A Vision-Based Autonomous UAV Inspection Framework for Unknown Tunnel Construction Sites With Dynamic Obstacles

Zhefan Xu , Baihan Chen, Xiaoyang Zhan, Yumeng Xiu, *Member, IEEE*, Christopher Suzuki, and Kenji Shimada

Abstract—Tunnel construction using the drill-and-blast method requires the 3D measurement of the excavation front to evaluate underbreak locations. Considering the inspection and measurement task's safety, cost, and efficiency, deploying lightweight autonomous robots, such as unmanned aerial vehicles (UAV), becomes more necessary and popular. Most of the previous works use a prior map for inspection viewpoint determination and do not consider dynamic obstacles. To maximally increase the level of autonomy, this letter proposes a vision-based UAV inspection framework for dynamic tunnel environments without using a prior map. Our approach utilizes a hierarchical planning scheme, decomposing the inspection problem into different levels. The high-level decision maker first determines the task for the robot and generates the target point. Then, the mid-level path planner finds the waypoint path and optimizes the collision-free static trajectory. Finally, the static trajectory will be fed into the low-level local planner to avoid dynamic obstacles and navigate to the target point. Besides, our framework contains a novel dynamic map module that can simultaneously track dynamic obstacles and represent static obstacles based on an RGB-D camera. After inspection, the Structure-from-Motion (SfM) pipeline is applied to generate the 3D shape of the target. To our best knowledge, this is the first time autonomous inspection has been realized in unknown and dynamic tunnel environments. Our flight experiments in a real tunnel prove that our method can autonomously inspect the tunnel excavation front surface.

Index Terms—Field robotics, motion and path planning, perception and autonomy, robotics and automation in construction.

I. INTRODUCTION

DRILLING and blasting is a common tunnel construction and excavation method. The main cycle of this method includes steps such as drilling for explosives, blasting, measuring underbreaks, and spraying concrete. Among these steps, measuring underbreaks in the tunnel excavation front is

dangerous for workers because of the potential falling rocks. With the emergence of lightweight unmanned aerial vehicles, the robot becomes suitable for handling measurement and inspection tasks as it can avoid potential human dangers and inspect unreachable locations. Consequently, an autonomous inspection framework is essential to improve the safety and efficiency of underbreaks measurement and tunnel construction.

There are two main challenges of autonomous UAV inspection in tunnel environments. First, since the tunnel environments under construction are changing with time, it is unlikely to have update-to-date maps of huge construction vehicles and equipment nearby the excavation front. In this way, the robot should be able to navigate from arbitrary positions in the tunnel towards the excavation front area (i.e., the end of the tunnel) based on the onboard sensing. Previous works of the sampling-based unknown exploration [1], [2], [3], [4] can make the robot successfully navigate and map unknown environments with the onboard sensor and applies this exploration method to the unknown tunnel inspection [5]. However, because their approaches only utilize the explored map information to randomly sample viewpoints, the output trajectory could be zigzag and over-conservative, making navigation less efficient. The second challenge comes from the moving workers and machines in tunnels, as the robot should track them and avoid them safely. Even though some recent research [6], [7], [8] has investigated the UAV dynamic obstacle avoidance problems, their local planning strategies without global path fusion make them insufficient for complex inspection tasks in tunnel environments, which contain complicated static structures and unpredictable dynamic obstacles.

To solve these issues, this paper proposes a vision-based autonomous UAV inspection framework for unknown and dynamic tunnel environments. We develop a small, lightweight quadcopter with an RGB-D camera for safely sharing and operating with vehicles, equipment and workers in the tunnel. The proposed approach utilizes the hierarchical planning method decomposing the entire inspection planning into high, mid, and low levels. The current task is determined at the high planning level to generate the goal position for navigation and exploration. Then, the mid-level planner will find and optimize a smooth trajectory toward the goal based on the static obstacle information from the incrementally built map. Finally, at the low level, our vision-aided gradient-based planner is applied to locally
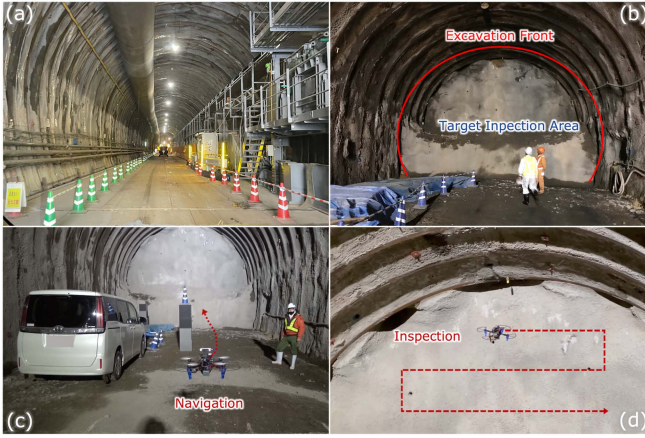
Fig. 1. Illustration of UAV navigating and inspecting the excavation front in the tunnel environment. (a) The tunnel under construction. (b) The target inspection area (the excavation front). (c) The robot navigates toward the inspection target and avoids obstacles. (d) The robot inspects the target area.

optimize the trajectory for avoiding dynamic obstacles. The example tunnel environment and the autonomous robot using the proposed method are shown in Fig. 1. The main contributions and novelties of this work are:

- *Autonomous UAV system for inspecting unknown and dynamic tunnels:* We design a novel autonomous UAV system tailored to tunnel inspection. The system adopts our hierarchical planning framework and the dynamic-obstacle-aware map representation to achieve safe inspection in unknown and dynamic tunnel environments.
- *Lightweight U-depth dynamic obstacle detector:* The paper introduces our lightweight U-depth map-based detector for dynamic obstacle detection. The detected bounding boxes of dynamic obstacles can be merged with the traditional occupancy map to form the dynamic map for safe static and dynamic obstacle avoidance.
- *Gradient-based dynamic obstacle avoidance:* We propose the circle-based guide-point method and the receding horizon distance field to efficiently compute the trajectory optimization gradients for obstacle avoidance.
- *Large-scale tunnel inspection experiments:* The entire system was verified using our customized quadcopter. The experiment was conducted in a tunnel under construction with a cross-section size of 8 meters in height and 8 meters in width in Japan. The experiment demonstrates that our system can safely inspect the tunnel and output 3D reconstruction results for tunnel construction.

II. RELATED WORK

This section first discusses the recent trends and approaches in construction site inspection by autonomous UAVs. Then, relevant works on the key challenges of tunnel inspection (i.e., exploration and dynamic obstacle avoidance) are reviewed.

There are mainly two categories of construction site and building inspection methods: model-based and non-model-based methods. For the model-based methods, the inspection target model is usually available, and the planner generates a set of optimal viewpoints based on the provided model. In [9], the

target bridge is first partitioned into surfaces with inspection nodes, and their GTSP solver is then applied to find optimal paths for inspection. Similarly, some works use the BIM model to find viewpoints of interest (VPI) and solve the path-planning problem using the TSP-based method [10], [11]. However, the target model can be unavailable for tunnel inspection, so the robot can only rely on the onboard sensors. In this way, the reactive methods are proposed for unknown tunnel navigation using the lidar points measurement [12], [13]. Their methods can navigate tunnels of arbitrary shapes but do not consider obstacle avoidance. Bendris et al. [5] utilize the sampling-based method to generate viewpoints for unknown exploration and inspection. Their method can successfully avoid static obstacles but might not be safe for dynamic obstacles due to the long replanning time. Besides, their random sampling strategy in the explored area can lead to zigzag and over-conservative paths for navigation. In [14], it proposes a 3D reconstruction method for UAV tunnel inspection without the path-planning strategy.

The unknown exploration problem can be viewed as determining a series of informative viewpoints [15]. Yamauchi [16] first uses the frontier exploration approach, allowing robots to visit the map boundary to gain environment information. Later in [17], it extends the frontier exploration to high-speed UAVs. Some approach [18] applies the information-theoretic method to evaluate the information gains of viewpoints. Considering the limited computation power of lightweight UAVs, the sampling-based methods [1], [2], [3], [4] have been preferred in recent years. In [1], their RH-NBV planner grows an RRT with the information gains stored in each node. The robot will then follow the highest gain branch in a receding horizon manner. Selin et al. [2] combine the RH-NBV with frontier exploration, further improving the exploration efficiency. To save and reuse the computation in each planning iteration, Schmid et al. [3] adopt the RRT* algorithm with the rewiring to incrementally build the tree. With a similar incremental sampling idea in [4], it proposes a PRM-based method for exploration and obstacle avoidance in dynamic environments.

Dynamic obstacle avoidance problem still remains open in recent years. In the reactive-based methods, the robots directly generate control velocities to avoid obstacles. Khatib [19] constructs the artificial potential field to find the velocity for obstacle avoidance and navigation, and Berg et al. [20] use linear programming to optimize velocities based on Velocity Obstacle [21]. These methods require less computation than the trajectory-based methods but might lead to more myopic performance. The trajectory-based methods are more prevalent in UAV planning in recent years. Some [7], [22], [23], [24] use the model predictive control scheme to generate collision-free trajectories based on the kinematic constraints. In [8], it utilizes the B-spline optimization to generate collision-free trajectory with vision aided, and Chen et al. [25] evaluate trajectory risks using their dual-structure particle map.

III. PROBLEM DESCRIPTION

In an unknown tunnel space, $\mathcal{V}_t \in \mathbb{R}^3$, with a straight tunnel centerline $\mathcal{C}$ of a finite length, there exists an excavation front (i.e., the target wall for inspection) at the end of the tunnel.
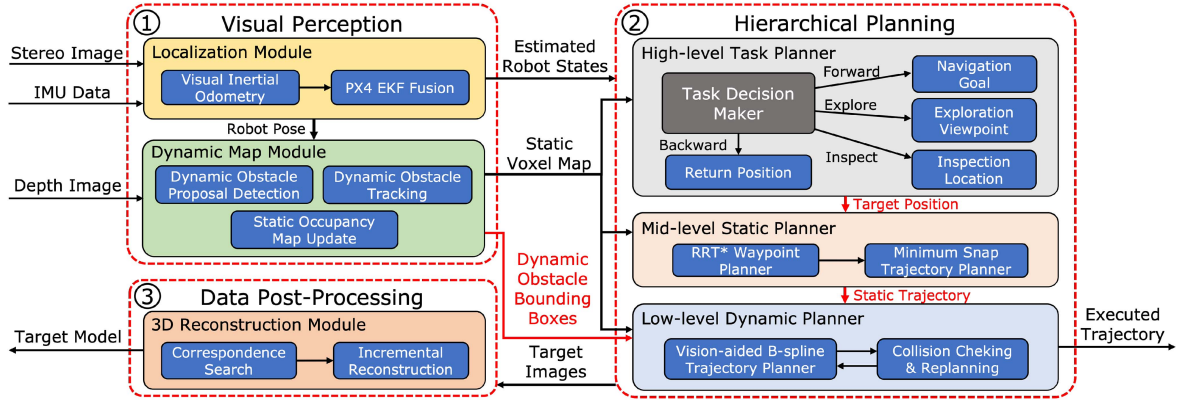
Fig. 2. System framework for autonomous inspection. Our proposed framework contains three parts: Visual perception, hierarchical planning, and data post-processing. In the visual perception step, the localization module applies the visual-inertial odometry with EKF fusion for state estimation. The dynamic map module builds the static voxel map and tracks dynamic obstacles based on depth images. In the hierarchical planning section, the high-level and mid-level planners use the static voxel map to generate the static trajectory. Then, the low-level planner uses the dynamic obstacle information to optimize the output trajectory for execution. The final data post-processing step takes the images collected from the inspection stage to reconstruct the target model for analysis.

Inside the tunnel space $\mathcal{V}_t$, there are different sizes of static obstacles $\mathcal{O}_{\text{static}}$ and dynamic obstacles $\mathcal{O}_{\text{dynamic}}$. A UAV with an onboard depth camera is deployed for the inspection task. Without a prior map $\mathcal{M}$, the robot needs to first navigate toward the excavation front area from an arbitrary position in the space $\mathcal{V}_t$, then generate an inspection path to collect RGB images of the inspection target, and finally return to the start location. During the forward navigation and returning period, the robot should avoid all static obstacles $\mathcal{O}_{\text{static}}^{\text{all}}$ and dynamic obstacles $\mathcal{O}_{\text{dynamic}}^{\text{sensor}}$ in its sensor range. The final output of the entire system should be the 3D shape of the inspection target reconstructed using the collected RGB images.

IV. PROPOSED METHOD

The proposed inspection framework has three main components shown in Fig. 2: visual perception, hierarchical planning, and data post-processing. The visual perception step processes the sensor measurements from the onboard depth camera and the inertial measurement unit (IMU). The localization module runs the visual-inertial odometry (VIO) algorithm with the EKF fusion to get robot state estimation. Besides, the dynamic map module utilizes depth images to track dynamic obstacles and update the occupancy information for static obstacles using the voxel map, which will be further discussed in Section IV-B. After the perception step, the hierarchical planning section generates collision-free trajectories for the robot to achieve the entire inspection task. Section IV-A will introduce the logic of our hierarchical planning for the tunnel inspection and the task decision maker in the high-level planner. Then, the obstacle avoidance based on the mid-level trajectory planner and low-level dynamic planner will be covered in Section IV-C. After finishing the inspection task, the data post-processing step, mentioned in Section IV-D, takes the collected target images to perform 3D reconstruction to obtain the target model.

A. Hierarchical Planning and High-Level Task Planner

Since our inspection problem consists of multiple complicated procedures, applying only one planner cannot efficiently accomplish the entire task. There are mainly three stages of the inspection: (a) approaching the inspection target (i.e., the end of the tunnel), (b) collecting target images, and (c) returning to the start location. Based on the inspection stages, we decompose the problem into the following abstract tasks:

$$\mathcal{S}_{\text{T}} = \{\text{Forward}, \text{Explore}, \text{Inspect}, \text{Return}\}, \qquad (1)$$

where the Forward task aims at approaching the inspection target, the Explore task helps the robot gain local map information for navigation, the Inspect task mode generates the path for collecting target images, and the Return task mode navigates the robot back to the starting location. During the inspection process, the robot constantly alternates the task mode using the proposed task planning algorithm (Algorithm 1). For each abstract task, the task planner generates the corresponding goal positions and passes them to the lower-level planners for path planning and trajectory optimization.

In the beginning stage of task planning (Algorithm 1), the task planner sets the robot to the Forward task mode as the robot needs first to approach the tunnel end (Line 1). The task planner runs at a certain replanning frequency to select the current task mode for the robot. Before the robot arrives at the inspection location, the **Forward Mode** (Lines 9-13) lets the robot generate a forward goal with a distance l from the current robot position for navigation. Since, at this stage, the robot does not have a complete environment map and can only rely on the partially built from its flight, it will first try using the partial map to perform local obstacle avoidance to achieve the forward goal (Line 11). Suppose the lower-level planner fails to find a collision-free trajectory due to the lack of environmental knowledge. In that case, the task planner will switch the current task to the **Explore Mode** to increase the local map information (Lines 12-13). In the Explore Mode, the planner first samples to get the best viewpoints with the highest sensor information gain in the current map then uses the lower-level planner to generate a feasible trajectory for exploration, and finally switches back to the previous task mode (Lines 14-17). For the information gain evaluation, refers to [1], [2], [3], [4] for further details. At the start of each replanning iteration, the algorithm checks whether

Algorithm 1: High-level Task Planning Algorithm.

1: $\mathcal{T}_{\text{curr}} \leftarrow$ Forward Mode;   ▷ initial task
2: $\mathcal{C}_t \leftarrow false$;   ▷ termination condition
3: **while** not $\mathcal{C}_t$ **do**
4:    $\mathcal{C}_t \leftarrow$ **isInspectionComplete**();
5:    $\mathcal{I}_{\text{reach}} \leftarrow$ **reachInspectionTarget**();
6:    **if** $\mathcal{I}_{\text{reach}}$ **then**
7:       $\mathcal{T}_{\text{curr}} \leftarrow$ Inspect Mode;
8:    **switch** $\mathcal{T}_{\text{curr}}$ **do**
9:       **case** Forward Mode **do**
10:          $\mathcal{P}_{\text{goal}} \leftarrow$ **getForwardGoal**();
11:          σ_{traj}, success $\leftarrow$ **lowerLevelPlanner**($\mathcal{P}_{\text{goal}}$);
12:          **if not** success **then**
13:             $\mathcal{T}_{\text{curr}} \leftarrow$ Explore Mode;
14:       **case** Explore Mode **do**
15:          $\mathcal{P}_{\text{goal}} \leftarrow$ **getBestViewpoint**();
16:          $\sigma_{\text{traj}} \leftarrow$ **lowerLevelPlanner**($\mathcal{P}_{\text{goal}}$);
17:          $\mathcal{T}_{\text{curr}} \leftarrow$ **getPreviousTaskMode**();
18:       **case** Inspect Mode **do**
19:          σ_{traj}, success $\leftarrow$ **getInspectionPath**();
20:          **if not** success
21:             $\mathcal{T}_{\text{curr}} \leftarrow$ Explore Mode;
22:          **else**
23:             **executeInspectionPath**();
24:             $\mathcal{T}_{\text{curr}} \leftarrow$ Return Mode;
25:       **case** Return Mode **do**
26:          $\mathcal{P}_{\text{goal}} \leftarrow$ **getReturnGoal**();
27:          $\sigma_{\text{traj}} \leftarrow$ **lowerLevelPlanner**($\mathcal{P}_{\text{goal}}$);

(a) RGB Camera View.   (b) Depth Detection.   (c) U-depth Map with the Detetion Results.
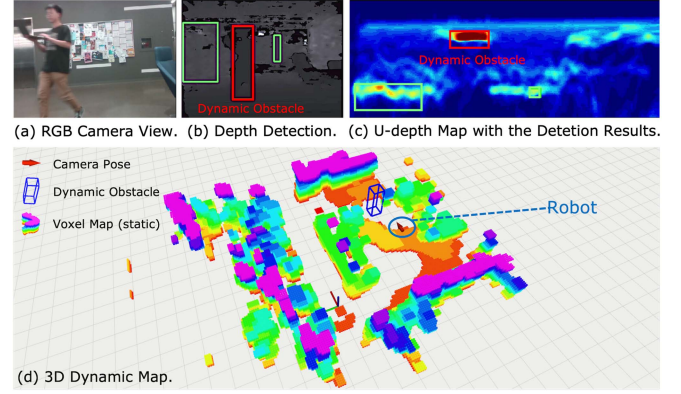
(d) 3D Dynamic Map.

Fig. 3. Illustration of the proposed 3D dynamic map. (a) A person walks in front of the robot in the RGB camera view. (b) The person is detected as a dynamic obstacle in the depth image. (c) The detection results in the U-depth map for obstacle widths and thicknesses. (d) The 3D dynamic map shows the dynamic obstacle as a bounding box and static obstacles as the voxel map.

the robot has reached the inspection target (Lines 5-7). If the robot detects the inspection target wall, the planner will enter the **Inspect Mode** and generate a zigzag path for collecting target images. However, when the built map around the target is not detailed enough for the inspection path generation, the planner will switch to the Explore Mode again to increase the explored map range (Lines 18-21). After finishing collecting images, the planner will enter the **Return Mode** and navigate back to the start position (Lines 25-27). Note that in the returning step, the robot has already had a sufficient informative map for static obstacles, incrementally built from the forward and explore step, to generate a global trajectory to the origin directly.

B. Perception and 3D Dynamic Mapping

This section introduces our proposed 3D dynamic map for navigating dynamic environments, as shown in Fig. 3(d). Our dynamic map adopts a hybrid method to represent environments by using the occupancy voxels for static obstacles and the bounding boxes for dynamic obstacles. For static obstacles, we predefine a static voxel map size (i.e., maximum voxel numbers) based on the environment and store the occupancy information of each voxel in an array with the preserved length. This allows our planners to access the occupancy information with $\mathcal{O}(1)$ time complexity. For the occupancy information update of each voxel, as most static occupancy mapping algorithm does, we

apply the classic Bayesian filter with the Markov assumption:

$$l_t(x) = \log \frac{p(x|z_t)}{p(\bar{x}|z_t)} + \log \frac{p(\bar{x})}{p(x)} + l_{t-1}(x), \quad (2)$$

where $l_t(x)$ is the log odds for the voxel being occupied. By applying (2), we can update the occupancy information (i.e., log odds) for each voxel by recursively adding the inverse sensor model $\log \frac{p(x|z_t)}{p(\bar{x}|z_t)}$ with the predefined prior $\log \frac{p(\bar{x})}{p(x)}$. Besides, since dynamic obstacles can also be mapped into the static voxel map, which can lead to noisy voxels, we iterate through each detected dynamic obstacle bounding box and set all voxels inside the dynamic regions to be free.

The dynamic obstacles are detected and tracked using the depth image and represented by axis-aligned 3D bounding boxes. There are mainly three steps in the proposed method: region proposal detection, map-depth fusion and dynamic obstacle filtering. In the region proposal detection step, we use the method mentioned in [6] to generate the U-depth map, as shown in Fig. 3(c), by constructing a histogram of the depth values using the depth image. The vertical axis from top to bottom of the U-depth map represents the depth range of the user-defined bin width. Intuitively, the U-depth map can be viewed as a top-down view image. Inspired by [6], [7], we apply the line grouping method to detect the obstacle regions in the U-depth map. With these detection results, we can obtain the widths and thicknesses of obstacles and then further find the corresponding heights in the original depth image as shown in Fig. 3(b). After this step, we can get the "region proposal bounding boxes" for dynamic obstacles by applying coordinate transformation into the map frame. Since the region proposals are only the rough detection results, our second step, map-depth fusion, inflates those region proposals locally with a ratio λ and then searches occupied voxels from the static voxel map to get the refined bounding boxes of obstacles. With the refined bounding boxes, the dynamic obstacle filtering method is applied to identify and track dynamic obstacles. First, we utilize the Kalman filter to track and compute the velocity of each obstacle bounding box

with the linear propagation model:

$$\mathbf{p}_o^{k+1} = \mathbf{p}_o^k + \mathbf{v}_o^k(t_{k+1} - t_k), \quad \mathbf{v}_o^k = \frac{\mathbf{p}_o^k - \mathbf{p}_o^{k-1}}{t_k - t_{k-1}}, \quad (3)$$

where $\mathbf{p}_o^{k+1}$ is the predicted obstacle position in the next time step and $\mathbf{v}_o^k$ is the previously estimated velocity. Then, we identify those bounding boxes with velocities greater than the threshold V_{th} as the dynamic obstacles. Finally, we remove the bounding boxes with jerky motions using the obstacles' historical velocities, considering the detection noises that make static obstacles shake back and forth slightly. Because we use velocity as the criterion to identify dynamic obstacles, if a static obstacle begins to move, it will be detected as a dynamic obstacle represented by a bounding box. Consequently, the occupied voxels in the map are freed, and vice versa.

C. Navigation and Obstacle Avoidance

When a goal position is determined by the high-level task planner, the mid-level static planner first finds a smooth trajectory considering static obstacles. Then, using this static trajectory, the low-level dynamic planner optimizes a collision-free trajectory based on static and dynamic obstacles at a certain replanning frequency. For the mid-level static planner, we apply the RRT* planner to find the waypoint path and use the minimum snap-based polynomial optimization with corridor constraints [26], [27] for trajectory generation. To achieve fast replanning for dynamic obstacle avoidance, the low-level planner adopts our gradient-based trajectory optimization. The B-spline trajectory with order k over a time knot vector can be parameterized as a series of control points:

$$\hat{\mathcal{S}} = \{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \dots, \mathbf{P}_{N-1}, \mathbf{P}_N\}, \quad \mathbf{P}_i \in \mathcal{R}^3, \quad (4)$$

where the optimization variable set $\mathcal{S}$ contains the $N - 2(k - 1)$ intermediate control points $\mathbf{P}_i$. With the trajectory optimization variables, we can write the objective function as follows:

$$\begin{aligned} C_{total}(\mathcal{S}) = {} & \alpha_{control} \cdot C_{control} + \alpha_{smooth} \cdot C_{smooth} \\ & + \alpha_{static} \cdot C_{static} + \alpha_{dynamic} \cdot C_{dynamic}, \quad (5) \end{aligned}$$

and the weighted sum has four costs to minimize: the control limit cost, the smoothness cost, the static collision cost, and the dynamic collision cost. The control limit cost ensures the trajectory has feasible velocities and accelerations. The control points for velocity $\mathbf{V}_i$ and acceleration $\mathbf{A}_i$ are computed by:

$$\mathbf{V}_i = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{\delta t}, \quad \mathbf{A}_i = \frac{\mathbf{V}_{i+1} - \mathbf{V}_i}{\delta t}, \quad (6)$$

where δt is the time step. We use the L2 norm to penalize the infeasible velocities and accelerations:

$$C_{control} = \sum_i \frac{\|\mathbf{V}_i - \mathbf{v}_{max}\|_2^2}{\lambda_{vel}} + \frac{\|\mathbf{A}_i - \mathbf{a}_{max}\|_2^2}{\lambda_{acc}}, \quad (7)$$

in which $\mathbf{v}_{max}$ and $\mathbf{a}_{max}$ are the maximum velocity and acceleration limits. The λ terms are the unit normalization factor. Note that the control limit costs are zero for velocities and acceleration that are less than the limits. The smoothness cost tries to reduce

(a) Circle-based guide point assignment for the static collision cost.

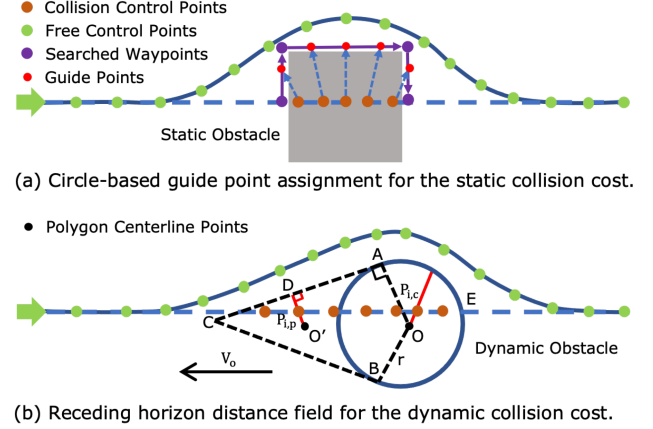(b) Receding horizon distance field for the dynamic collision cost.

Fig. 4. Illustration of the collision cost in our B-spline optimization. (a) The static collision cost is calculated using the proposed circle-based guide points (red dots). (b) The dynamic collision cost is obtained by the receding horizon distance field, which considers the future predictions of the obstacle positions.

the jerk (i.e., the third derivative to position) of the trajectory using the following equations:

$$C_{smooth} = \sum_i \|\mathbf{J}_i\|_2^2, \quad \mathbf{J}_i = \frac{\mathbf{A}_{i+1} - \mathbf{A}_i}{\delta t}. \quad (8)$$

The static collision cost is computed based on the proposed circle-based guide-point method shown in Fig. 4(a). The initial trajectory is shown as the blue dot line with the brown collision control points. To calculate the costs for those collision control points, we first search a collision-free path (purple dots and lines in Fig. 4(a)) using A* or Dijkstra to bypass the static obstacle. If there are N collision control points, we cast a ray for the collision control point of sequence order n with the angle $\frac{180}{n+1}$ degree. Note that the angle is between the casting ray (dot blue arrow) and the line connecting the first and last collision control points. The guide points $\mathbf{P}_{guide}$ are the intersection points of the casting ray with the searched path. The algorithm is circle-based because the direction angles sweep a semi-circle. With the associated guide points for each collision control point, we design the total static collision cost based on experiments as a clipped cubic penalty function:

$$C_{static} = \sum_i \left(\mathbf{max}\left(d_{safe} - \mathbf{signDist}(\mathbf{P}_i, \mathbf{P}_{guide}^i), 0 \right) \right)^3, \quad (9)$$

where d_{safe} is the user-defined safe distance, and the signed distance function defines the positive and negative distance as the control point outside and inside the obstacle. Intuitively, we penalize the control points with small or negative distances to obstacles, and the static collision costs are zero for control points with a distance greater than the safe distance.

Since the dynamic obstacles are moving, it is unreliable to only use the current detected information for cost computation. So, we propose the receding horizon distance field to estimate the dynamic collision cost with future predictions shown in Fig. 4(b). In this figure, the dynamic obstacle with left moving velocity V_o is represented as the blue circle with the center O and the radius r. We apply linear prediction to get the obstacle's

future position C with the prediction horizon k time step. Since the reliability of future prediction decreases with the increasing prediction time, we linearly decrease the obstacle size to zero at the final predicted position C in the receding horizon manner. So, we can obtain the collision region as the combination of a polygon region AOBC and a circular region enclosed by the arc AEB, line AO, and line BO. When the control point $\mathbf{P}_{i,p}$ is inside the polygon region, we draw a red line through the control point $\mathbf{P}_{i,p}$ perpendicular to the line AC intersecting at point D. The distance d_i to the safe area (outside the collision region) can be computed as:

$$\Delta d_i = \|\mathbf{D} - \mathbf{O}'\|_2 - \|\mathbf{P}_{i,p} - \mathbf{O}'\|_2. \tag{10}$$

On the other hand, when the control point $\mathbf{P}_{i,c}$ is inside the circular region, the distance d_i to the safe area is:

$$\Delta d_i = r - \|\mathbf{P}_{i,c} - \mathbf{O}_0\|_2. \tag{11}$$

For the control points $\mathbf{P}_{i,out}$ that are outside both polygon and circular regions, we set the distance d_i to the safe area to zero. So, with the distance to the safe area, we can use the following equation to compute the final dynamic collision cost:

$$C_{dynamic} = \sum_i \Big(\mathbf{max}(\Delta d_i, 0)\Big)^3. \tag{12}$$

For both static and dynamic collision costs, the gradients can be computed using the chain rule with (9) and (12).

D. Inspection and 3D Reconstruction

After finishing the entire inspection task, the data post-processing module applies the Structure-from-Motion (SfM) to reconstruct the 3D shape of the inspection target from the collected target images. When the robot has reached the inspection target, it first explores the local area until having enough map information about the target. Then, the robot generates a zigzag pattern path with its camera facing toward the target wall and collects color images during the flight. After finishing the zigzag path, the robot will turn 45 degrees towards the corner of the target wall, following a rectangular shape path to collect images of the wall fringe. We adopted the zigzag pattern inspection path because it is a simple yet efficient way to cover the target surface, making it suitable for industrial applications. However, there are more intelligent ways to find the best viewpoints considering execution time and reconstruction quality, which are outside the scope of this paper. Our SfM pipeline for reconstruction is based on COLMAP [28]. The algorithm first extracts the features of each image using a numerical descriptor. Since our input images are from the streaming of an RGB camera, the second step utilizes sequential matching to find the correspondence in different images. Finally, from an initial corresponding image pair, the algorithm incrementally reconstructs the 3D shape of the inspection target by triangulating new points.
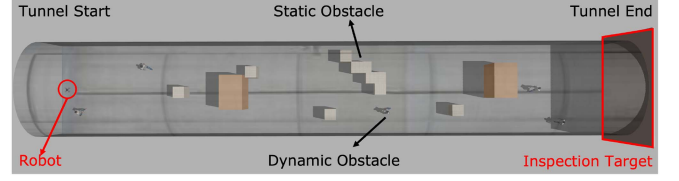

Fig. 5. Illustration of an example simulation tunnel environment in Gazebo. In the forward task, the robot needs to navigate from the tunnel start (left side) to the tunnel end (right side) and avoid static and dynamic obstacles.

V. RESULT AND DISCUSSION

A. Implementation Details

We conduct simulation experiments and physical flight tests in dynamic tunnel environments to evaluate the proposed method's performance. The simulation environments are based on ROS and Gazebo. For the physical experiments, we visited a tunnel under construction in Japan and applied our customized quad-copter (Fig. 6 left) to test the proposed framework. The quad-copter is equipped with a RealSense D435i camera, a PX4-based flight controller, and an NVIDIA Xavier NX onboard computer. We adopt the visual-inertial odometry (VIO) algorithm for robot state estimation. All of the perception and planning computations are performed within the onboard computer. The color images are collected during the inspection stage with the RealSense D435i camera, and the data post-processing for 3D reconstruction is completed using the desktop with an NVIDIA RTX 3080 GPU.

B. Evaluation of Navigation and Obstacle Avoidance

The navigation and obstacle avoidance in the forward task (i.e., approaching the tunnel end) is the most challenging and time-consuming part of the entire inspection process since the environment is cluttered and unknown. So, to evaluate the performance of forward navigation and obstacle avoidance, we prepared 5 simulation environments containing different static and dynamic obstacles, with one example environment shown in Fig. 5. For benchmarking, we select the sampling-based planning methods (SBP) [1], [5] and the dynamic exploration planning (DEP) method [4] with modifications to the tunnel environments. Besides, we also include our method without using the dynamic map (mentioned in Section IV-B) to compare the obstacle avoidance performance. In each experiment, we let the robot navigate from the start of the tunnel to the end of the tunnel. We run 10 experiments in each environment of different obstacles and record the average navigation time, the average replanning time for dynamic obstacle avoidance, and the collision rate over all experiments. Note that we set the navigation time and replanning time of the sampling-based planning methods (SBP) [1], [5] to 100% for comparison. The collision rate is calculated by the number of experiments with collisions divided by the total number of experiments.

From the results in Table I, one can see that our method has the second least navigation time, which is 81.69% of the sampling-based planning (SBP) method, and takes almost the same amount of time as its non-dynamic-map version.

TABLE I
COMPARISON OF NAVIGATION AND OBSTACLE AVOIDANCE

Methods	Nav. Time	Replan. Time	Collision Rate
SBP [1][5]	$100 \pm 0\%$	100%	30.00%
DEP [4]	$92.80 \pm 3.01\%$	54.30%	24.00%
Ours w/o DM	$\mathbf{81.06 \pm 4.40}\%$	1.20%	32.00%
Ours	$81.69 \pm 3.66\%$	$\mathbf{1.16}\%$	$\mathbf{0.00}\%$

TABLE II
MEASUREMENT OF THE 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

The dynamic exploration planning (DEP) method uses less time than the sampling-based method and longer time than our method. From our observations, both the SBP and the DEP generate their trajectories inside the explored regions, which is over-conservative, leading to more stop-and-go behavior. On the contrary, since our planner adopts a hierarchical scheme, the task planner first tries using the more aggressive local planner for obstacle avoidance by planning in the unknown regions and only applies the conservative exploration planner when the local planning fails. This task-switching behavior hugely reduces the navigation time. For the replanning time, our method takes only 1.16% of the time compared to the SBP and significantly less than the DEP. This huge difference in the replanning speed is mainly due to our computationally lightweight gradient-based trajectory optimization and the long computation time in the information gain evaluation of the SBP and the DEP. For the collision rate, it is shown that our method has no collision among all experiment runs, and both the SBP and our method without the dynamic map have a high collision rate (around 30%). The DEP has a lower collision rate than the SBP since it utilizes an incremental roadmap for faster dynamic obstacle avoidance but still has more collisions than our method. Comparing our method with and without the dynamic map shows that the dynamic map version has a much lower collision rate by using dynamic obstacle information.

C. Evaluation of Dynamic Obstacle Tracking

We measure the average tracking errors in positions, velocities, and obstacle sizes shown in Table II to evaluate the dynamic obstacle detection and tracking performance. The ground truth states of the obstacles can be easily obtained in the simulation experiments, and we apply the OptiTrack motion capture system in the physical tests to obtain the ground truth states. We let two persons walk within the motion capture area, compare the tracking results from the robot and the motion capture system, and use the average value differences as tracking errors. From Table II, one can see that the position errors are 0.09 m and 0.19 m in simulation and physical tests, respectively. The position errors in the physical tests are larger than in simulation tests due to the image's noises from the depth camera. Similarly, the camera noises also make the velocity errors in physical tests greater than the simulations'. The size errors are similar in both simulation
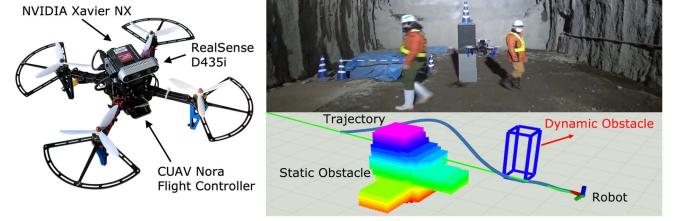

Fig. 6. Inspection experiments with the customized quadcopter (left) in a tunnel under construction in Japan (top right). The bottom right visualizes the static and dynamic obstacles and the robot planned trajectory.
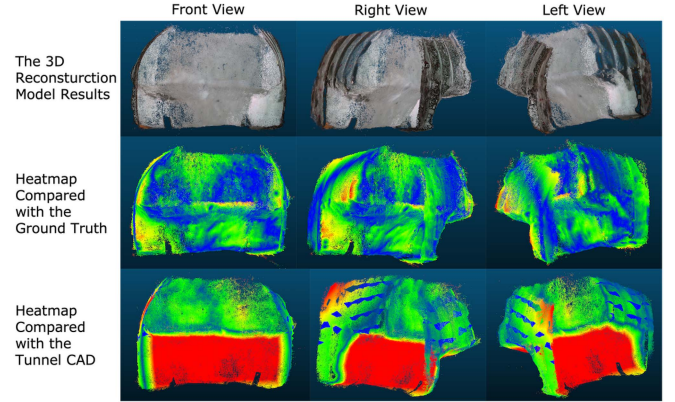

Fig. 7. 3D reconstruction results of the excavation front of the tunnel under construction in Japan. The first row shows the 3D reconstruction model from different views. The second row visualizes the error heatmap obtained from the comparison of the laser-scanned ground truth. The third row presents the heatmap comparison of the reconstruction model with the CAD model.

and physical tests. In the experiments, to account for the tracking errors in the positions, velocities, and sizes, we increase the safety distance to obstacles by a self-defined size r, and our experiment results prove that our dynamic obstacle tracking system can let successfully avoid moving obstacles.

D. Physical Flight Tests

To evaluate and verify the proposed framework, we ran flight tests in a tunnel under construction in Japan, shown in Figs. 1 and 6. In each flight test, the robot starts at 20 meters in front of the tunnel excavation front and navigates toward the inspection area. Note that there are static and dynamic obstacles (i.e., walking workers) on the robot's way to its target location shown at the top of Fig. 6. The corresponding Rviz visualization is shown at the bottom of Fig. 6, and one can see that the robot can generate a collision-free trajectory for navigation. After reaching the inspection area, the robot will follow the zigzag path to inspect the tunnel excavation front shown in Fig. 1(d) and collect RGB images for further 3D reconstruction. During the navigation period, the robot's velocity is maintained at 1.0 m/s. The results show that our framework can complete the entire inspection task autonomously.

E. Evaluation of 3D Reconstruction

The final output of our framework is the 3D shape of the tunnel excavation front shown in Fig. 7. To obtain the results, we run

the SfM-based reconstruction mentioned in Section IV-D with 294 color images of 640 × 480 resolution. The total processing time is 30 minutes using an NVIDIA RTX 3080 GPU, and the minimum number of images required for this experiment is 60 images which take only 5 minutes for reconstruction. In Fig. 7, the first row shows the reconstruction results from different views, and the second row visualizes the error heatmap from the comparison with the ground truth model. Note that we use the Topcon laser scanner to obtain the ground truth model of the inspection target. The red and blue portion of the heatmap represents the high and low reconstruction error values. The reconstruction model has an average error of 5.38 cm, which is 0.67% of the tunnel's width and height, with a standard deviation of 7.96 cm. This reconstruction error falls within an acceptable range to perform construction operations based on our industrial partners' suggestions. The third row shows the heatmap comparison with the tunnel CAD model, the designed shape for the tunnel. From the heatmap, the workers can identify the yellow and red regions as the locations for concrete spraying and excavation.

VI. Conclusion and Future Work

This paper presents a vision-based autonomous UAV inspection framework for tunnel environments. The proposed framework adopts a hierarchical planning scheme to solve the complicated inspection problem using different planning layers. Our depth-based 3D dynamic map can represent static obstacles and track dynamic obstacles simultaneously. The experiment results prove that our framework can make the quadcopter safely navigate toward the inspection target to perform the inspection and return to the origin. The final 3D reconstruction results obtained from our SfM-based data post-processing pipeline have a low error compared to the ground truth. For future work, we want to apply learning-based methods to classify dynamic obstacles. Moreover, a more optimal inspection path than the current zigzag path could be explored using a lightweight online reconstruction method, which may lead to improved reconstruction quality.

References

[1] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3D exploration," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 1462–1468.

[2] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-D environments," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1699–1706, Apr. 2019.

[3] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart, and J. Nieto, "An efficient sampling-based method for online informative path planning in unknown environments," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 1500–1507, Apr. 2020.

[4] Z. Xu, D. Deng, and K. Shimada, "Autonomous UAV exploration of dynamic environments via incremental sampling and probabilistic roadmap," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 2729–2736, Apr. 2021.

[5] B. Bendris and J. C. Becerra, "Design and experimental evaluation of an aerial solution for visual inspection of tunnel-like infrastructures," *Remote Sens.*, vol. 14, no. 1, 2022, Art. no. 195.

[6] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for MAV flight," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 50–56.

[7] J. Lin, H. Zhu, and J. Alonso-Mora, "Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 2682–2688 .

[8] Z. Xu, Y. Xiu, X. Zhan, B. Chen, and K. Shimada, "Vision-aided UAV navigation and dynamic obstacle avoidance using gradient-based B-spline trajectory optimization," 2022, *arXiv:2209.07003*.

[9] P. Shanthakumar et al., "View planning and navigation algorithms for autonomous bridge inspection with UAVs," in *Proc. Int. Symp. Exp. Robot.*. Springer, 2020, pp. 201–210.

[10] N. Bolourian and A. Hammad, "LiDAR-equipped UAV path planning considering potential locations of defects for bridge inspection," *Automat. Construction*, vol. 117, 2020, Art. no. 103250.

[11] Y. Tan, S. Li, H. Liu, P. Chen, and Z. Zhou, "Automatic inspection data collection of building surface based on BIM and UAV," *Automat. Construction*, vol. 131, 2021, Art. no. 103881.

[12] T. Elmokadem, "A 3D reactive navigation method for UAVs in unknown tunnel-like environments," in *Proc. IEEE Australian New Zealand Control Conf.*, 2020, pp. 119–124.

[13] T. Elmokadem and A. V. Savkin, "A method for autonomous collision-free navigation of a quadrotor UAV in unknown tunnel-like environments," *Robotica*, vol. 40, no. 4, pp. 835–861, 2022.

[14] R. S. Pahwa, K. Y. Chan, J. Bai, V. B. Saputra, M. N. Do, and S. Foong, "Dense 3D reconstruction for visual tunnel inspection using unmanned aerial vehicle," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 7025–7032.

[15] C. Connolly, "The determination of next best views," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 2, 1985, pp. 432–435.

[16] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Automat. CIRA'97.'Towards New Comput. Princ. Robot. Automat.*, 1997, pp. 146–151.

[17] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 2135–2142.

[18] B. Charrow et al., "Information-theoretic planning with trajectory optimization for dense 3D mapping," in *Proc. Robot.: Sci. Syst.*, vol. 11, 2015, pp. 3–12.

[19] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*. Berlin, Germany: Springer, 1986, pp. 396–404.

[20] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal *n*-body collision avoidance," in *Robotics Research*. Berlin, Germany: Springer, 2011, pp. 3–19.

[21] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, 1998.

[22] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Trans. Robot.*, vol. 27, no. 6, pp. 1080–1094, Dec. 2011.

[23] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for MAVs in dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 776–783, Apr. 2019.

[24] Z. Xu, D. Deng, Y. Dong, and K. Shimada, "DPMPC-planner: A real-time UAV trajectory planning framework for complex static environments with dynamic obstacles," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 250–256.

[25] G. Chen, P. Peng, P. Zhang, and W. Dong, "Risk-aware trajectory sampling for quadrotor obstacle avoidance in dynamic environments," 2022, *arXiv:2201.06645*.

[26] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.

[27] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Berlin, Germany: Springer, 2016, pp. 649–666.

[28] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4104–4113.