

Vision-aided UAV Navigation and Dynamic Obstacle Avoidance using Gradient-based B-spline Trajectory Optimization

Zhefan Xu, Yumeng Xiu, Xiaoyang Zhan, Baihan Chen, and Kenji Shimada

Abstract—Navigating dynamic environments requires the robot to generate collision-free trajectories and actively avoid moving obstacles. Most previous works designed path planning algorithms based on one single map representation, such as the geometric, occupancy, or ESDF map. Although they have shown success in static environments, due to the limitation of map representation, those methods cannot reliably handle static and dynamic obstacles simultaneously. To address the problem, this paper proposes a gradient-based B-spline trajectory optimization algorithm utilizing the robot’s onboard vision. The depth vision enables the robot to track and represent dynamic objects geometrically based on the voxel map. The proposed optimization first adopts the circle-based guide-point algorithm to approximate the costs and gradients for avoiding static obstacles. Then, with the vision-detected moving objects, our receding-horizon distance field is simultaneously used to prevent dynamic collisions. Finally, the iterative re-guide strategy is applied to generate the collision-free trajectory. The simulation and physical experiments prove that our method can run in real-time to navigate dynamic environments safely.

I. INTRODUCTION

Light-weight autonomous UAVs are massively deployed in various industrial applications, such as inspection, exploration, and search and rescue. The environments of those applications are usually highly complex and dynamic, involving human workers, static structures, robots, and vehicles. As one of the most fundamental components of robot autonomy, the safe trajectory planning algorithm becomes essential to let UAVs deal with complex environment structures while actively sensing and avoiding dynamic obstacles.

Safe navigation in dynamic environments mainly involves three challenges. Firstly, the robot must track and represent static and dynamic obstacles simultaneously. Recent popular planning methods [1][2] apply the voxel-based map including the occupancy map and ESDF map [3][4] as obstacle representation. Although those mapping algorithms can deal with arbitrarily complex static environments, they can hardly distinguish and capture dynamic obstacles, leading to the limited performance of the mentioned planners. Secondly, the planner should be able to generate trajectories with complicated static structures. Some vision-based algorithms [5][6][7] represent obstacles geometrically using the bounding boxes or ellipsoids. Those methods can safely avoid dynamic obstacles but might fail when environmental structures become complicated. Finally, due to the unpredictable and quickly changing environments, high-frequency real-time

Zhefan Xu, Yumeng Xiu, Xiaoyang Zhan, Baihan Chen, and Kenji Shimada are with the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA. zhefanx@andrew.cmu.edu

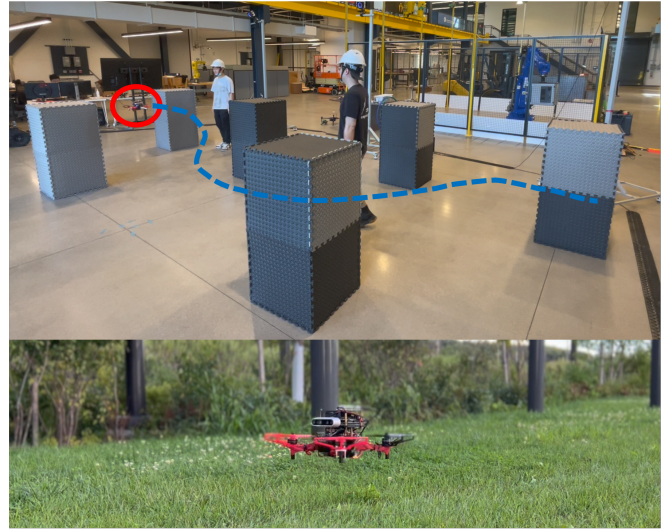


Fig. 1. The UAV navigating with obstacles using the proposed algorithm. The upper presents the physical autonomous flight in a dynamic environment. The bottom shows our customized UAV with onboard sensors.

planning is necessary to prevent dynamic collisions, which further adds burden to the limited onboard computations.

To solve these issues, this paper proposes the **Vision-aided Gradient-based B-spline Trajectory Optimization (ViGO)** algorithm. The algorithm utilizes our vision-aided 3D dynamic map, enabling tracking dynamic obstacles and representing complex static environments simultaneously. The proposed circle-based guide-point algorithm approximates the costs and gradients of static collision to improve optimization speed. With future predictions, the receding horizon distance field is applied to prevent collisions considering dynamic obstacles. Finally, the iterative re-guide strategy is used to real-time generate collision-free trajectories. Our customized UAV and the dynamic environment navigation example are shown in Fig. 1, and the main contributions of this work are:

- **Vision-aided 3D Dynamic Map:** This algorithm utilizes the depth image with the occupancy voxel map to track 3D dynamic obstacles and adopts this 3D dynamic map to perform the trajectory optimization.
- **Circle-based Guide-Point Algorithm:** We propose the guide-point algorithm to approximate the costs and gradients of static collisions for trajectory optimization.
- **Receding Horizon Distance Field:** The proposed receding horizon distance field utilizes the obstacles’ future predictions to estimate the collision costs and gradients for preventing dynamic collisions.

II. RELATED WORK

Recent years have seen many works of UAV navigation in dynamic environments. There are mainly two categories of methods based on their map representations: the voxel map-based method and the geometric map-based method. With the advantage of representing arbitrary complex 3D objects, the voxel maps such as [3][4][8] are widely used in trajectory planning. Due to the differential flatness property [9] of the quadcopter, [10] generates the trajectory using the minimum snap optimization, iteratively adding intermediate waypoints to prevent collisions. Inspired by [11][12], [1] formalizes the unconstrained optimization utilizing the distance information from the ESDF map. Later, [2] applies the collision force to avoid obstacles locally and reduce the ESDF update computation. In [13], incremental sampling is proposed to reduce the computation time for dynamic environment exploration. Although the above methods prove high computational efficiency and success in aggressive flights, their map cannot capture and distinguish the dynamic obstacle well, leading to a limited performance in highly dynamic environments.

Unlike the voxel map, the geometric map usually represents each obstacle as a single bounding box, sphere, or ellipsoid. Based on the analytical form of those geometries, early approaches propose the artificial potential field [14] and velocity obstacles [15] to generate simple control commands to prevent collisions. Recently, the model predictive control-based (MPC) methods [16][17][6][18][19] have become popular for obstacle avoidance while considering robot dynamics. In [16], the distance cost is applied for each obstacle for the trajectory penalty. [17] adopts the chance-constrained formulation and reduces computation by disjunctive programming to consider the uncertainty. Similarly, [6][18] approximates the chance constraint by linearization and achieves real-time obstacle avoidance. Those mentioned methods assume the obstacles can always be represented using geometric shapes and rely on vision-based obstacle detection [18][5]. However, when the static environmental structure becomes complex, those geometric shapes might become over conservative or even unable to model static obstacles, resulting in collisions and suboptimal performance.

A few methods also distinguish static and dynamic obstacles for trajectory planning. A two-layer planner scheme is proposed in [20] to deal with static and dynamic obstacles simultaneously. With the dual-structure particle-based map, [21] adopts the sampling-based planner to evaluate the risk of trajectories. However, their solutions can be suboptimal since they only sample from the predefined motion primitives. In [22], learning-based detection is applied to track the dynamic obstacles with the occupancy map. [23] applies point cloud-based detection with the voxel map and uses linear prediction of obstacle state to achieve safe navigation. However, their method of fully trusting linear predicted future states might be over-conservative and even fail to find the solution in some narrow environments. Unlike the previous methods, our approach utilizes the vision-aided 3D voxel map to track dynamic obstacles and applies the gradient-based trajectory

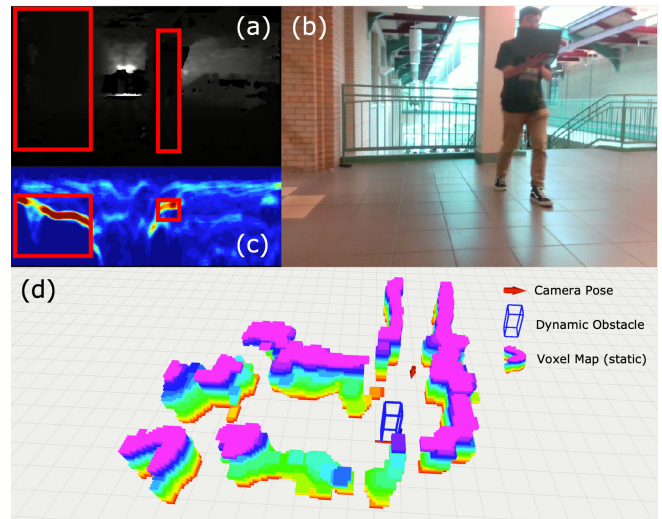


Fig. 2. Illustration of the 3D dynamic map. (a) Depth image with detected objects. (b) RGB camera view. (c) U-depth map generated by the depth image with the detected results. (d) Visualization of the dynamic map.

optimization considering the dynamic obstacles' future states in a receding horizon manner.

III. METHODOLOGY

This section presents each component of our proposed trajectory optimization framework. In Sec. III-A, the vision-aided 3D dynamic map system is introduced. Then, the unconstrained B-spline trajectory optimization is defined in Sec. III-B. The static collision cost using our circle-based guide point algorithm is discussed for static obstacle avoidance in Sec. III-C. Then, we formalize the receding horizon distance field for calculating the dynamic obstacle cost in Sec. III-D. Finally, Sec. III-E introduces the iterative re-guide optimization to generate the collision-free trajectory.

A. Vision-aided 3D Dynamic Mapping

This part introduces a lightweight mapping method to handle static and dynamic obstacles simultaneously. The dynamic map method has become popular in recent years to track dynamic voxels [24][25][26]. Unlike those methods, this method is based on the 3D occupancy map and utilizes the depth image to perform dynamic obstacle tracking. There are mainly four steps to update the map and track dynamic obstacles: region proposal detection, map-depth fusion, dynamic obstacle filtering, and dynamic region cleaning.

The proposed method first applies the depth image-based detection to find the possible obstacle regions as the proposals. Given the input depth image, we first generate the u-depth map shown in Fig. 2c, as described in [5][18]. The u-depth map can be interpreted as the top view of the objects, so the lengths and widths of potential objects can be detected as bounding boxes by line grouping shown in Fig. 2c. Then, with obstacles' widths, the bounding boxes of objects can be found in the original depth image by depth continuity checking. After performing the coordinate transformation, we can obtain 3D bounding boxes of potential objects in the world frame. Note that the obtained bounding boxes only

preserve the objects' rough positions and sizes and might include static obstacles. To refine the positions and sizes of obstacles as the second step, we enlarge those detected bounding boxes as the region proposals by an inflating factor and check the occupancy information in the voxel map, which stores discrete but more accurate information. Through this region proposal search, we can get refined results.

With the refined bounding boxes, the third step is to filter out static obstacles. We first apply the Kalman filter to track and determine the velocity of each detected object. We can remove most static obstacles by using the minimum velocity threshold criteria. However, the detection noises make some static obstacles shake back and forth slightly, leading to some velocity errors. To solve this issue, we apply our continuity filter to remove objects with jerky motions using the history of object velocities. Finally, since the static occupancy map might still keep the occupied voxel of dynamic obstacles, we apply the histories of dynamic obstacles to clean those regions. The final map output is shown in Fig. 2d.

B. B-spline Trajectory Optimization

B-spline curves are widely used in trajectory optimization [1][2] due to its local curve control ability, convex hull property, etc. A B-spline curve with order k is composed of several $k - 1$ degree polynomial basis functions with corresponding control points defined over a knot vector. Given the global trajectory or the goal position, we can parameterize the trajectory into a set of control points:

$$\hat{\mathcal{S}} = \{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \dots, \mathbf{P}_{N-1}, \mathbf{P}_N\}, \quad \mathbf{P}_i \in \mathcal{R}^3, \quad (1)$$

where the first and last $k - 1$ control points are the position of start and goal, respectively. The optimization variables are the set \mathcal{S} of the intermediate $N - 2(k - 1)$ control points.

The optimization follows the gradient-based unconstrained formulation. With the variable set \mathcal{S} , the objective cost function is defined as:

$$C_{\text{total}}(\mathcal{S}) = \alpha_{\text{control}} \cdot C_{\text{control}} + \alpha_{\text{smooth}} \cdot C_{\text{smooth}} + \alpha_{\text{static}} \cdot C_{\text{static}} + \alpha_{\text{dynamic}} \cdot C_{\text{dynamic}}, \quad (2)$$

which is a weighted combination of control limit, trajectory smoothness, static collision, and dynamic obstacle costs. The static collision cost and dynamic obstacle cost will be discussed in detail in Sec. III-C and III-D.

The control limit cost forces the trajectory to have feasible velocities and accelerations. The derivative of the B-spline curve can be represented by another B-spline, so the control points \mathbf{V}_i and \mathbf{A}_i for velocity and acceleration are:

$$\mathbf{V}_i = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{\delta t}, \quad \mathbf{A}_i = \frac{\mathbf{V}_{i+1} - \mathbf{V}_i}{\delta t}, \quad (3)$$

where δt is the time step. So, given the maximum velocity \mathbf{v}_{max} and acceleration \mathbf{a}_{max} , the cost function is defined as:

$$C_{\text{control}} = \sum_i \frac{\|\mathbf{V}_i - \mathbf{v}_{\text{max}}\|_2^2}{\lambda_{\text{vel}}} + \frac{\|\mathbf{A}_i - \mathbf{a}_{\text{max}}\|_2^2}{\lambda_{\text{acc}}}, \quad (4)$$

in which the L2 norm is applied for the control limit penalty with the unit normalization factor λ when the velocity and acceleration exceed the limits.

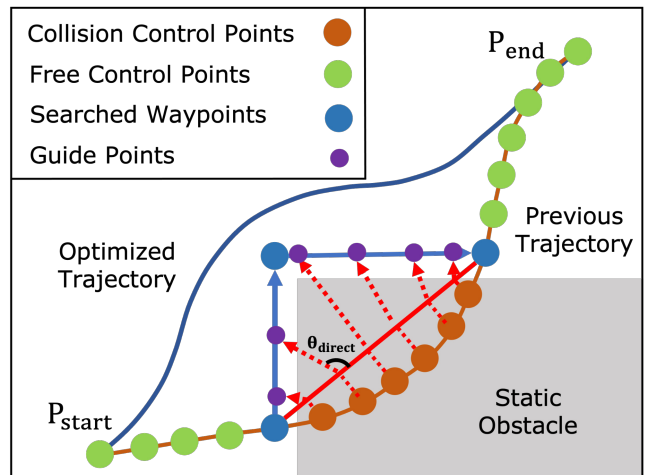


Fig. 3. Illustration of circle-based guide-point assignment. For the given collision trajectory, we first find the collision control points and search the collision-free paths. The guide points shown as purple points are the intersections between circle-based raycasting and the searched paths.

Algorithm 1: Circle-based Guide-Point Algorithm

```

1  $\mathcal{C}_{\text{guide}} \leftarrow \emptyset$ ; ▷ guide-point set
2  $\sigma_{\text{traj}} \leftarrow$  collision trajectory;
3  $\mathcal{S}_{\text{col}} \leftarrow \text{findCollisionSet}(\sigma_{\text{traj}})$ ;
4  $\mathcal{P}_{\text{search}} \leftarrow \text{pathSearch}(\mathcal{S}_{\text{col}}.\text{start}, \mathcal{S}_{\text{col}}.\text{end})$ ;
5  $\mathcal{L} \leftarrow \text{formLine}(\mathcal{P}_{\text{search}}.\text{start}, \mathcal{P}_{\text{search}}.\text{end})$ ;
6  $n \leftarrow 1$ ;  $\mathcal{N} \leftarrow \mathcal{S}_{\text{col}}.\text{size} + 1$ ;
7 for  $\mathbf{P}_c$  in  $\mathcal{S}_{\text{col}}$  do
8    $\mathbf{P}_{\text{proj}} \leftarrow \mathbf{P}_c.\text{projectOnto}(\mathcal{L})$ ;
9    $\theta_{\text{direct}} \leftarrow \frac{n}{\mathcal{N}} \cdot \pi$ ; ▷ direction angle
10   $\mathbf{P}_{\text{guide}} \leftarrow \text{raycastOntoPath}(\mathbf{P}_{\text{proj}}, \theta_{\text{direct}}, \mathcal{P}_{\text{search}})$ ;
11   $\mathcal{C}_{\text{guide}}.\text{insert}(\mathbf{P}_c, \mathbf{P}_{\text{guide}})$ ;
12   $n \leftarrow n + 1$ 
13 return  $\mathcal{C}_{\text{guide}}$ ;

```

The smoothness cost is applied to prevent the jerky trajectory. Following the same manner, we can get the control points \mathbf{J}_i for the jerk, the derivative of acceleration, and apply the following cost function:

$$C_{\text{smooth}} = \sum_i \|\mathbf{J}_i\|_2^2, \quad \mathbf{J}_i = \frac{\mathbf{A}_{i+1} - \mathbf{A}_i}{\delta t} \quad (5)$$

C. Static Collision Cost

Since the static obstacles are represented by the occupancy voxels on the map, the collision costs and gradients for the above optimization cannot be directly obtained. So, we apply the proposed circle-based guide-point algorithm to estimate the costs and gradients of static collisions.

The costs and gradients for the optimization variable, control points, can be estimated by the guide points. The procedure of finding guide points is illustrated in Fig. 3 and Alg. 1. Given a collision trajectory, the planner first find the "collision set" \mathcal{S}_{col} which consists of collision control points (Fig. 3 orange points). Inspired by [2], we apply the path-search algorithm, such as A* or Dijkstra, to locally

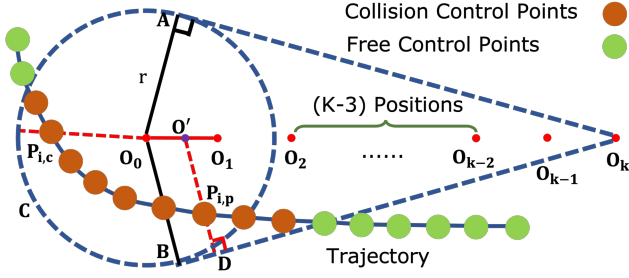


Fig. 4. Illustration of receding horizon distance field. We linearly decrease the safety distance r from the current obstacle position \mathbf{O}_0 to the last predicted position \mathbf{O}_k in a receding horizon manner.

find a collision-free path $\mathcal{P}_{\text{search}}$ bypassing the collision set. Then, we project each point in the collision set onto the connecting line between the start and goal position of the searched path (Alg. 1 Line 8). With the projected point, we calculate the direction angles θ_{direct} ranging from $(0, \pi)$ and use the point-angle pair to cast a ray onto the searched path. The intersection point between the ray and the searched path is the guide point for that collision control point shown as purple points in Fig. 3. The algorithm is circle-based because the direction angles sweep a semi-circle.

With the associated guide point and a predefined safety distance d_{safe} , the collision cost of collision control points can be calculated by:

$$C_{\text{static}} = \sum_i \left(\max \left(d_{\text{safe}} - \text{signDist}(\mathbf{P}_i, \mathbf{P}_{\text{guide}}^i), 0 \right) \right)^3, \quad (6)$$

where the signed distance defines the positive and negative distances as the control point outside and inside the obstacle, respectively. Eqn. 6 penalizes the control points whose signed distance is less than the safe distance using the cubic function. In this way, the gradient for **each collision control point** can be calculated with the chain rule:

$$\nabla g_i = 3 \left(d_{\text{safe}} - \text{signDist}(\mathbf{P}_i, \mathbf{P}_{\text{guide}}^i) \right)^2 \cdot \frac{\hat{\mathbf{P}}_i - \mathbf{P}_{\text{guide}}^i}{\|\hat{\mathbf{P}}_i - \mathbf{P}_{\text{guide}}^i\|_2}, \quad (7)$$

where $\hat{\mathbf{P}}_i$ is the initial collision control point, and the directions of the negative gradients are to push the collision control points to a free region, and Fig. 3 shows the optimized trajectory got a smooth turning to avoid the static obstacle by using the circle-based guide points.

D. Dynamic Obstacle Cost

Unlike static obstacles, the states of dynamic obstacles are changing and uncertain, making it unreliable to use only the current captured information for trajectory optimization. So, this section introduces the receding horizon distance field for collision cost estimation, which evaluates the distance to the safe area of the given points based on future predictions.

The illustration of the receding horizon distance field for dynamic obstacles is shown in Fig. 4. The dynamic obstacles are represented as positions with estimated velocities and sizes. Given the obstacle's current position \mathbf{O}_0 and velocity \mathbf{V}_0 , we apply the linear predictor to obtain the k steps'

Algorithm 2: Iterative Re-guide Optimization

```

1  $\alpha_{\text{static}}, \alpha_{\text{dynamic}} \leftarrow \alpha_0$ ; ▷ Cost Weights
2  $\lambda \leftarrow 1.5$ ; ▷ user-defined inflate factor
3  $\mathcal{P} \leftarrow$  Optimization Solver;
4  $\mathcal{S} \leftarrow$  Control Points;
5  $\mathcal{T}_{\text{sc}} \leftarrow \text{true}$ ; ▷ static collision
6  $\mathcal{T}_{\text{dc}} \leftarrow \text{true}$ ; ▷ dynamic collision
7 while  $\mathcal{T}_{\text{sc}}$  or  $\mathcal{T}_{\text{dc}}$  do
8    $\text{reguide}, \mathcal{S}_{\text{reguide}} = \mathcal{P}.\text{isReguideRequired}(\mathcal{S})$ ;
9   if  $\text{reguide}$  or  $\text{firstTime}$  then
10      $\mathcal{P}.\text{assignNewGuidePoints}(\mathcal{S}_{\text{reguide}})$ 
11   else if not  $\text{reguide}$  and  $\mathcal{T}_{\text{sc}}$  then
12      $\alpha_{\text{static}} \leftarrow \lambda \cdot \alpha_{\text{static}}$ 
13   else if not  $\text{reguide}$  and  $\mathcal{T}_{\text{dc}}$  then
14      $\alpha_{\text{dynamic}} \leftarrow \lambda \cdot \alpha_{\text{dynamic}}$ 
15    $\mathcal{S} \leftarrow \mathcal{P}.\text{updateCostAndSolve}()$ ;
16    $\sigma_{\text{traj}} \leftarrow \mathcal{P}.\text{evaluateTrajectory}(\mathcal{S})$ ;
17    $\mathcal{T}_{\text{sc}} \leftarrow \mathcal{P}.\text{checkStaticCollision}(\sigma_{\text{traj}})$ ;
18    $\mathcal{T}_{\text{dc}} \leftarrow \mathcal{P}.\text{checkDynamicCollision}(\sigma_{\text{traj}})$ ;
19 d return  $\sigma_{\text{traj}}$ ;
```

future positions $\{\mathbf{O}_1, \mathbf{O}_2, \dots, \mathbf{O}_k\}$. To construct the distance field, we draw a circle with the safe radius r centered at \mathbf{O}_0 . Then, considering the unreliability of future predictions, we linearly decrease the safe radius r to zero in a receding horizon manner, leading to a cone-shape collision region (blue dot lines). By avoiding this region, we can generate the trajectory considering future predictions while preventing over-conservative behaviors. So, for dynamic obstacle cost estimation, we first calculate their distances to the safe area considering the following two cases:

Case 1. The control point $\mathbf{P}_{i,c}$ is in the circular region enclosed by arc \widehat{ACB} , line \mathbf{AO}_0 and \mathbf{BO}_0 . The distance to safe area Δd_i can be calculated by:

$$\Delta d_i = r - \|\mathbf{P}_{i,c} - \mathbf{O}_0\|_2, \quad (8)$$

Case 2. The control point $\mathbf{P}_{i,p}$ is in the polygon region $\mathbf{AO}_0\mathbf{BO}_k$. We draw a line through $\mathbf{P}_{i,p}$ which is perpendicular to the line \mathbf{BO}_k and intersects $\mathbf{O}_0\mathbf{O}_k$ at \mathbf{O}' . The distance to safe area Δd_i can be calculated by:

$$\Delta d_i = \|\mathbf{D} - \mathbf{O}'\|_2 - \|\mathbf{P}_{i,p} - \mathbf{O}'\|_2. \quad (9)$$

With the distance to safe area Δd_i , we can finally calculate the costs for all the collision control points by:

$$C_{\text{dynamic}} = \sum_i \left(\max(\Delta d_i, 0) \right)^3, \quad (10)$$

and the gradient for each collision control point can be calculated using the chain rule with Eqns. 8 and 9.

E. Iterative Re-guide Optimization

The proposed optimization problem is unconstrained and involves multiple objectives, so solving it once might not guarantee trajectory safety. So, we iteratively solve this problem using the re-guide strategy presented in Alg. 2 until the entire trajectory is collision-free. From our observations,

the designed static collision and dynamic obstacle costs help the optimizer find the collision-free trajectory, but it might fail for the following reasons. First, the collision cost weights in Eqn. 2 are not large enough. For this scenario, Alg. 2 initializes the weights α_{static} and α_{dynamic} for static and dynamic collision cost with a user-defined cost inflate factor λ (Lines 1-2) and increase the collision cost weights if they are too small (Lines 12 and 14). Second, control points might be pushed towards new obstacles after optimization due to the "push force" from cost functions. In this case, we treat those control points as the re-guide required points as the previous collision cost and gradient approximations become invalid. So, Alg. 2 checks whether there exists the re-guide required points (Line 8) in each iteration and assigns new guide points to those control points (Line 10). The process will repeat until a collision-free trajectory is generated.

IV. RESULT AND DISCUSSION

To evaluate the proposed method's performance, we conduct simulation experiments and physical flight tests in dynamic environments. The algorithm implementation is based on C++ and ROS running on Intel i7-10750H@2.6GHz for simulation experiments and Nvidia Xavier NX for physical tests. The runtime for each component of our system is shown in Fig. 5. Overall, the entire system can run in real-time for the laptop and the UAV's onboard computer. The 3D dynamic map contains the obstacle tracking module and voxel mapping module, both using a small computational cost and taking less than 20ms each iteration. The proposed planner can run up to around 100Hz by the UAV's onboard computer to guarantee a fast response to dynamic obstacles.

A. Obstacle Tracking Evaluation

To quantitatively evaluate the performance of our dynamic obstacle tracking method, we measure the errors in simulation and physical experiments using the external motion capture system (Fig. 6). The collected data is shown in Table I. The table results present the tracking algorithm's mean position, velocity, and obstacle size measurement errors. One can observe that both errors for position and velocity are relatively small in the simulation compared to the physical tests. This larger errors are mainly caused by the noises from physical depth images. The position and velocity errors in the real world are with the mean of 0.19m and 0.21m/s, which are reasonable considering the low computation requirement and safe for dynamic obstacle avoidance. The obstacle size measurement errors are acceptable, with a mean of 0.25m for both simulation and physical tests.

TABLE I
MEASUREMENT OF DETECTION AND TRACKING ERRORS.

Errors	Simulation Tests	Physical Tests
Position Error (m)	0.09	0.19
Velocity Error (m/s)	0.10	0.21
Size Error (m)	0.25	0.25

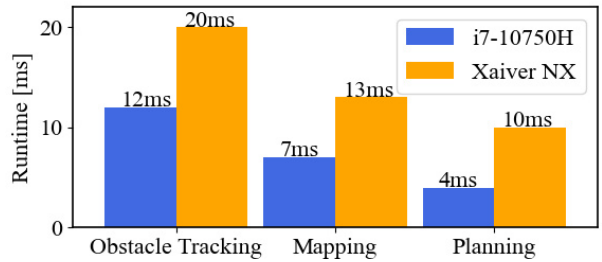


Fig. 5. The recorded average runtime for each component of our system. The entire system is able to run in real-time by the onboard computer.

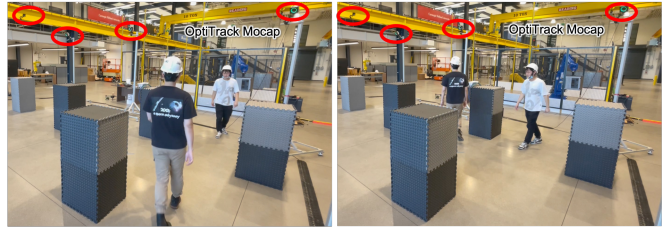


Fig. 6. Illustration of using the OptiTrack motion capture system to estimate the error of the proposed dynamic obstacle detection and tracking method.

B. Simulation Experiments

The simulation experiments include three dynamic environments of various static structures. Fig. 7 shows the corridor environment with the robot avoiding obstacles. We evaluate and benchmark the performance quantitatively using the collision, freezing, and success rates. The freezing rate denotes the rate of the robot failing to generate the trajectory and being forced to stop. For each environment, we select several target points for the robot to navigate and count the collision, freezing, and success times. We ran the experiments 20 times in the three environments (420 trajectories in total) and obtained the results in Table II. Note that we do not have the pre-built map for navigation, and the robot needs to navigate through the environments with only the given goal positions and observations from its sensor.

The state-of-the-art local planner [2] is used for performance benchmarking. We also include our methods without vision detection (ViGO w/o Vision) and without the receding horizon distance field (ViGO w/o RH) to analyze the effects. From Table II, we can see that our proposed method has the highest total success rate. The EGO-Planner and our approach without vision detection have similar results in dynamic environments. From our observations, those failures are mainly from the latency of voxel map updates for moving obstacles, so adding vision-aided obstacle tracking to the system can largely improve the success rate. Besides, the performance of our method without the receding horizon distance field is slightly worse, with a lower success rate and higher collision rate. In the experiments, the method using the receding horizon distance field can make the robot act to obstacles earlier based on future predictions, which can help improve safety. In addition, our method has a higher freezing rate than its counterparts, which is the primary cause of failure. This phenomenon is reasonable since we apply

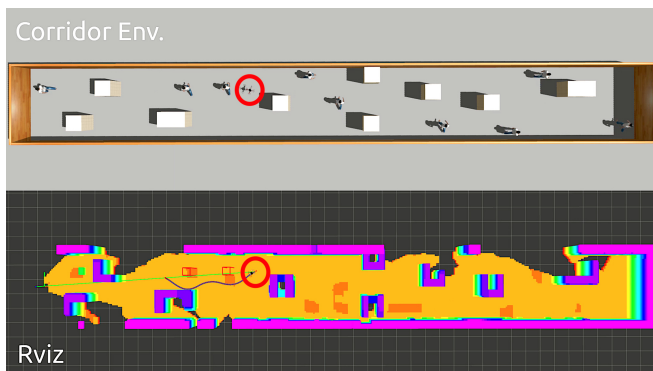


Fig. 7. The corridor environment simulation. The robot (shown in the red circle) safely navigates through the environment without a pre-built map.

the more strict constraints on dynamic obstacles, and it is effective as it increases the success rate.

TABLE II

THE BENCHMARK OF THE COLLISION RATE, FREEZING RATE, AND SUCCESS RATE IN THE SIMULATION EXPERIMENTS.

Methods	Collision Rate	Freezing Rate	Success Rate
EGO-Planner [2]	31.67%	2.14%	66.19%
ViGO w/o Vision	34.52%	0.95%	64.53%
ViGO w/o RH	7.14%	4.76%	88.10%
ViGO (Ours)	1.43%	5.95%	92.62%

C. Physical Flight Tests

The physical experiments are conducted to verify the performance of the proposed method with the example target environments shown in Fig. 1 and Fig. 8. Our customized quadcopter (shown in Fig. 1) is equipped with an Intel Realsense D435i depth camera with Nvidia Xavier NX onboard computer. We apply the visual-inertial odometry (VIO) [27] to estimate the robot's position and velocity. The PX4-based flight controller is used to control the quadcopter to track the trajectories. The computation of all modules, including obstacle tracking, mapping, localization, and trajectory optimization, is performed onboard by the quadcopter.

An example of a physical flight experiment is shown in Fig. 8. In the experiment, we have two persons walking at around 1m/s and trying to block the robot's moving directions. The robot must navigate towards the goal location and avoid static and dynamic obstacles without a pre-built map. From the bottom two figures, we can see that the robot is able to detect and track the dynamic obstacles as the blue bounding boxes and build the static map shown as the colorful voxels. When the robot has potential collisions, it can successfully generate the trajectory (blue curves) to bypass static and dynamic obstacles. The robot history trajectory is presented as the red curves. The velocity profile of this physical flight test is shown in Fig. 9. The data are collected using the onboard state estimation, and the maximum velocity is around 2m/s. By repeating the experiment with different static environment structures, we verify that our proposed methods can successfully perform navigation and obstacle avoidance in dynamic environments.

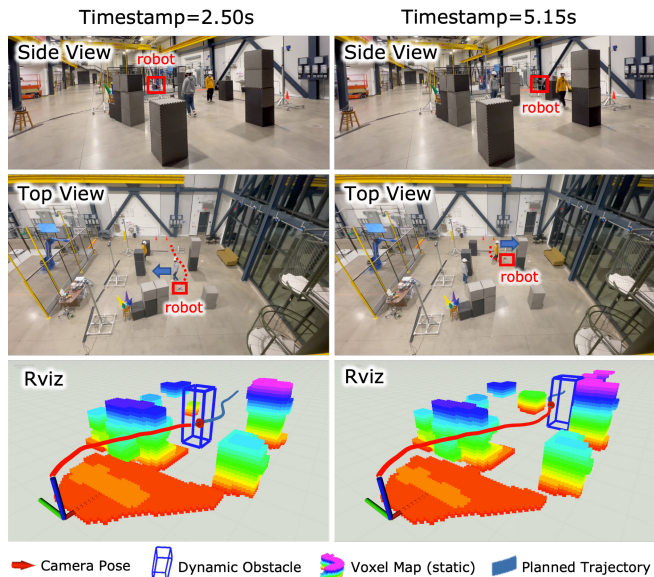


Fig. 8. Illustration of a physical flight test in a dynamic environment. The side and top views from different timestamps are shown in the upper figures, and the Rviz visualization is presented at the bottom.

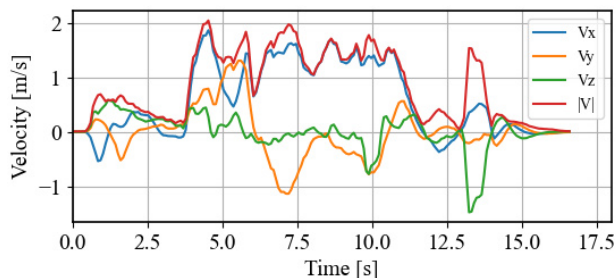


Fig. 9. The velocity profile of a physical flight test for each axis. The velocity data are obtained using the onboard visual-inertial state estimation.

V. CONCLUSION AND FUTURE WORK

This paper presents our vision-aided gradient-based B-spline optimization (**ViGO**) for navigating dynamic environments. The proposed method utilizes the vision-aided dynamic map to track dynamic obstacles. Our circle-based guide point algorithm is applied to estimate the costs and gradients for static collisions, while the receding horizon distance field is used to calculate dynamic obstacle costs. The proposed method follows the iterative re-guide optimization to generate a safe trajectory for navigation and obstacle avoidance. The simulation shows that our approach outperforms the state-of-the-art planner's success rate by using the vision obstacle tracking and receding horizon distance field. The physical experiments prove that our system can successfully avoid moving obstacles and run in real-time. For future improvement, we will focus on the 3D obstacle detection and tracking accuracy and apply multiple cameras to increase the tracking field of view.

VI. ACKNOWLEDGEMENT

The authors would like to thank TOPRISE CO., LTD and Obayashi Corporation for their financial support in this work.

REFERENCES

- [1] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.
- [2] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.
- [3] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1366–1373.
- [4] L. Han, F. Gao, B. Zhou, and S. Shen, "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4423–4430.
- [5] H. Oleynikova, D. Honegger, and 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.
- [6] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for mavs in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.
- [7] J. Lin, H. Zhu, and 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.
- [8] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.
- [9] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.
- [10] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics research*. Springer, 2016, pp. 649–666.
- [11] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494.
- [12] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4569–4574.
- [13] Z. Xu, D. Deng, and K. Shimada, "Autonomous uav exploration of dynamic environments via incremental sampling and probabilistic roadmap," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2729–2736, 2021.
- [14] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1985, pp. 500–505.
- [15] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.
- [16] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 236–243.
- [17] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.
- [18] J. Lin, H. Zhu, and 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.
- [19] B. Guo, N. Guo, and Z. Cen, "Obstacle avoidance with dynamic avoidance risk region for mobile robots in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 5850–5857, 2022.
- [20] Z. Xu, D. Deng, Y. Dong, and K. Shimada, "Dpmc-planner: A real-time uav trajectory planning framework for complex static environments with dynamic obstacles," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 250–256.
- [21] G. Chen, P. Peng, P. Zhang, and W. Dong, "Risk-aware trajectory sampling for quadrotor obstacle avoidance in dynamic environments," *arXiv preprint arXiv:2201.06645*, 2022.
- [22] T. Eppenberger, G. Cesari, M. Dymczyk, R. Siegwart, and 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. 10 528–10 535.
- [23] Y. Wang, J. Ji, Q. Wang, C. Xu, and F. Gao, "Autonomous flights in dynamic environments with onboard vision," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1966–1973.
- [24] D. Nuss, S. Reuter, M. Thom, T. Yuan, G. Krehl, M. Maile, A. Gern, and K. Dietmayer, "A random finite set approach for dynamic occupancy grid maps with real-time application," *The International Journal of Robotics Research*, vol. 37, no. 8, pp. 841–866, 2018.
- [25] M. Schreiber, V. Belagiannis, C. Gläser, and K. Dietmayer, "Dynamic occupancy grid mapping with recurrent neural networks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6717–6724.
- [26] Y. Min, D.-U. Kim, and H.-L. Choi, "Kernel-based 3-d dynamic occupancy mapping with particle tracking," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5268–5274.
- [27] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.