the CBRDD algorithm has the same effect in some cases, such as when obstacles have a similar distance to the lidar.



(a) The corrected point cloud (b) DBSCAN $Eps = 0.2, k = 5$

(c) DBSCAN $Eps = 0.5, k = 5$ (d) CBRDD $ds = 0.1, k = 5$

FIGURE 18. DBSCAN and CBRDD algorithm clustering results.

Besides, the performance of the Raspberry Pi was monitored in actual experiments, shown in Table 2. The point cloud correction algorithm requires less time than the clustering algorithm, and the time consuming of 0.5 ms while the clustering algorithm has a large time consuming. CPU occupancy is the occupied proportion of single-core (RPi 3B has 4 cores). The occupied memory (MEM) between the two

TABLE 2. Consumption of algorithms.

	Point cloud correction	CBRDD
Time consuming (ms)	0.5	14.2
CPU occupancy $(\%)$	8.3	18.4
MEM occupancy $(\%)$	1.5	1.9

methods is not much different, because both need to store a large amount of point cloud data.

In summary, the above experiments verify the feasibility of the obstacle detection algorithm proposed in this paper. The point cloud correction algorithm can effectively reduce the influence of the movement on the obstacle point cloud, and the corrected point cloud is closer to the real position of the obstacle. By comparing the CBRDD and DBSCAN algorithms, the CBRDD algorithm has the same result as the DBSCAN algorithm when the point cloud density is uniform, such as the clustering algorithm actual experiment. However, in the clustering algorithm simulation experiment, CBRDD has a better effect than DBSCAN when the point cloud density is not uniform. Therefore, the CBRDD algorithm is more suitable for the uneven density point clouds acquired by lidar.

VII. CONCLUSION

The paper proposed an obstacle detection system consists of a lidar and a raspberry pie, which is lightweight and low cost. The detection method is divided into two parts: point cloud correction and CBRDD. To correct the point cloud, the real velocity of the lidar is estimated by polynomial fitting, and then correct the distorted point cloud using position calculated according to the estimated speed. CBRDD is used to cluster the point cloud with uneven density obtained by lidar.

In the experimental section, methods are verified by simulation experiments and actual experiments respectively. The popular clustering algorithms DBSCAN for point clouds was selected as a comparison. The experimental results show that the method proposed in this paper can correct the offset of the point cloud and effectively cluster the point cloud with uneven density. Besides, the experiment also proves that the lightweight and inexpensive obstacle detection system can have a good effect on the UAV.

In future works, we will optimize the system model and algorithm, looking for a method that can replace the relative distance or address its shortcomings. At the same time, we will combine it with the gimbal or similar institutions [27] to achieve the detection of 3D obstacles.

REFERENCES

[1] B. Sinopoli, M. Micheli, G. Donato, and T.-J. Koo, "Vision based navigation for an unmanned aerial vehicle," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, vol. 2, May 2001, pp. 1757–1764.

[2] W. Zhao, Q. Meng, and P. W. H. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 902–915, Apr. 2016.

[3] W. Ni, G. Sun, Y. Pang, Z. Zhang, J. Liu, A. Yang, Y. Wang, and D. Zhang, "Mapping three-dimensional structures of forest canopy using UAV stereo imagery: Evaluating impacts of forward overlaps and image resolutions with LiDAR data as reference," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 10, pp. 3578–3589, Sep. 2018.

[4] U. Zengin and A. Dogan, "Real-time target tracking for autonomous uavs in adversarial environments: A gradient search algorithm," *IEEE Trans. Robot.*, vol. 23, no. 2, pp. 294–307, Apr. 2007.

[5] F. Liao, R. Teo, J. L. Wang, X. Dong, F. Lin, and K. Peng, "Distributed formation and reconfiguration control of VTOL UAVs," *IEEE Trans. Control Syst. Technol.*, vol. 25, no. 1, pp. 270–277, Jan. 2017.

[6] L. F. Bertuccelli and M. L. Cummings, "Operator choice modeling for collaborative UAV visual search tasks," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 5, pp. 1088–1099, Sep. 2012.

[7] A. Y. Hata, F. T. Ramos, and D. F. Wolf, "Monte Carlo localization on Gaussian process occupancy maps for urban environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 9, pp. 2893–2902, Sep. 2018.

[8] C. Yin, Z. Xiao, X. Cao, X. Xi, P. Yang, and D. Wu, "Offline and online search: UAV multiobjective path planning under dynamic urban environment," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 546–558, Apr. 2018.

[9] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a UAV with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015.

[10] H. Jing-Lin, S. Xiu-Xia, L. Ri, D. Xiong-Feng, and L. Mao-Long, "UAV real-time route planning based on multi-optimized RRT algorithm," in *Proc. 29th Chin. Control Decis. Conf. (CCDC)*, May 2017, pp. 837–842.

[11] A. Mukhtar, L. Xia, and T. B. Tang, "Vehicle detection techniques for collision avoidance systems: A review," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2318–2338, May 2015.

[12] C. Hu, F. Arvin, C. Xiong, and S. Yue, "Bio-inspired embedded vision system for autonomous micro-robots: The LGMD case," *IEEE Trans. Cogn. Develop. Syst.*, vol. 9, no. 3, pp. 241–254, Sep. 2016.

[13] K. McGuire, G. de Croon, C. D. Wagter, K. Tuyls, and H. Kappen, "Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1070–1076, Apr. 2017.

[14] T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—Concepts and a path to the future," *Proc. IEEE*, vol. 100, pp. 1831–1839, May 2012.

[15] Z. J. Chong, B. Qin, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, and D. Rus, "Mapping with synthetic 2D LIDAR in 3D urban environment," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 4715–4720.

[16] A. Lay-Ekuakille, P. Vergallo, D. Saracino, and A. Trotta, "Optimizing and post processing of a smart beamformer for obstacle retrieval," *IEEE Sensors J.*, vol. 12, no. 5, pp. 1294–1299, May 2012.

[17] M. R. Strakowski, B. B. Kosmowski, R. Kowalik, and P. Wierzba, "An ultrasonic obstacle detector based on phase beamforming principles," *IEEE Sensors J.*, vol. 6, no. 1, pp. 179–186, Feb. 2006.

[18] Z. Li, J. Tan, and H. Liu, "Rigorous boresight self-calibration of mobile and UAV LiDAR scanning systems by strip adjustment," *Remote Sens.*, vol. 11, no. 4, p. 442, 2019.

[19] M. D. Adams, "Lidar design, use, and calibration concepts for correct environmental detection," *IEEE Trans. Robot. Autom.*, vol. 16, no. 6, pp. 753–761, Dec. 2000.

[20] J. Xu, J. Lv, Z. Pan, Y. Liu, and Y. Chen, "Real-time LiDAR data assocation aided by IMU in high dynamic environment," in *Proc. IEEE Int. Conf. Real-Time Comput. Robot. (RCAR)*, Aug. 2018, pp. 202–205.

[21] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, "LiDAR point clouds correction acquired from a moving car based on CAN-bus data," 2017, *arXiv:1706.05886*. [Online]. Available: https://arxiv.org/abs/1706.05886

[22] J. N. Gross, Y. Gu, M. B. Rhudy, S. Gururajan, and M. R. Napolitano, "Flight-test evaluation of sensor fusion algorithms for attitude estimation," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 48, no. 3, pp. 2128–2139, Jul. 2012.

[23] K. Konolige, M. Agrawal, and J. Sola, "Large-scale visual odometry for rough terrain," in *Robotics Research*. Berlin, Germany: Springer, 2010, pp. 201–212.

[24] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-D lidar," in *Proc. IEEE Int. Conf. Inf. Automat.*, Aug. 2015, pp. 1648–1653.

[25] S. Hong, H. Ko, and J. Kim, "VICP: Velocity updating iterative closest point algorithm," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2010, pp. 1893–1898.

[26] S. Schneider, M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Fusing vision and lidar-synchronization, correction and occlusion reasoning," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2010, pp. 388–393.

[27] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in *Robotics: Science and Systems*, vol. 2. 2014.

[28] Z. Rozsa and T. Sziranyi, "Obstacle prediction for automated guided vehicles based on point clouds measured by a tilted LIDAR sensor," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 8, pp. 2708–2720, Aug. 2018.

[29] Y. Choe, S. Ahn, and M. J. Chung, "Online urban object recognition in point clouds using consecutive point information for urban robotic missions," *Robot. Auton. Syst.*, vol. 62, no. 8, pp. 1130–1152, Aug. 2014.

[30] B. Li, "3D fully convolutional network for vehicle detection in point cloud," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 1513–1518.
[31] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2017, pp. 1355–1361.
[32] X. Wang, C. Yang, Z. Ju, H. Ma, and M. Fu, "Robot manipulator self-identification for surrounding obstacle detection," *Multimedia Tools Appl.*, vol. 76, no. 5, pp. 6495–6520, 2017.
[33] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "SegMatch: Segment based place recognition in 3D point clouds," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2017, pp. 5266–5272.
[34] A. Asvadi, L. Garrote, C. Premebida, P. Peixoto, and U. Nunes, "DepthCN: Vehicle detection using 3D-LIDAR and ConvNet," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, 2017, pp. 1–6.
[35] J. Duan, L. Shi, J. Yao, D. Liu, and Q. Tian, "Obstacle detection research based on four-line laser radar in vehicle," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2013, pp. 2452–2457.
[36] M. Hammer, M. Hebel, M. Laurenzis, and M. Arens, "Lidar-based detection and tracking of small UAVs," *Proc. SPIE*, vol. 10799, Oct. 2018, Art. no. 107990S.
[37] P. Wu, S. Xie, H. Liu, J. Luo, and Q. Li, "A novel algorithm of autonomous obstacle-avoidance for mobile robot based on LIDAR data," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2015, pp. 2377–2382.
[38] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.
[39] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD*, vol. 96, no. 34, 1996, pp. 226–231.
[40] S. Świerczynski and A. Felski, "Determination of the position using receivers installed in UAV," in *Proc. Eur. Navigat. Conf. (ENC)*, Apr. 2019, pp. 1–4.

LANXIANG ZHENG received the B.S. degree in electronic information engineering from Shenzhen University, Shenzhen, China, in 2017. He is currently pursuing the M.S. degree in computer science with the South China University of Technology.

His current research interests include distributed control, obstacle detection, and collaborative obstacle avoidance.

PING ZHANG received the B.S. degree in mechanical engineering and the M.S. and Ph.D. degrees in robotics from Tianjin University, Tianjin, China, in 1985, 1988, and 1994, respectively.

He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His research interests include intelligent networked robotics, intelligent networked manufacturing, human–computer interaction, and real-time embedded systems.

JIA TAN received the B.S. degree in electronic information science and technology from the Wuhan University of Technology, Wuhan, China, in 2008, and the M.S. degree in software engineering from the South China University of Technology, in 2012. He is currently pursuing the Ph.D. degree in computer science with the South China University of Technology.

His current research interests include robotic software architecture, multimobile robot systems, and mobile wireless ad-hoc networks.

FANG LI received the B.S. and M.S. degrees from the Mechanical and Electronic Engineering Department, Central South University, Changsha, China, and the Ph.D. degree from the Mechanical and Automotive Engineering Department, South China University of Technology, Guangzhou, China.

She is currently a Teacher with the Computer Science and Engineering Department, South China University of Technology. Her research direction is mechatronic engineering. Her current research interests include embedded system development approach and cyber-physical system development.