Real-time Identification and Avoidance of Simultaneous Static and Dynamic Obstacles on Point Cloud for UAVs Navigation*

Han Chen[a], Peng Lu[b,*]

[a]Department of Aeronautical and Aviation Engineering, Hong Kong Polytechnic University, Hong Kong, China.
[b]Department of Mechanical Engineering, The University of Hong Kong, Hong Kong, China.

Abstract

Avoiding hybrid obstacles in unknown scenarios with an efficient flight strategy is a key challenge for unmanned aerial vehicle applications. In this paper, we introduce a more robust technique to distinguish and track dynamic obstacles from static ones with only point cloud input. Then, to achieve dynamic avoidance, we propose the forbidden pyramids method to solve the desired vehicle velocity with an efficient sampling-based method in iteration. The motion primitives are generated by solving a nonlinear optimization problem with the constraint of desired velocity and the waypoint. Furthermore, we present several techniques to deal with the position estimation error for close objects, the error for deformable objects, and the time gap between different submodules. The proposed approach is implemented to run onboard in real-time and validated extensively in simulation and hardware tests, demonstrating our superiority in tracking robustness, energy cost, and calculating time.

Keywords: Motion planning, UAVs, Point cloud, Dynamic environment

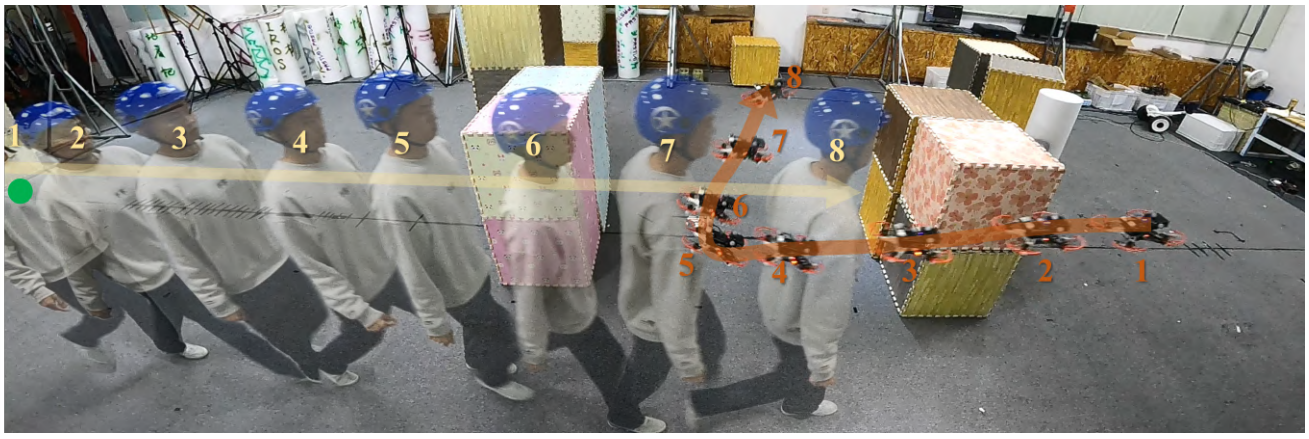


Figure 1: The composed image of one of the hardware flight tests. The drone takes off from the right side, and the goal is located at the left side, denoted by a green dot. The numbers mark the corresponding frames, increasing with time.

1. Introduction

In unknown and chaotic environments, unmanned aerial vehicles (UAVs), especially quadcopters always face rapid unexpected changes, while moving obstacles pose a greater threat than static ones. To tackle this challenge, the trajectory planner for UAVs needs to constantly and quickly generate collision-free and feasible trajectories in different scenarios, and its response time is required to be as short as possible. In addition, the optimality of the motion strategies should also be considered to save the limited energy of quadrotors.

Most existing frameworks that enable drones to generate collision-free trajectories in completely unknown environments only take into consideration stationary obstacles. However, as quadrotors often fly at low altitudes, they are faced with various moving obstacles such as vehicles and pedestrians on the ground. One primary solution to avoid collision is to raise flight altitude to fly above all the obstacles. This method is not feasible for some indoor applications, because the flight altitude is limited in

*This project is supported by the seed funding for strategic interdisciplinary research scheme of the University of Hong Kong
*Corresponding author
Email addresses: stark.chen@connect.polyu.hk (Han Chen), lupeng@hku.hk (Peng Lu)

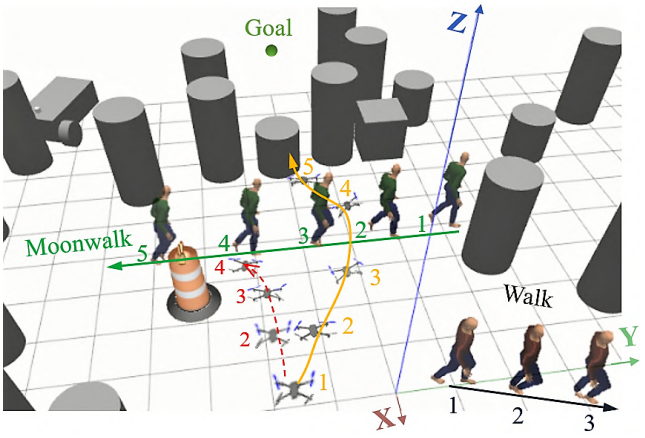


Figure 2: The composite picture of the simulation in Gazebo for the process that the drone avoids static and dynamic obstacles. 5 screenshots are used for composition and the cut time interval is fixed to 0.7 seconds. The line with an arrowhead shows the moving direction and the numbers mark the corresponding frame, the numbers increase over time. The yellow line is generated by the method in this paper, while the red line is by the static planning method.

narrow indoor space, and the drones are often requested to interact with humans as well. Another solution is to assume all detected obstacles as static. But this method cannot guarantee the safety of the trajectory [1], considering measurement errors from the sensors and unmissable displacement of the dynamic obstacles.

As shown in Figure 2, the collision may happen if the motion information of obstacles is ignored (the red line). Therefore, a more efficient and safer way to avoid moving obstacles is to predict and consider the obstacles' position in advance based on the velocity, which can avoid detours or deadlock on some occasions.

To fly in dynamic scenes for micro aerial platforms, we propose a complete system in this paper, composed of a position and velocity estimator for moving obstacles, an upper-level planner to obtain the desired velocity, and a motion planner to generate final motion primitives. Considering the limited computation resource and demands for low cost, all the computation involved should be light and do not require high-precision sensor data. For the perception of dynamic obstacles, the dynamic ones are identified from static ones by clustering and comparing the displacement from two point cloud data frames. An RGB-D camera is the only sensor utilized to obtain the point cloud. First, we set up a Kalman filter for each dynamic obstacle for tracking and output more accurate and continuous estimating results. The feature vector for each obstacle is adopted to improve the obstacle matching accuracy and robustness, thus the dynamic obstacle tracking and position and velocity estimating performance are improved compared to the related existing works. In addition, we introduce the track point to reduce the displacement estimation error involved by the self-occlusion of the obstacle.

Then, with the estimated position and velocity of obstacles and the current states and kinodynamic limitations of a real vehicle, the forbidden pyramids method is applied to plan the desired velocity to avoid obstacles. The desired velocity is obtained from a sampling-based method in the feasible space, and the sampled velocity with the minimal acceleration cost is chosen. Finally, the motion primitives are efficiently solved from a well-designed nonlinear optimization problem, where the desired velocity is the constraint. For navigation tasks, the proposed velocity planning method is also flexible to combine with most path planning algorithms for static environments, giving them the ability to avoid dynamic obstacles. The waypoint in the path acts as the trajectory endpoint constraints at a further time horizon. In this paper, we test it with our former proposed waypoint planning method heuristic angular search (HAS) [2] to complete the system and conduct the flight tests. The safety and the lower acceleration cost of this method can be verified by the data from flight tests. The computational efficiency of the whole system also shows great advantages over state-of-the-art (SOTA) works [3]-[4].

In summary, the main contributions of the paper are as follows:

- The feature vector is introduced to help match the corresponding obstacle in two point cloud frames. It is proved to be more robust than existing works that match the obstacles with only position information predicted by the Kalman filter. The neighbor frame overlapping and ego-motion compensation techniques are further introduced to reduce the estimating errors of the obstacle's position.
- To compensate for the resultant displacement estimation error from the self-occlusion of obstacles, the object track point is proposed.
- Based on the relative velocity, the forbidden pyramids method is designed to efficiently plan the safe desired velocity to avoid both dynamic and static obstacles. The various time gaps which may cause control lag error are also well compensated.
- We integrate those proposed methods and a path planning method into a complete quadrotor system, demonstrating its reliable performance in flight tests as shown in Figure 1. Also, the code is open-source for the community's reference[1].

2. Related work

At present, there are many methods for path planning and obstacle avoidance in static environments. Although some researchers have published their safe planning framework for UAVs in an unknown static scenario, it is a more complex problem for a UAV with a single depth camera flying in an unknown environment with dynamic obstacles.

[1] https://github.com/arclab-hku/dynamic_navigation

For identifying and tracking moving obstacles from the environment, most researchers employ the raw image from the camera and mark the corresponding pixels before measuring the depth. The semantic segmentation network with a moving consistency check method based on images can distinguish the dynamic objects [5]. Also, a block-based motion estimation method to identify the moving obstacle is used in [6], but the result is poor if the background is complex. Some work [7]-[8] segment the depth image and regard the points with similar depth belong to one object. But such methods cannot present the dynamic environment accurately because static and dynamic obstacles are not classified. If only human is considered as moving obstacle, the human face recognition technology can be applied [9]. However, the above-mentioned works do not estimate the obstacle velocity and position. A multi-purpose approach is proposed in [10], which jointly estimates the camera position, stereo depth, and object detections, and tracks the trajectories. Some works adopt feature-based vision systems to detect dynamic objects [11]-[12], which require dense feature points. Also, detector-based or segmentation-network-based methods can work well in predefined classes such as pedestrians or cars [13]. However, they cannot handle generic environments. Considering the limited resource of the onboard microcomputer, the above image-based methods are computation-expensive and thus not able to run in real-time.

Based on the point cloud data, it is also possible to estimate the moving obstacles' position and velocity in the self-driving cars [14]-[15]. However, they all rely on high-quality point clouds from LiDAR sensors and powerful GPUs to detect obstacles from only predefined classes. To enhance the versatility, tracking point clusters with the global feature has been proved a practical idea in simple environments [16]. As for the point cloud of a depth camera, the existing works are rare and they all match the obstacle by only the center position of obstacles, depending on the Kalman filter to predict the position of dynamic obstacles from past to present. However, this may fail when the predicted position of one obstacle is close to other obstacles at present. Varying from them, we propose the feature vector for each obstacle to tackle this challenge, and the matching robustness and accuracy are improved a lot. Our method also can be run at a higher frequency with low computational power. Event cameras can distinguish between static and dynamic objects and enable the drone to avoid the dynamic ones in a very short time [17]. However, the event camera is expensive for low-cost UAVs, and the high-quality depth information of the obstacle may also rely on another depth camera because the generated events are sparse [18].

In terms of the avoidance of moving obstacles for navigation tasks, the majority of research works are based on the applications of ground vehicles. The forbidden velocity map [19] is designed to solve out all the forbidden 2D velocity vectors and they are represented as two separate areas in the map. The artificial potential field (APF) method can avoid the moving obstacles by considering their moving directions [20]-[21]. For UAVs, the model predictive control (MPC) method is tested, but the time cost is too large for real-time flight [3]. The probabilistic safety barrier certificates (PrSBC) define the space of admissible control actions that are probabilistically safe, which is more compatible for multi-robot systems [22]. Recently, some global planners for UAV navigation in crowded dynamic environment are proposed [23]-[24]. However, the states of all obstacles are known, they are not suitable for a fully autonomous aerial platform. [4] utilizes the kinodynamic A* algorithm to find a feasible initial trajectory first and the parameterized B-spline is used to optimize the trajectory from the gradient. However, it requires dense samples along the trajectory in the optimization problem, and the object function composes the integration of the whole trajectory. Our motion optimization method is more efficient in computation.

3. Technical approach

Our proposed framework is composed of two submodules that run parallelly and asynchronously: the obstacle classifier and motion state estimator (section 3.1 & 3.2) and the waypoint and motion planner (section 4.1 & 4.2). The additional technical details for improving the accuracy of dynamic perception are introduced in section 3.3. Figure 3 illustrates the whole framework, including the important message flowing between the submodules.

3.1. Obstacle tracking

The raw point cloud is first filtered to remove the noise and converted into earth coordinate $E-XYZ$. The details about the filter are in section 5. We use Pcl_{t_1} and Pcl_{t_2} to denote two point cloud frames from the sensor at the former timestamp t_1 and the latest timestamp t_2 respectively. $t_2 - t_1 = d_t$ and $d_t > 0$. The time interval d_t is set to be able to make the displacement of the dynamic obstacles obvious enough to be observed, while maintaining an acceptable delay to output the estimation results. Pcl_{t_1} and Pcl_{t_2} are updated continuously while the sensor is operating. To deal with the movement of the camera between t_1 and t_2, Pcl_{t_2} is filtered, only keeping the points in the overlapped area of the camera's FOVs at t_1 and t_2 [4]. The newly appeared obstacles in the latest frame are removed, so only the obstacles appear in both of the two point cloud frames are further analyzed. Pcl_{t_1} and Pcl_{t_2} are clustered into individual objects using density-based spatial clustering of applications with noise (DBSCAN) [25], resulting in two sets of clusters $OB_1 = \{ob_{11}, ob_{12}, ...\}$ and $OB_2 = \{ob_{21}, ob_{22}, ...\}$. Then, matching the two clusters $ob_{2k} \in OB_2$ and $ob_{1j} \in OB_1$ corresponding to the same obstacle is necessary before the dynamic obstacle identification.

At the time when we obtain the first frame of Pcl_{t_2}, a list of Kalman filters with the constant velocity model

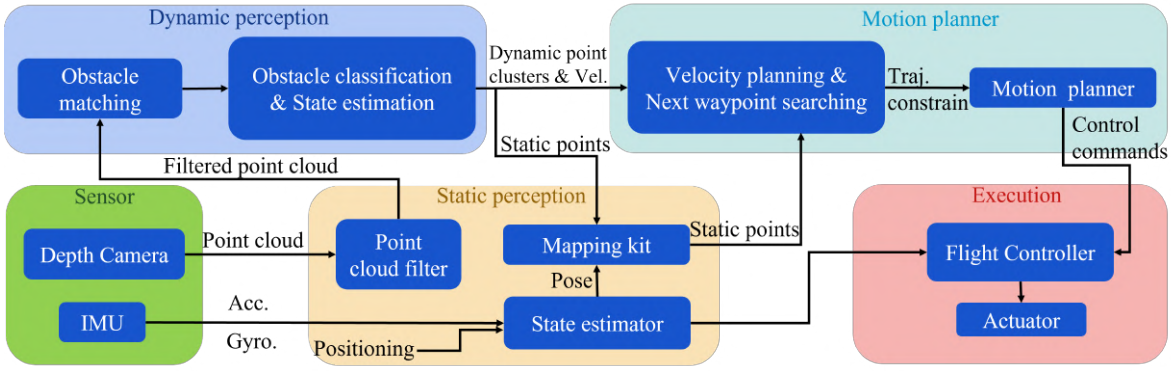


Figure 3: The proposed system for the autonomous navigation in dynamic environments. The positioning can be done by the outer motion capture system or onboard VIO toolkit.

is initialized for each cluster (obstacle) in OB_2. The position and velocity are updated after the obstacle is matched and the observation values are obtained. To match the obstacle, we preliminarily sort out the two clusters that satisfies the condition $\|pos(ob_{2k}) - pos(ob_{1j})\|_2 < d_m$. d_m is the distance threshold. $pos() \in \mathbb{R}^3$ gives the obstacle position when the input cluster is from the latest frame Pcl_{t_2}, while it returns the predicted position at t_2 by the corresponding Kalman filter of the cluster from Pcl_{t_1} [4]. It is designed to associate current clusters to the forward propagated Kalman filters rather than clusters in the previous frame. If we cannot find an ob_{1j} for ob_{2k} there, we skip it and turn to the next cluster ($k \leftarrow k+1$, k is the cluster index). For each Kalman filter, it is also necessary to assign a reasonable maximal propagating time t_{kf} before being matched with a new observation. Because the camera FOV is narrow, we hope to predict the clusters which just move out of the FOV for safety consideration, and they are assumed to continue to move at their latest updated velocity in a short period. A Kalman filter is deleted together with its tracking history if it has not been matched for over t_{kf}.

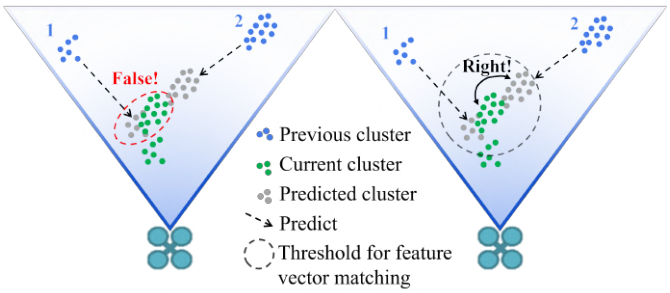


Figure 4: The left figure shows a situation that two obstacles are mismatched. The predicted cluster of obstacle 1 is closer than the predicted cluster 2 to the current cluster of obstacle 2. By comparing the feature vector, the correct predicted cluster for obstacle 2 can be matched for the current cluster, as shown in the right figure.

In the related works [4, 26], matching the clusters in two frames as the same object is based only on the center position. However, it may fail when obstacles are getting close, as shown in Figure 4. To improve the matching robustness, we design a novel technique based on the feature vector to help match the clusters. The feature vector is composed of several statistic characters of a point cluster with aligned color information from the obstacle, which is defined as

$$fte(ob) = [len(ob), V_P^A(ob), V(ob), M_C(ob), V_C^A(ob)], \quad (1)$$

where ob denotes any point cluster. $len()$ is the function that returns the size of the input cluster. $V_P^A() \in \mathbb{R}^3$ returns the position variance of the cluster, and $V()$ returns the volume of the axis-aligned bounding box (AABB) of the cluster. $M_C() \in \mathbb{R}^3$ and $V_C^A() \in \mathbb{R}^3$ return the mean and variance of the RGB value of points. The idea is: if there is not a significant difference in the shape and color of the two point clusters extracted from two timely close point cloud frames respectively, then they are commonly believed to be the same object. The global features for each obstacle are very cheap to calculate and proved to be effective in tests.

At last, the Euclidean distance $d_{fte} = \|fte(ob_{1j}) - fte(ob_{2k})\|_2$ between feature vectors is calculated with each cluster pair $\{ob_{2k}, ob_{1j}\}$ that satisfies the position threshold. For each cluster $ob_{2k} \in OB_2$, the cluster $ob_{1J} \in OB_1$ results in the minimal d_{fte} is matched with it. The feature vector is normalized to 0-1 since the order of magnitude of each element varies. We use $ob_2^m \in OB_2$ and $ob_1^m \in OB_1$ to represent any two successfully matched clusters.

3.2. Obstacle Velocity Estimation and Classification

Here, we will introduce the track point which effectively reduces the velocity estimation error compared to the existing methods.

After the obstacles in two sensor frames are matched in pairs, the velocity of the obstacle ob_2^m can be calculated by $v_2^m = \overrightarrow{p_2^m p_1^m}/(t_2 - t_1)$, where p_2^m and p_1^m are the position of the corresponding obstacle. The obstacles have certain shapes, they are not points. In the related works, the mean of the points in each cluster is adopted as the obstacle position, since it is easy to calculate and close to the centroid for a common obstacle if we ignore the self-occlusion. However, due to the self-occlusion, the backside

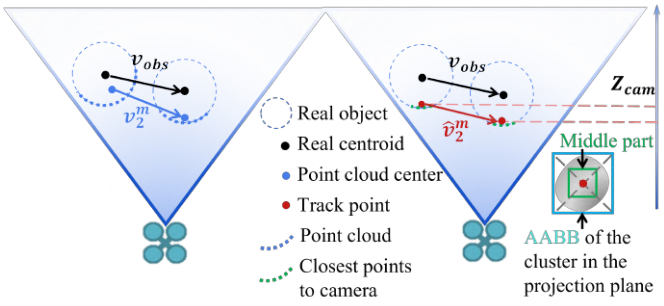


Figure 5: The left figure illustrates the velocity estimation error caused by the self-occlusion of the obstacle. v_{obs} is the velocity ground truth. When the obstacle approaches the camera, the visible part shrink, resulting in the relative displacement between the point cloud center and obstacle centroid. The track point in the right figure can reduce the velocity estimation error. The middle part of the cluster is bounded by the green box.

of the obstacle is invisible, so the mean of points is closer to the camera than the obstacle centroid. In addition, the occluded part of a moving obstacle changes when the relative movement occurs between the camera and obstacle. Thus, the relative position between the position mean and the real mass center also changes. This will lead to a wrongly estimated velocity, as shown in Figure 5, the error is mainly distributed on the Z axis of the camera Z_{cam} and it is not a fixed error can be estimated. As a consequence, the constant velocity model in the Kalman filters will no longer hold even for constant velocity obstacles. For the position estimation of obstacles, the self-occlusion is not important, considering the visible part only is safe for obstacle avoidance. But the velocity is the key information of moving obstacles in the vehicle velocity planning. Therefore, we propose a method to reduce this velocity estimation error by choosing the appropriate track point for the matched pair of clusters, as illustrated in the right figure of Figure 5.

For a moving obstacle in translational motion, the closest part to the camera is believed not to be self-occluded in ob_2^m and ob_1^m. In addition, the middle part of the cluster is close to the centroid of the common obstacles, so the rotational movement is weak in this part. Thus, we only use the center point $\hat{p}_2^m$ and $\hat{p}_1^m$ of the closest N_c points to the camera in the middle part of the cluster ob_2^m and ob_1^m to estimate the displacement. $\hat{p}_2^m$ and $\hat{p}_1^m$ are named as the track point. Here the distance to the camera is measured only along Z_{cam}. The middle part of the cluster is divided in the projection plane corresponding to the depth image. The bounding box for the middle part is shrunk from the AABB of the obstacle to the center proportionally. The shrinking scale factor is α ($\alpha < 1$). Considering the common obstacles usually performs slow rotation, and the time gap d_t is small, we can neglect the influence on the closest part caused by rotation in the displacement estimation. To update the Kalman filter, p_2^m (the mean of current cluster) is still the observed position, but the velocity observation is $\hat{v}_2^m = \overrightarrow{\hat{p}_1^m \hat{p}_2^m}/(t_2 - t_1)$. The classification for static and dynamic obstacle is done by comparing the velocity magnitude with a pre-assigned threshold v_{dy}, i.e. $\|\hat{v}_2^m\|_2 > v_{dy}$ indicates a moving obstacle. If an obstacle is classified as static in S_c consecutive times, the corresponding Kalman filter is abandoned and the static point cluster is forwarded for map fusion.

The Kalman filter for one cluster is detailed with

$$\hat{x}_t^- = F_t \hat{x}_{t-1} + B_t a_{t-1}, \tag{2}$$

$$P_t^- = F_t P_{t-1} F_t^T + Q, \tag{3}$$

$$K_t = P_t^- H^T \left(H P_t^- H^T + R\right)^{-1}, \tag{4}$$

$$P_t = (I - K_t H) P_t^-, \tag{5}$$

$$\hat{x}_t = \begin{cases} \hat{x}_t^- + K_t \left(x_t - H\hat{x}_t^-\right) & \text{(found dynamic obstacle)}, \\ \hat{x}_t^- & \text{(no dynamic obstacle)}, \end{cases} \tag{6}$$

$$x_t = \begin{bmatrix} p_2^m \\ \hat{v}_2^m \end{bmatrix}, F_t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, B_t = \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}, \tag{7}$$

where R, H, Q are the observation noise covariance matrix, observation matrix, and process noise covariance matrix respectively. The superscript $^-$ indicates a matrix is before being updated by the Kalman gain matrix K_t, applicable for the state matrix $\hat{x}_t$ and the posterior error covariance matrix P_t. The subscripts t and $t-1$ distinguish the current and the former step of the Kalman filter. F_t is the state transition matrix and B_t is the control matrix. x_t is composed of the observation of the obstacle position and velocity, and $\hat{}$ marks the filtered results for x_t. $\hat{x}_t$ equals to the predicted state $\hat{x}_t^-$ if no dynamic obstacle is caught. $\hat{x}_t^-$ is also utilized as the propagated cluster state in the obstacle matching introduced above. Δt is the time interval between each run of the Kalman filter. In the current stage, we assume the moving obstacle performs uniform motion between t_1 and t_2, $a_{t-1} = 0$.

We summarize our proposed dynamic environment perception method in **Algorithm 1**.

3.3. Ego-motion compensation and neighbor data overlapping

To improve the estimation accuracy, we notice the time gap between the latest point cloud message from the camera and the vehicle state message from the IMU in the flight controller, which is an important detail ignored in the existing works. A constant acceleration motion model is adopted to describe the vehicle motion in a short period, and the ego-motion compensation can be done with

$$\hat{p}_{cam} = p_{cam} + v_{cam} t_{gap} + \frac{1}{2} a_{cam} t_{gap}^2, \tag{8}$$

to result in the compensated camera pose $\hat{p}_{cam}$. p_{cam}, v_{cam} and $a_{cam} \in \mathbb{R}^{2\times 3}$ are the pose, velocity and acceleration of the camera obtained from raw data (translational and rotational motion), translated from the installation matrix

Algorithm 1 Dynamic environment perception

while true: **do**
  Obtain Pcl_{t_1}, Pcl_{t_2} from the point cloud filter, cluster them to OB_1, OB_2
  if it is the first loop **then**
    Initialize the Kalman filters for each cluster in OB_2
  end if
  Predict the position of former clusters with the Kalman filters
  for ob_{2k} in OB_2 (k is the iteration number): **do**
    Match ob_{2k} with the predicted clusters
    if successfully match the clusters: **then**
      Estimate the velocity of ob_{2k} with the paired ob_{1j}
      Classify it as static or dynamic and record the class as the history together with the corresponding Kalman filter
      if the cluster marked as static for S_c consecutive times: **then**
        Delete the corresponding Kalman filter, submit ob_{2k} for map fusion
      else if ob_{2k} is dynamic: **then**
        Update the Kalman filter with p_2^m and $\hat{v}_2^m$
      end if
    end if
  end for
  if no dynamic obstacle is found: **then**
    Update the Kalman filter with the predicted state
  end if
end while

of the camera. t_{gap} is the time gap between the message, equals to the timestamp of point cloud minus the timestamp of vehicle state. As a result, the point cloud can be converted to $E-XYZ$ more precisely.

For non-rigid moving obstacles, for example, walking animals (including humans), the body posture is continuously changing. The point cloud deformation may cause additional position and velocity estimation error of obstacle since the waving limbs of a walking human to interfere with the current estimation measurements. We notice that when two neighbor frames of point cloud are overlaid, the point cloud of the human trunk is denser than the other parts which rotate over the trunk. Then, an appropriate point density threshold of DBSCAN can remove the points corresponding to the limbs. So the overlapped point cloud can replace the filtered raw point cloud. For instance, the w^{th} point cloud frame Pcl_w is replaced with $Pcl_w \cup Pcl_{w-1}$. $\hat{p}_{cam}$ is also replaced by the mean of the value from its neighbor data frame, to align with the point cloud.

4. Motion planning

The motion of the drone is more aggressive for avoiding moving obstacles than flying in a static environment. To address the displacement of the drone during the time costed by the trajectory planner and flight controller, position compensation is adopted before the velocity planning, which is another important detail usually not mentioned in the references. The current position of drone p_n is updated by the prediction

$$\hat{p}_n = p_n + (t_{pl} + t_{ct} + t_{pm})v_n + \frac{1}{2}(t_{pl} + t_{ct} + t_{pm})^2 a_n, \quad (9)$$

where t_{pl} is the time cost of the former step of the motion planner. t_{ct} is the estimated fixed responding time for the flight controller. t_{pm} is the timestamp gap between UAV pose and current time. In addition, due to the time cost of obstacle identification and communication delay, the timestamp on the information of dynamic obstacles is always later than that of the pose and velocity message of the drone. Based on the constant velocity assumption, the obstacle position $\hat{p}_2^m$ in the planner at the current time is predicted and updated as

$$\hat{p}_2^m = p_2^m + (t_{pl} + t_{ct} + t_{pm} + t_{dp})\hat{v}_2^m, \quad (10)$$

where t_{dp} is the timestamp gap between the UAV pose and the received dynamic clusters. The dynamic clusters published by the perception module share the same timestamp with the latest point cloud pcl_{t2}.

4.1. Velocity planning

This subsection will introduce a novel velocity planning method based on the relative velocity and the forbidden pyramids. First, the planner receives the moving clusters and the velocity from the dynamic perception module. The currently unclassified clusters are also conveyed to the planner and treated as static obstacles together with the classified static clusters. In addition, we adopt the mapping kit to offer the static environmental information out of the current FOV, because the FOV of a single camera is narrow. To tackle the autonomous navigation tasks, the drone is required to reach the goal position. The desired velocity of the drone is initialized as v'_{des}, $\|v'_{des}\|_2 = v_{max}$ and v'_{des} heads towards the goal. v_{max} is the maximum speed constraint. Also, considering the path optimality, a path planner is usually adopted in the navigation. Thus, the waypoint w_p generated from the planned path is used to replace the navigation goal if a path planner is required. Otherwise, w_p denotes the navigation goal. w_p can be generated from a guidance law or assigned directly as the first waypoint in the path to enable the drone to follow the path. It is a choice to combine the velocity planning method with the path planning algorithms to adapt to navigation applications better. Figure 6 illustrates the collision check by calculating the relative velocity of v'_{des} towards the obstacles, and if the check fails the velocity

re-planning will be conducted. For dynamic obstacles, the relative velocity equals v'_{des} minus the obstacle velocity. For static obstacles, the relative velocity is v_n itself. If v'_{des} is checked to be safe, the finally desired velocity v_{des} is given by v'_{des}.

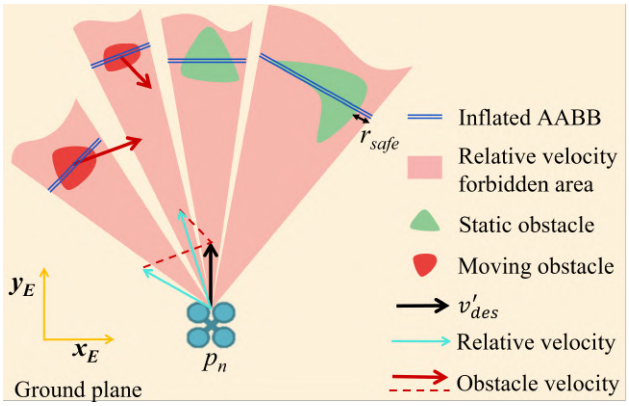


Figure 6: Check if the current relative velocities towards each obstacle lies in the forbidden area. In this figure, the relative velocity towards one dynamic obstacle and one static obstacle all fail the collision check. The forbidden area is the projection area (space) of the inflated obstacle AABB in the projection plane to the camera. r_{safe} is the inflating size.

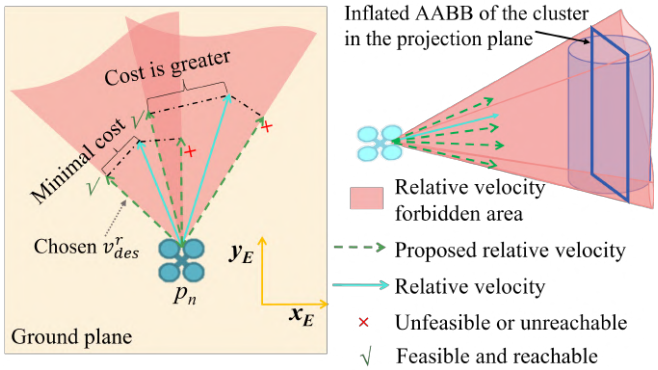


Figure 7: The left figure explains the velocity planning for multiple forbidden pyramids. We use a floor plan to better demonstrate the method. The right figure is a forbidden pyramid for one obstacle in 3D view, four sides of the pyramid result in four proposed relative velocity vectors. "Unreachable" refers to that a relative velocity is out of the maximal velocity bound of the drone, which is detailed in Figure 9.

Then, the velocity re-planning method is explained in Figures 7-9, the forbidden area (space) is extended to a pyramid for the 3D case, different from the triangle for the 2D case. Here, we introduce a hypothesis that the object in the environment does not have a ring topology, thus the flight trajectory through a single object is forbidden. If the relative velocity lies in the corresponding forbidden pyramid, four proposed relative velocity vectors are found by drawing perpendicular lines from the relative velocity vector perpendicular to the four sides of the pyramid. They are the samples to be checked later. The vertical line segments stand for the acceleration cost to control the vehicle to reach the proposed velocity. For any cluster, v_{rel} denotes the relative velocity, the five vertices

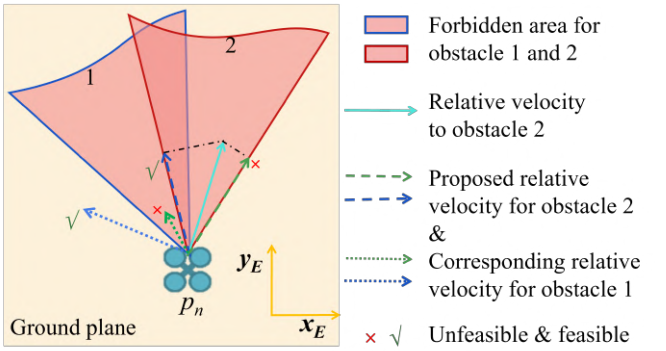


Figure 8: The feasibility check of the relative velocity for one obstacle, as the supplementary for Figure 7. The proposed relative velocity is checked if feasible for other moving obstacles. In this figure, the left proposed relative velocity for obstacle 2 is also feasible for obstacle 1 (navy blue arrows), while the right one (green arrows) is not.

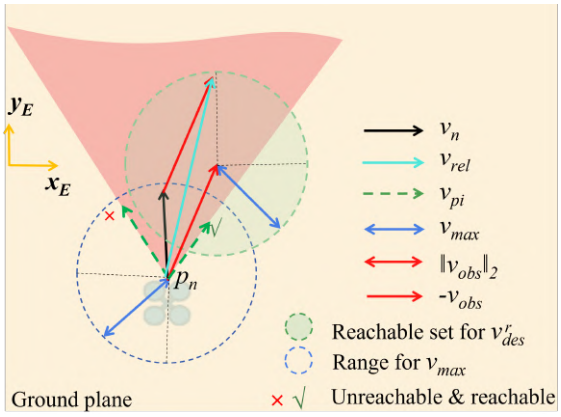


Figure 9: The reachable check for the proposed relative velocities. v_{obs} is moved to start from p_n, and the endpoint is the center of the spherical reachable set. The possible relative velocity constrained by v_{max} towards this obstacle is included in this set. Only the relative velocity vectors in the reachable set are chosen.

of the forbidden pyramid are

$$\{\hat{p}_n(x_0, y_0, z_0), vt_1(x_1, y_1, z_1), ..., vt_4(x_4, y_4, z_4)\}. \quad (11)$$

The acceleration cost a_{ci} of the proposed relative velocity vectors v_{pi} $(i \in \{1, 2, 3, 4\})$ can be calculated by solving the 3D geometric equations, as follows:

$$C^p = \overrightarrow{\hat{p}_n vt_1} \times \overrightarrow{\hat{p}_n vt_2}, \quad (12)$$

$$c^p_d = -C^p \hat{p}^T_n, \quad (13)$$

$$a_{ci} = \frac{|C^p(v_{rel} + \hat{p}_n)^T + c^p_d|}{\|C^p\|_2}, \quad (14)$$

$$v_{pi} = v_{rel} - C^p \frac{C^p(v_{rel} + \hat{p}_n)^T + c^p_d}{\|C^p\|^2_2}. \quad (15)$$

In (12)-(15), triangle $\{\hat{p}_n, vt_1, vt_2\}$ is taken as the example, $C^p[x, y, z]^T + c^p_d = 0$ is the corresponding plane equation, $\hat{p}_n$ is the common vertex of all the 4 triangles.

Obviously, for only one obstacle, the desired relative velocity with the minimal cost is from the four proposed ones. For multiple obstacles and forbidden pyramids, the desired relative velocity is chosen by comparing the feasibility, reachability, and cost. Among all the proposed relative velocity vectors, the one checked to be feasible and reachable and with the minimal cost is selected (v^r_{des}), and the desired vehicle velocity $v_{des} = v^r_{des} + v_{obs}$. v_{obs} is the velocity for the corresponding obstacle. Although the globally optimal solution for acceleration cost cannot be guaranteed within the samples, the computation complexity is greatly reduced compared to solving the optimal solution. We use the inflated bounding box because the character radius of the vehicle r_{uav} can not be neglected.

The feasibility check is to guarantee v_{des} is safe for all obstacles, not for only one of them, which is described in Figure 8. Besides the feasibility check, v_{des} should satisfy the maximum speed constraints v_{max}. We introduce the reachable set for the relative velocity vector to check if the proposed relative velocity v_{pi} ($i \in \{1, 2, 3, 4\}$) is reachable, as Figure 9 indicates. For the relative velocity towards one obstacle, $v_{rel} = v_n - v_{obs}$ always hold. v_{rel} is the current relative velocity towards the obstacle, v_{obs} is the obstacle velocity.

In addition, the lag error of the velocity planning caused by the time cost to reach the desired velocity is also considerable. The relative displacement between the moving obstacle and vehicle during this time gap should be estimated, because the forbidden pyramid is also directly related to the relative position. We can assume the solved jerk J_n is very close to its boundary J_{max}, because the time cost t_v is minimized in the optimization problem (20). Thus, the time cost is estimated as

$$\begin{aligned} v_n + a_n t_v + \frac{1}{2} J_n t_v^2 = v_{des} \\ \Rightarrow t_v = \min_{t_v} \{ \| \frac{2(v_{des} - v_n - a_n t_v)}{t_v^2} \|_\infty = J_{max} \wedge t_v > 0 \}, \end{aligned} \tag{16}$$

and the displacement of the vehicle and obstacle is calculated by

$$d_{uav} = v_n t_v + \frac{1}{2} a_n t_v^2 + \frac{1}{6} J_n t_v^3, \tag{17}$$

$$d_{obs} = \hat{v}_2^m t_v. \tag{18}$$

At last, the estimated displacement d_{uav} and d_{obs} are added to the position after v_{des} is obtained, and a new v_{des} is planned in iteration until it is checked to be safe. The accurate time cost t_v can only be determined after solving the motion optimization problem. However, involving the optimization problem in the iteration will be time-consuming, so we use a closed-form solution as the approximate value. To speed up the convergence, the safety radius is inflated by a small value ϵ (equivalent to the tolerance in the safety check) to calculate v_{pi}:

$$r_{safe} = r_{uav} + \epsilon \ (\epsilon > 0). \tag{19}$$

In a situation where the obstacles are too dense, the forbidden area may cover all the space around the vehicle. We first sort all the clusters with the increasing order of distance to p_n, the farther obstacles are considered less threatening for the drone. Then, the last j clusters are excluded, j is the iteration number increasing from 0. Algorithm 2 reveals the process of velocity planning. As a result, the vehicle can always quickly plan the velocity to avoid static and dynamic obstacles and follow the path to meet the different task requirements.

Algorithm 2 Velocity planning

$v'_{des} \leftarrow v_{max} \frac{\overrightarrow{\hat{p}_n w_p}}{|w_p - \hat{p}_n|}$
if v'_{des} is unsafe (Figure 6): **then**
  $j \leftarrow 0$
  Sort the clusters with the distance to p_n
  while v_{des} is not found: **do**
    Remove the last j clusters from original sequence
    Get all the feasible relative velocity vectors for the remained clusters
    if feasible and reachable relative velocity exist: **then**
      Choose v^r_{des} with the minimal acceleration cost, and $v_{des} \leftarrow v^r_{des} + v_{obs}$
    end if
    $j \leftarrow j + 1$
  end while
  repeat
    $\hat{p}_n \leftarrow \hat{p}_n + d_{uav}$, $\hat{p}_2^m \leftarrow \hat{p}_2^m + d_{obs}$
    Repeat line 7-10 with updated forbidden pyramids
  until v_{des} is safe
else
  $v_{des} \leftarrow v'_{des}$
end if

4.2. Motion planning

After the desired velocity v_{des} is obtained, it appears as the constraint in the motion planning and will be reached in a short time. The waypoint constraints w_p is also considered to follow the path, as shown in Figure 10.

The optimization problem to obtain motion primitives is constructed as

$$\begin{aligned} \min_{J_n, t_v} \quad & \eta_1 t_v + \eta_2 d_{trj} \\ \text{s.t.} \quad & a_{n+1} = a_n + J_n t_v, \\ & v_{des} = v_n + a_n t_v + \frac{1}{2} J_n t_v^2, \\ & d_{trj} = \frac{\|\overrightarrow{\hat{p}_n p_{end}} \times \overrightarrow{w_p p_{end}}\|_2}{|w_p - \hat{p}_n|}, \\ & p_{end} = \hat{p}_n + v_n K t_v + \frac{1}{2} a_n (K t_v)^2 + \frac{1}{6} J_n (K t_v)^3, \\ & 0 < t_v, \ \|a_{n+1}\|_2 \le a_{\max}, \ \|J_n\|_2 < J_{max}, \end{aligned} \tag{20}$$

where the jerk of the vehicle J_n is the variable to be optimized. t_v is the time required to reach v_{des}, which is the variable and the optimization object at the same time. a_{n+1} and p_{end} are calculated by the kinematic formula. a_{max} and J_{max} are the kinodynamic constraints of acceleration and jerk of the vehicle respectively. The velocity constraint v_{max} is satisfied in the equality constraint with v_{des}. η_1, η_2 are coefficients. The default values are shown in Table 1. After the desired trajectory piece is solved, a default cascade PID controller of PX4 is utilized to track this trajectory in position, velocity, and acceleration.

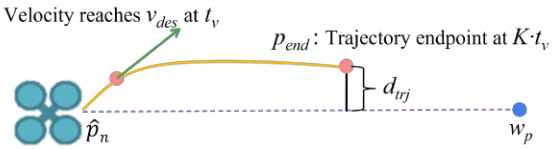


Figure 10: The proposed motion planning method. The objective function is designed to minimize the time cost to reach the desired velocity and the distance from trajectory endpoint p_{end} to the path line. The solid yellow line represents the predicted trajectory.

5. Experimental implementation and results

5.1. Point cloud filters

For the dynamic environment perception, filtering the raw point cloud is necessary, because the obstacle state estimation is sensitive to the noise. The noise should be eliminated strictly, even losing a few true object points is acceptable. The filter has the same structure as our former work [2], as shown in Figure 11, but the parameters are different. The distance filter removes the points too far ($\geq 6.5\ m$) from the camera, the voxel filter keeps only one point in one fixed-size ($0.1\ m$) voxel, outlier filter removes the point that does not have enough neighbors (≤ 13) in a certain radius ($0.25\ m$). Based on such configuration, the density threshold for DBSCAN is at least 18 points in the radius of $0.3\ m$. These metrics are tuned manually during extensive tests on the hardware platform introduced in the next subsection, to balance the point cloud quality and the depth detection distance. They are proved satisfactory for obstacle position estimation. The point cloud filtering also reduces the message size by one to two orders of magnitude, so the computation efficiency is much improved, while the reliability of the collision check is not affected.

However, when the drone performs an aggressive maneuver, the pose estimation of the camera (including the ego-motion compensation) is not accurate enough for dynamic obstacle perception. To solve this problem, we propose a practical and effective measurement: The filtered point cloud is accepted only when the angular velocity of the three Euler angles of the drone is within the limit ω_{max}.

5.2. Map building

In our implementation, we adopt a simple method to store the static points in a list ("mapping kit" in Fig. 3),

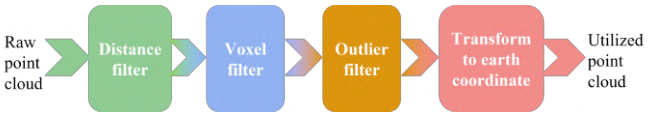


Figure 11: The filtering process for the raw point cloud.

and visualize the points as the point cloud map. After the first dynamic obstacle identification, we push all the static clusters into a list for initialization. When a new static point cluster is found afterward, we compare the distance between the new static cluster center with the existing clusters' centers in the list. If the new cluster is very close to the existing static clusters, it will not be added to the list to avoid duplicating and saving the RAM of the onboard computer. In the velocity planning, only the static clusters in the list in the range of $6.5\ m$ (the cut-off depth of the distance filter) to the vehicle are considered. To obtain a high-quality map, the existing mapping toolkits (such as Octomap) are also capable in our system.

5.3. Experimental Configuration

The detection and avoidance of obstacles are tested and verified in the Robot Operation System (ROS)/Gazebo simulation environment first and then in the hardware experiment. The drone model used in the simulation is 3DR IRIS, and the underlying flight controller is the PX4 1.10.1 firmware version. The depth camera model is an Intel Realsense D435i with a resolution of 424*240 (30 fps). For hardware experiments, we use a self-assembled quadrotor with a Q250 frame and a LattePanda Alpha 864s with an Intel m3-8100y processor, other configuration keeps unchanged. A motion capture system VICON is adopted to obtain the pose of the drone. Table 1 shows the parameter settings for the simulation tests. The supplementary videos for the tests have been uploaded online[2][3].

Table 1: Parameters for the tests

Parameter	Value	Parameter	Value
S_c	3	t_{ct}	0.01 s
v_{dy}	0.3 m/s	d_t	0.2 s
t_{kf}	0.7 s	d_m	0.9 m
N_c	12	α	0.5
η_1	10	η_2	6
K	3	ϵ	0.05 m
a_{max}	6 m/s^2	ω_{max}	1.5 rad/s
J_{max}	12 m/s^3	v_{max}	1.5 m/s

5.4. Simulation Test

5.4.1. Dynamic perception module test

First, the accuracy and stability of the estimation method for the obstacle position and velocity are verified.

[2]https://youtu.be/1g9vHfoycs0
[3]https://www.youtube.com/watch?v=5CwFATodSvU

In the simulation world depicted in Figure 12(a), there is one moving ball, two moving human models, and some static objects. The moving obstacles reciprocate on different straight trajectories. The camera is fixed on the head of the drone, facing forward straightly. Since the point cloud from the simulated sensor is clean and the noise is very light, the distance filter threshold is extended to 8 m. The drone is hovering around the point $(-6, 0, 1.2)$. Figure 12(b) depicts the visualized estimation results in Rviz. The Euclidean distance of the feature vectors utilized for obstacle tracking is illustrated in Figure 13. The estimation numeral results are shown in Figure 14, and they are compared with the ground truth. The dynamic perception performance is also compared with SOTA works in Table 2, where the metrics Multiple Object Tracking Precision (MOTP) and Multiple Object Tracking Accuracy (MOTA) are adopted as defined in the work of Bernardin [27]. MOTP is the average position estimation error in this test. Only walking or running pedestrians are tested in Table 2. The second line marked with * is for our method without using the track point to correct the velocity estimation, and the third line marked with # is for our method without the neighbor frame overlapping. We record the point cloud (at 30 Hz) and UAV states data (at 100 Hz) for about 600 s, and repeat the test on the data 5 times to give the average results.

The estimation test results in Gazebo simulation demonstrate that our estimation algorithm is practical for dynamic obstacle avoidance. In addition, our method efficiently improves the estimation accuracy and robustness in the clustered environment. For our MOTA, it is composed of a false negatives rate $f_n = 6.7\%$ (covering non-detected dynamic objects and dynamic objects erroneously classified as static or uncertain), a false positives rate $f_p = 6.9\%$ (static objects misclassified as dynamic), and a mismatch rate $f_m = 2.1\%$.

Table 2: Obstacle State Estimation Comparison

Method	$error_{vel}(m/s)$	MOTA (%)	MOTP
Ours	**0.21**	**84.3**	**0.15**
Ours*	0.29	83.9	0.16
Ours#	0.25	83.6	0.18
[4]	0.37	76.4	0.28
[3]	0.41	70.1	0.30

5.4.2. Motion planning module test

In addition, we compare the motion planning method with [4] and [28] in Table 3. The metrics a_{mean}, v_{mean}, l_{traj} and t_{opt} stand for the average acceleration, the average velocity, the average flight trajectory length and the time cost for the motion optimization part. The time costs are measured on a laptop computer with an Intel i7-8550U CPU and 8 GB RAM. Similarly in [4], we consider the environment with only dynamic obstacles, because the locations and velocities of all obstacles are known and considered as dynamic in [28]. The obstacles are ellipsoids with human-like size (0.5×0.5×1.8m) and move at constant velocities, as shown in Figure 15. For each planner, the drone flies between two points $(0, 0, 1.2)$ and $(20, 0, 1.2)$ back and forth for 10 times, 20 obstacles with velocities at 1-3 m/s cross this straight path disorderly. The camera FOV is also considered, and simulated to be $85.2° \times 58°$ with the maximal sensing depth 8 m. From Table 3 we conclude that the average acceleration cost of our motion planning method is smaller because our velocity planning method considers the minimal acceleration cost in all the sample velocities for the current time. Also, the computing time is much shorter thanks to the simple but efficient object function and constraints, which shows the potential to avoid faster obstacles. In these tests, our planning approach produces a longer trajectory because the farther obstacles are more likely to be ignored when the obstacles are dense. Only the obstacle close to the drone is considered sometimes, the trajectory optimality in length from the global view is weak compared to the compared works, as they optimize the trajectory with all the obstacles in the sensing range. In addition, the average speed is smaller be-

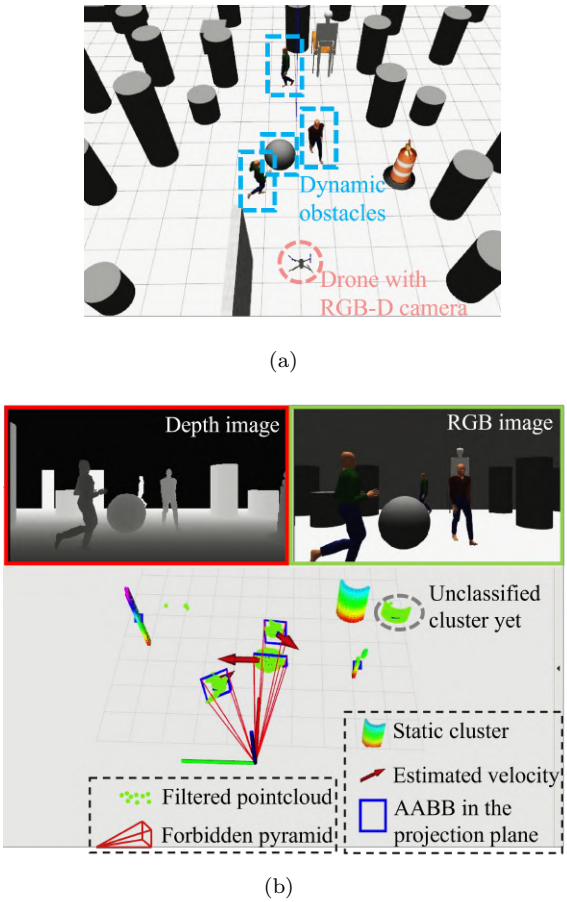


Figure 12: (a): The simulation environment for the moving obstacles' position and velocity estimation test. (b): The visualized estimation results in RVIZ, corresponding to (a). Only the forbidden pyramids for dynamic clusters are visualized. The pedestrians always face their moving direction. It can be seen that the obstacles are correctly tracked even though they are very close.

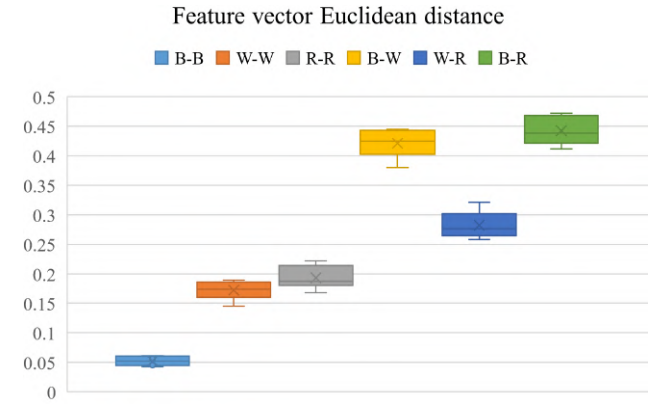


Figure 13: The box chart of the Euclidean distance of the feature vector $fte()$ between obstacles from OB_1 and OB_2. B, W, and R represent the moving **B**all, **W**alking and **R**unning person in Figure 12 respectively. The distance of the same obstacle is obviously lower than that of different obstacles, so the obstacles are matched correctly.

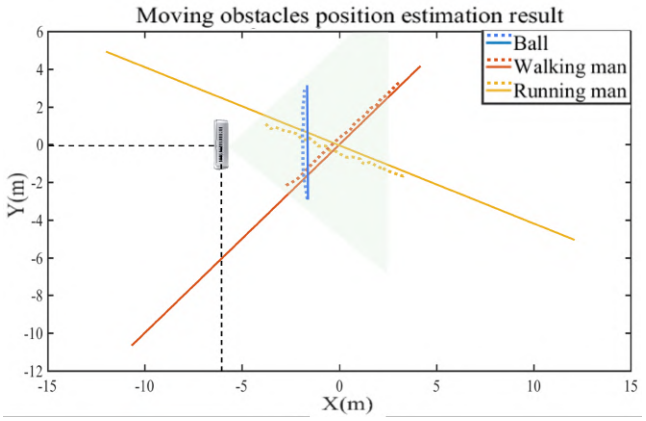


Figure 14: The estimation results of the moving obstacles' position. The FOV of the camera is represented with a light green area. The dotted line is the estimated result, while the solid line is the ground truth.

cause our velocity planning approach is based on sampling and the unreachable samples are simply abandoned, the drone's movement capability is not fully used.

5.4.3. System test

To test the whole framework for navigation tasks, we utilize the HAS method [2] as the path planning algorithm at the front end to generate the waypoint w_p for equation (20). The flight simulation world is revealed in Figure 2 and 12(a), there are four walking or running pedestrians, one moving ball, and many static pillars and boxes. In Figure 2, the necessity for estimating the obstacle's velocity is illustrated: To avoid the moving man which is at a similar speed to the drone, the aircraft choose to fly in the "opposite" direction from the man so the threat is removed easily. If only the static HAS method [2] is utilized in the same situation, the drone decelerates and flies alongside the man (red line), which is very inefficient and dangerous.

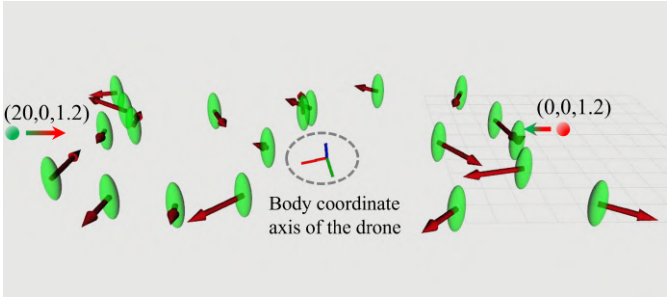


Figure 15: The simulated test environment for the motion planning module. The drone flies between the two points for the assigned times. The red arrows represent the velocity vectors of dynamic obstacles.

Table 3: Dynamic Planning Comparison

Method	$a_{mean}(m/s^2)$	$v_{mean}(m/s)$	$l_{traj}(m)$	$t_{opt}(ms)$
Ours	**2.96**	2.24	25.21	**3.15**
[4]	3.43	2.37	23.65	8.61
[28]	3.18	2.33	22.96	31.23

5.5. Hardware Test

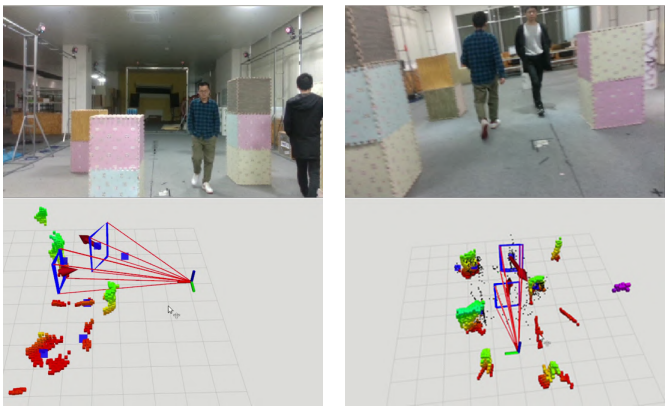
5.5.1. Perception module evaluation

In the above simulation tests, the proposed perception module is verified with the simulated depth camera. However, the noise of the point cloud from a real depth camera is much more severe. In subsection 5.1, the parameters of the point cloud filters for the real camera are tuned to eliminate the ghost points (the points that do not correspond to any real obstacles), but still a few ghost points remained. Also, the points for real obstacles are not accurate as those in simulation, especially the depth error is greater for the farther objects. The vehicle state estimation also has a greater error than that in simulation, which adds additional error when transforming the points from the body to the earth coordinate. Therefore, some parameters for perception should be adjusted before flight tests. We collect over 430 s data of the raw point cloud (at 30 Hz), raw RGB image (at 30 Hz), and the vehicle states (at 100 Hz) under VICON in different scenarios, and study the influence on the dynamic obstacle detecting and tracking performance from the parameters. As a result, v_{dy} and d_t are found to be more influential than other parameters. In Table 4, we use MOTA (%) to evaluate the performance under different configurations with the collected data, and repeat the test 5 times for each configuration. Other metrics are discarded since the position and velocity ground truth of a moving pedestrian are complex to obtain. (a), (b), (c) in the first row refers to the different test scenarios, and the scenarios are introduced in Fig 16.

We can conclude from Table 4 that greater v_{dy} and d_t are helpful to suppress the noise and depth error in obstacle tracking. However, v_{dy} should be much smaller than the slowest object in the environment to make the

Table 4: Obstacle tracking performance under different parameters

v_{dy}(m/s), d_t(s)	MOTA(a)	MOTA(b)	MOTA(c)
0.2, 0.2	69.43	65.26	59.68
0.3, 0.2	71.91	67.71	60.26
0.5, 0.2	75.86	71.12	64.11
0.9, 0.2	74.37	70.59	59.14
0.5, 0.1	71.65	68.24	64.87
0.5, 0.3	78.45	76.12	71.36
0.5, 0.5	74.57	69.11	55.73
0.5, 0.7	66.62	62.78	48.83



(a) (b)

Figure 16: The images from the onboard camera of the dynamic perception test scenarios and the visualized results. Two pedestrians are walking among several boxes and pillars. (a): The camera is fixed, for MOTA(a). (b): The camera is held by hands and moving at around 1.5 m/s and 2.5 m/s, for MOTA(b) and MOTA(c) respectively.

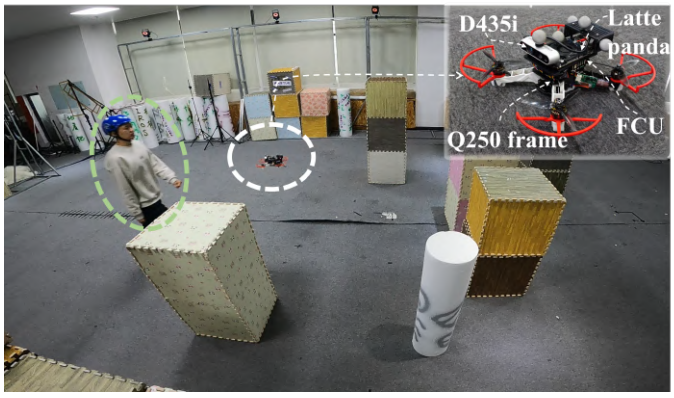


Figure 17: The dynamic hardware test environment. The aerial platform is introduced in the upper right corner. The pedestrian walks directly through the area while the drone is flying among the static obstacles.

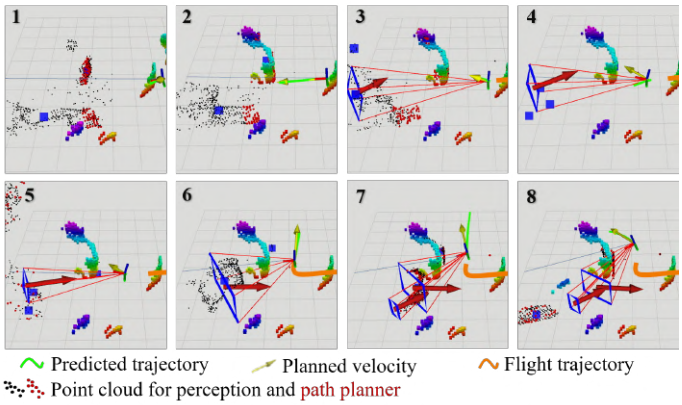


Figure 18: The corresponding visualized data in RVIZ for the frames in Figure 1

classification robust to the velocity estimation error. If d_t is too large, due to the limited camera FOV, an object may be neglected since the continuously observed time is even shorter than d_t. According to the results, we choose $v_{dy} = 0.5$ m/s and $d_t = 0.3$ s, other parameters of the perception module stay unchanged.

5.5.2. Flight test

We set up a hardware test environment as Figure 17, the drone takes off behind the boxes and then a person enters the FOV of the camera and walks straight towards the drone during the flight to test the effectiveness of our method. In Figure 1, the camera is fixed and takes photos every 0.33 s during the flight. Eight photos are composed together. In the first frame the pedestrian appeared, the orange line shows the trajectory of the drone while the yellow line is for the person. It can be concluded that the reaction of the drone is similar to the simulation above. The visualized data for this hardware test is shown in Figure 18, the planned velocity and the predicted trajectory react promptly once the moving obstacle appears. In Figure 19, the gray line is only for the path planning algorithm (HAS), and the blue line represents the whole planning with **Algorithm 2**. The gray line has a strong positive linear correlation with time because the number of the input points of the collision check procedure determines the distance calculation times. The blue and red lines show the irregularity, because the moving obstacle brings an external computation burden to **Algorithm 2**, and the number of moving obstacles has no relation to the point cloud size. The single-step time cost of our proposed method (excluding the path planning) is even smaller than 0.01 s, indicating the fast-reacting ability towards moving obstacles.

At last, we compare our work with SOTA works on the system level in Table 5. Since most related works differ significantly from ours in terms of application background and test platform, for numeral indicators we only compare the total time cost for reference. The time cost is obtained

from the references directly, to compare roughly at orders of magnitude. The abbreviations stand for: obs (obstacle), cam (camera), UUV (underwater unmanned vehicle), N/A (not applicable). "N/A" in the sensor type column refers to the work that gets obstacle information from an external source and does not include environment perception. Most works have severe restrictions on the obstacle type or incompleteness in environment perception, and the computing time cost is not satisfactory for real-time applications. Our work has a great advantage in generality and system completeness, the computing efficiency is also at the top level.

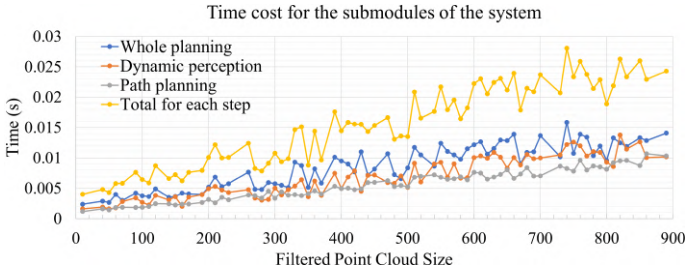


Figure 19: The time cost for different modules under different filtered point cloud size.

Table 5: System comparison between different works

Work	Sensor type	Vehicle	Obs limits	Time cost (s)
[29]	Sonar	UUV	dynamic obs	1-2
[20]	N/A	Robots	dynamic obs	0.045-0.13
[16]	Cam & Lidar	Car	**N/A**	0.1
[9]	N/A	UAV	human	0.2-0.3
[17]	Event cam	UAV	dynamic obs	**0.0035**
[3]	RGB-D cam	UAV	**N/A**	0.024
Ours	RGB-D cam	UAV	**N/A**	**0.015**

6. Conclusion and future work

In this paper, we present a computationally efficient algorithm framework for both static and dynamic obstacle avoidance for UAVs based only on point clouds. The test results indicate our work is feasible and shows great promise in practical applications.

However, when the speed or angular velocity of the drone is high, and also because of the narrow FOV of a single camera, the dynamic perception becomes significantly unreliable. In future research, we intend to improve the robustness of our method in aggressive flights and test it with different sensors such as lidar.

Acknowledgements

The authors sincerely thank FAST Lab of Zhejiang University for offering assistance with the experimental site and assisting staff.

References

[1] J. Chen, T. Liu, S. Shen, Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments, in: 2016 IEEE International Conference on Robotics and Automation (ICRA), Vol. 2016, 2016, pp. 1476–1483.

[2] H. Chen, P. Lu, Computationally efficient obstacle avoidance trajectory planner for uavs based on heuristic angular search method, arXiv preprint arXiv:2003.06136 (2020).

[3] J. Lin, H. Zhu, J. Alonso-Mora, Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 2682–2688.

[4] Y. Wang, J. Ji, Q. Wang, C. Xu, F. Gao, Autonomous flights in dynamic environments with onboard vision, arXiv preprint arXiv:2103.05870 (2021).

[5] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, Q. Fei, Ds-slam: A semantic visual slam towards dynamic environments, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1168–1174.

[6] J. Kim, Y. Do, Moving obstacle avoidance of a mobile robot using a single camera, Procedia Engineering 41 (2012) 911–916.

[7] H. Oleynikova, D. Honegger, M. Pollefeys, Reactive avoidance using embedded stereo vision for mav flight, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2015, pp. 50–56.

[8] P. Skulimowski, M. Owczarek, A. Radecki, M. Bujacz, D. Rzeszotarski, P. Strumillo, Interactive sonification of u-depth images in a navigation aid for the visually impaired, Journal on Multimodal User Interfaces 13 (3) (2019) 219–230.

[9] T. Nageli, J. Alonso-Mora, A. Domahidi, D. Rus, O. Hilliges, Real-time motion planning for aerial videography with real-time with dynamic obstacle avoidance and viewpoint optimization, in: IEEE Robotics and Automation Letters, Vol. 2, 2017, pp. 1696–1703.

[10] A. Ess, B. Leibe, K. Schindler, L. van Gool, Moving obstacle detection in highly dynamic scenes, in: 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 4451–4458.

[11] K. Berker Logoglu, H. Lezki, M. Kerim Yucel, A. Ozturk, A. Kucukkomurler, B. Karagoz, E. Erdem, A. Erdem, Feature-based efficient moving object detection for low-altitude aerial platforms, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 2119–2128.

[12] I. A. Bârsan, P. Liu, M. Pollefeys, A. Geiger, Robust dense mapping for large-scale dynamic environments, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 7510–7517.

[13] S. Zhang, R. Benenson, M. Omran, J. Hosang, B. Schiele, Towards reaching human performance in pedestrian detection, IEEE transactions on pattern analysis and machine intelligence 40 (4) (2017) 973–986.

[14] S. Kraemer, C. Stiller, M. E. Bouzouraa, Lidar-based object tracking and shape estimation using polylines and free-space information, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018, pp. 4515–4522.

[15] J. Miller, A. Hasfura, S.-Y. Liu, J. P. How, Dynamic arrival rate estimation for campus mobility on demand network graphs, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2016, pp. 2285–2292.

[16] A. Cherubini, F. Spindler, F. Chaumette, Autonomous visual navigation and laser-based moving obstacle avoidance, IEEE Transactions on Intelligent Transportation Systems 15 (5) (2014) 2101–2110.

[17] D. Falanga, K. Kleber, D. Scaramuzza, Dynamic obstacle avoidance for quadrotors with event cameras., Science Robotics 5 (40) (2020).

[18] B. He, H. Li, S. Wu, D. Wang, Z. Zhang, Q. Dong, C. Xu, F. Gao, Fast-dynamic-vision: Detection and tracking dynamic

objects with event and depth sensing, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2021, pp. 3071–3078.
[19] B. Damas, J. Santos-Victor, Avoiding moving obstacles: the forbidden velocity map, in: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 4393–4398.
[20] N. Malone, H.-T. Chiang, K. Lesser, M. Oishi, L. Tapia, Hybrid dynamic moving obstacle avoidance using a stochastic reachable set-based potential field, IEEE Transactions on Robotics 33 (5) (2017) 1124–1138.
[21] H. Febbo, J. Liu, P. Jayakumar, J. L. Stein, T. Ersal, Moving obstacle avoidance for large, high-speed autonomous ground vehicles, in: 2017 American Control Conference (ACC), 2017, pp. 5568–5573.
[22] W. Luo, W. Sun, A. Kapoor, Multi-robot collision avoidance under uncertainty with probabilistic safety barrier certificates, in: Advances in Neural Information Processing Systems, Vol. 33, 2020.
[23] C. Cao, P. Trautman, S. Iba, Dynamic channel: A planning framework for crowd navigation, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 5551–5557.
[24] D. Zhu, T. Zhou, J. Lin, Y. Fang, M. Q.-H. Meng, Online state-time trajectory planning using timed-esdf in highly dynamic environments, arXiv preprint arXiv:2010.15364 (2020).
[25] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. 1996 Int. Conf. Knowledg Discovery and Data Mining (KDD '96), 1996, pp. 226–231.
[26] T. Eppenberger, G. Cesari, M. Dymczyk, R. Siegwart, R. Dubé, Leveraging stereo-camera data for real-time dynamic obstacle detection and tracking, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 10528–10535.
[27] K. Bernardin, A. Elbs, R. Stiefelhagen, Multiple object tracking performance metrics and evaluation in a smart room environment, in: Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV, Vol. 90, Citeseer, 2006.
[28] H. Zhu, J. Alonso-Mora, Chance-constrained collision avoidance for mavs in dynamic environments, IEEE Robotics and Automation Letters 4 (2) (2019) 776–783.
[29] W. Zhang, S. Wei, Y. Teng, J. Zhang, X. Wang, Z. Yan, Dynamic obstacle avoidance for unmanned underwater vehicles based on an improved velocity obstacle method., Sensors 17 (12) (2017) 2742.