2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Efficient Planning for High-Speed MAV Flight in Unknown Environments Using Online Sparse Topological Graphs

Matthew Collins and Nathan Michael

***Abstract*— Safe high-speed autonomous navigation for MAVs in unknown environments requires fast planning to enable the robot to adapt and react quickly to incoming information about obstacles within the world. Furthermore, when operating in environments not known *a priori*, the robot may make decisions that lead to dead ends, necessitating global replanning through a map of the environment outside of a local planning grid. This work proposes a computationally-efficient planning architecture for safe high-speed operation in unknown environments that incorporates a notion of longer-term memory into the planner enabling the robot to accurately plan to locations no longer contained within a local map. A motion primitive-based local receding horizon planner that uses a probabilistic collision avoidance methodology enables the robot to generate safe plans at fast replan rates. To provide global guidance, a memory-efficient sparse topological graph is created online from a time history of the robot's path and a geometric notion of visibility within the environment to search for alternate pathways towards the desired goal if a dead end is encountered. The safety and performance of the proposed planning system is evaluated at speeds up to 10m/s, and the approach is tested in a set of large-scale, complex simulation environments containing dead ends. These scenarios lead to failure cases for competing methods; however, the proposed approach enables the robot to safely reroute and reach the desired goal.**

I. INTRODUCTION

Autonomous navigation in unknown environments presents a challenging planning problem due to the safety guarantees that must be met and the number of subsystems that are affected, e.g., control, mapping, and state estimation. Micro aerial vehicle (MAV) deployment in applications including search and rescue and reconnaissance place additional demands on safety, planner reaction time, and robustness due to their time critical nature. The core problem from a planning perspective in these scenarios is that the robot must be capable of online navigation at high-speeds through unknown environments. Fast flight with limited sensor field of view (FOV) requires the system to process sensor data quickly, making real-time planning and mapping paramount for ensuring performance and safety. The challenge is that MAVs are constrained in terms of size, weight, power, and compute. As a result, online planning must be both **computationally-** and **memory-efficient**.

Current MAV planning methodologies have enabled high-rate planning by using motion primitive libraries [2], [16], [9], search-based methods [6], [7], [11], or fast optimization-based techniques [19], [20], [13], [14], [8], [12], [4], but these planners can be myopic, limiting their applicability in large-scale, unknown environments. Typically, the robot builds a global map of the environment to be used for planning; however, when the scale of the environment is not known *a priori*, sliding maps are often employed to cap memory usage. One drawback of a sliding map is that it provides no information about the world outside its boundaries or the robot's travel history. This becomes an issue if the local sliding map is not large enough to cover the entire area of interest, leading to scenarios where the robot gets stuck and can no longer plan to the desired location or retrace its steps.

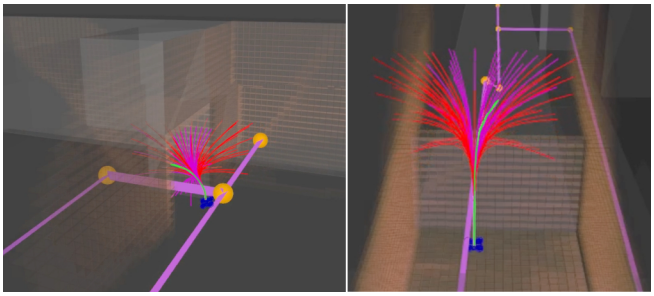


Fig. 1: MAV using the sparse topological graph (yellow nodes and purple edges) created online to plan through the environment, e.g., through a doorway (left) and over a wall (right). The selected motion primitive from the library is shown in green, infeasible primitives are in red, and the local map is shown in orange.

These challenges and the operational requirements necessitate a planning methodology that is both computationally- and memory-efficient, while still generating smooth, safe trajectories at high-speeds. Moreover, the planner cannot be overly myopic, failing in cases where the robot needs to make an informed decision with regard to the global structure of the environment outside of its sliding map. This work addresses the planning problems associated with navigating safely at high-speeds in unknown environments while taking into account the computation and memory limitations for real-time operation. The contributions are as follows:

- A computationally-efficient local receding horizon planner using an adaptive motion primitive set and probabilistic collision checking to ensure safety when operating at high-speeds.
- The online construction of a memory-efficient 3D global sparse topological graph using geometric line of sight visibility.
- A hierarchical planning architecture (Fig. 2) enabling rapid replanning and the ability to efficiently plan

The authors are affiliated with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. mcollin1@alumni.cmu.edu, nmichael@cmu.edu

We gratefully acknowledge support from industry.

outside of a local map using a sparse topological environment representation. Using the proposed approach, the robot may safely reroute and reach the desired goal in scenarios where competing methods fail. Simulation results show the efficacy of the proposed framework in complex, large-scale environments

II. RELATED WORKS

1) Planning for High-Speed MAV Flight: For high-speed flight, receding horizon planners must be capable of operating at fast rates to ensure safety. Reactive approaches [9] enable fast collision avoidance, but are myopic. Motion primitives are a common method due to their fast calculation, and are used by Florence et al. [2], one of the first works to integrate perception with planning for high-speed flight, and by Ryll et al. [16], who use a fixed set of 270 primitives and two 60m×60m sliding grids for collision avoidance and global planning. Both works, however, are restricted to a fixed primitive set. Tordesillas et al. [19] use a multi-fidelity model to better capture some of the higher-order dynamics to improve the interaction between the local and global planners, and extend this approach in [20] to enable the robot to plan into unknown space through a MIQP formulation for faster flight speeds. Mohta et al. [11] present a framework using a search-based planner [6] that reaches velocities of 18m/s in uncluttered environments.

2) Topological Graphs: Topological graphs represent the connectivity of space within an environment, and therefore can be utilized for fast, efficient planning using a small set of nodes. Thrun [18] develops one of the earliest works of topological planning in robotics by constructing a 2D topological graph; however, due to computational demands, few approaches operate in 3D. Two methods that do are that from Blochliger et al. [1] that generating convex free space clusters from a feature-based SLAM system and that from Oleynikova et al. [15] that extracts a 3D Generalized Vornoi Diagram from an Signed Distance Field; however, both of these methods assume an initial exploratory phase to generate a map that may be used to revisit locations and are not designed for online usage. Liu et al. [8] enables online usage by leveraging a geometric path to seed a convex decomposition, but is limited to a single homotopy and cannot provide information about alternative pathways.

III. LOCAL TRAJECTORY SELECTION

For high-speed, aggressive flight in unknown environments the robot must be able to plan at a high rate locally to ensure safety with respect to the objects revealed through the incoming sensor measurements. The local planner in this work consists of three main components—an adaptive motion primitive library (Sect. III-A), a probabilistic collision checking method (Sect. III-B), and a search-based local path finder (Sect. III-C)—that will be detailed within this section.

A. Motion Primitive Library Generation

For accurate position tracking for multirotors—a differentially flat system—trajectory references should be continuous through high-order derivatives of position [10]. To

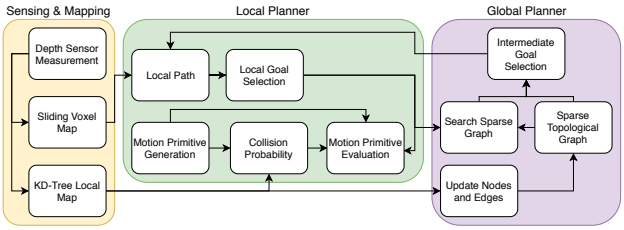
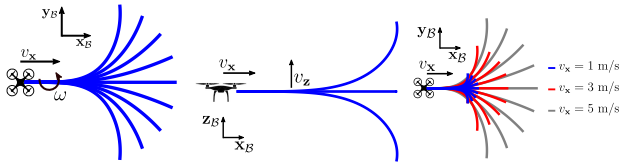


Fig. 2: System diagram of the local and global planning components. The local planner is discussed in Sect. III, the global planner in Sect. IV, and their integration in Sect. V.



(a) ω variation (b) $v_\mathbf{z}$ variation (c) $v_\mathbf{x}$ variation

Fig. 3: Parameterization of forward-arc motion primitive library [21]

quickly replan, this work utilizes the forward-arc motion primitives framework for MAVS proposed by Yang et al. [21]. These primitives are generated as time-parameterized 8^{th} order polynomials (1) with an action space defined by $\mathbf{a} = [v_\mathbf{x}\, v_\mathbf{z}\, \omega]$, where $v_\mathbf{x}$ and $v_\mathbf{z}$ are the desired forward ($\mathbf{x}_\mathcal{B}$ axis) and altitude ($\mathbf{z}_\mathcal{B}$ axis) end velocities in the body frame $\mathcal{B}$ of the robot, and ω is the yaw angular rate (Fig. 3). The high order of the polynomial ensures that the trajectory will be smooth (snap-continuous) and that it can be accurately tracked by the robot. Please refer to [21], [17] for a more detailed description on forward-arc motion primitives.

$$\dot{\gamma}(t) = [v_\mathbf{x}\cos(\omega t) \quad v_\mathbf{x}\sin(\omega t) \quad v_\mathbf{z} \quad \omega], \quad t \in [0, \tau] \quad (1)$$

To create a library the action space can be uniformly sampled; however, a trade-off exists between maneuverability (resolution) and computation (library size). Instead of building a fixed primitive set as is done in [16] and [2], the proposed framework utilizes an adaptive method for sampling over ω and $v_\mathbf{x}$ that is built around the current reference velocity of the robot v_r. Due to the high replan rate and the system dynamics, the speed of the robot will naturally ramp up or down (choose primitives close to its current speed), while acceleration constraints limit the jump between velocity levels in the primitive library. By taking an adaptive approach, the size of the library is reduced, enabling faster computation times without a substantial loss in performance. An example of three velocity levels is provided in (2). δ_{vl} and δ_{vh} set the velocity change for each level from v_r, while v_{min} sets the minimum velocity and v_{max} is the velocity limit. Due to the dynamic limits of the robot, large values of ω will be infeasible at high-speeds leading to a smaller, coarser set. Therefore, ω is scaled with the speed of the robot, so at low speeds the bound is high and the primitives are fanned out, and at higher speeds the primitives are more concentrated providing fine-grained heading resolution.

$$v_\mathbf{x} = [\max(v_r - \delta_{vl}, v_{min}), v_r, \min(v_r + \delta_{vh}, v_{max})] \quad (2)$$

B. Collision Checking

To ensure safe navigation within unknown environments that also accounts for uncertainties within the robot system (e.g., state estimation, sensor or control noise), a probabilistic collision checking strategy is utilized. The environment is represented as a spatially consistent kd-tree local map that stores a history of depth sensor measurements obtained at poses that lie in the vicinity of the vehicle's current pose [17]. This approach runs at sensor rate, ensuring that the most recent sensor information is incorporated for collision checks to maintain safety.

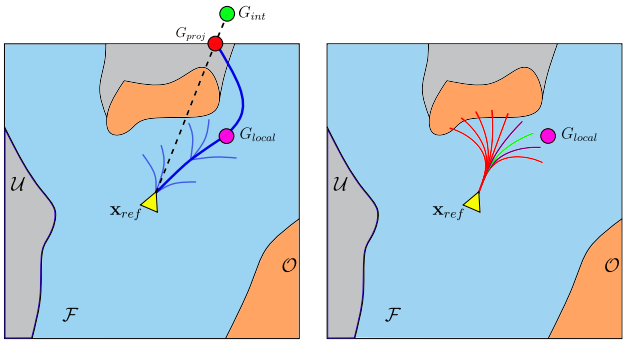
The proposed framework utilizes the methodology from (8) in Zhu et al. [23] that assumes the state uncertainty follows a Gaussian distribution, but provides a tight bound on the collision probability. The collision probability $P(C, p_i)$ for a position p_i along the trajectory may be calculated solely through the mean and covariance of the robot's position. For safety, the upper bound is taken as the collision probability and any point outside of the local map's set of field of view frustums is considered occupied. To get the collision probability for the entire primitive, samples are taken uniformly in time along the trajectory; an independence assumption for each point along the path is made, so that the total probability $P(\gamma, C)$ is just the product of the non-collision probabilities $P(\neg C, p_i)$ at each sample point.

The robot position covariance is scaled linearly with the speed of the vehicle so that the collision probability is dependent upon not only the distance to the obstacles, but also the velocity vector of the robot; this method has been shown to work at high-speeds and state uncertainties by Florence et al. [2] . At low speeds, the position uncertainty will be small, allowing the robot to move more tightly around obstacles. At higher speeds the covariance will be larger, increasing the effective collision radius for the robot. This will push the robot further away from obstacles or force the robot to slow down to remain safe; both behaviors are desirable to account for the lower control margin as the flight approaches the dynamic limits of the robot.

C. Selection of Local Goal and Primitive

To handle environments containing non-convex obstacles, the robot must determine how to move around obstacles to reach a desired goal. The proposed planning framework uses an efficient kinodynamic planner [22] to provide guidance for the primitive selection. Since this planner incorporates the robot's dynamics, the search is biased in the direction of the robot's motion, preventing paths from lying in infeasible homotopies, as can occur with geometric planners.

At each planning iteration, a reference state $\mathbf{x}_{ref}$ is chosen at a future time along the current trajectory to be used as the starting point for the local planner. The kinodynamic planner is run over a sliding voxel grid representation of the environment from this reference to a goal position G_{proj} that is obtained through a straight-line projection of the intermediate goal G_{int} chosen by the global planner (discussed in Sect. IV) onto the sliding map boundary. A local goal G_{local} is then chosen from this local path returned from the kinodyamic planner. Criteria for the local goal includes an angle difference between the robot heading and a vector to a node along the path being greater than δ_{ang} degrees or the altitude changing by more than δ_zm; otherwise, the end point of the local path is chosen. Due to the the initial state of the robot and the limited primitive resolution during the search, there is greater variability in the kinodynamic planner as compared to a geometric planner that may provide an inconsistent path; therefore, Jump Point Search (JPS) [3], a geometric A* variant, is run as a backup if there are large changes between planning iterations. The local path search runs in a separate thread at 5Hz.



(a) Local goal selection (b) Motion primitive evaluation

Fig. 4: Local planner example. The map depicts free $\mathcal{F}$, occupied $\mathcal{O}$, and unknown space $\mathcal{U}$. (**Left**) The intermediate goal G_{int} is projected onto the sliding map to yield G_{proj}. A path ( ) between $\mathbf{x}_{ref}$ and G_{proj} is calculated by running a kinodynamic search-based planner [22] from which a local goal G_{local} is selected. (**Right**) A motion primitive library ( ) is generated given the reference state. Primitives that are not dynamically feasible or that have a collision probability above a set threshold ( ) are discarded from consideration, and the primitive with the minimum cost ( ) is selected for execution.

This local goal is utilized as a component of the cost function for the evaluation of the motion primitive library. The total cost c_i (3) of primitive γ_i in the library Γ depends upon a weighted sum of a Euclidean progress metric c_{goal}, i.e., how close does the end position of the primitive $\mathbf{x}_i(\tau)$ comes to reaching G_{local}, a collision probability cost $c_{collision}$, and a smoothness cost $c_{heading}$ to penalize large changes in ω values from the previously selected primitive. The cost function seeks to balance progress towards the goal vs. safety in terms the robot's minimum distance to obstacles. If all primitives are infeasible or in collision, a stopping primitive that is calculated during the previous planning iteration is performed. An example showcasing the local planner algorithm is shown in Fig. 4.

$$\begin{aligned} c_i &= w_c c_{\text{collision}} + w_g c_{\text{goal}} + w_h c_{\text{heading}} \\ c_{\text{collision}} &= P(C, \gamma_i) \\ c_{\text{goal}} &= ||G_{local} - \mathbf{x}_i(\tau)|| \\ c_{\text{heading}} &= |\omega_i - \omega_{prev}| \end{aligned} \tag{3}$$

IV. GLOBAL SPARSE TOPOLOGICAL PLANNER

For memory and planning efficiency, this work represents the environment globally through a topological graph. This section details the geometric approach for online construction and updating of the graph and how it may be efficiently searched to enable operation in unknown environments.

A. Online Sparse Graph Construction

Since the sparse graph must be used for online planning, it is designed to allow for very fast updates through a geometric approach. The graph combines a history of the robot's positions during the course of operation with the identification of potential pathways that the robot has passed that could be used later if a dead end is encountered. This generates a rough topology of the environment based on where the robot has already traversed and locations where traversal is possible.

The sparse graph contains two types of nodes: position nodes and visibility nodes. Position nodes keep a time history of the robot's path, and are added to the sparse graph $\mathcal{G}_s$ if the robot has traveled a distance greater than a set threshold δ_p from a previously added node. To quickly determine the closest node from which to branch off, node positions are stored in a kd-tree structure. These nodes are the primary type, create the skeleton of the graph, and help the robot through previously explored sections of the environment.

The more important node type are *visibility nodes* that are added at critical points within the environment where potential pathways are detected, e.g., doorways or corridors that the robot has not yet entered. To identify these critical locations, a set of test points $\mathbf{t}_n$ are created that encircle the robot as it moves through the environment. Each test point has a set of neighbors that are used to check for its visibility through a geometry-based testing procedure. Since most critical points exist in 2.5D, this work utilizes eight test points on the x-y plane at the robot's current altitude with two neighbors on either side, but the test points could be extended to 3D. Points along the straight-line connections between the test node and its neighbors are checked for collision with the kd-tree local map. If more than one connection is blocked and the test node is visible from the current robot position, it is added to the sparse graph as a visibility node with an edge connection to the goal node n_g. When a visibility node is added, a position node at the robot's current location is also incorporated into the graph to maintain structure. The process for adding a visibility node is shown in Fig. 5.

To add a node into the sparse graph, three quantities are needed: the id of the node(s) to which it connects, the Euclidean distance (cost) between the nodes, and whether the edge is traversable, i.e., has the robot physically moved between the two nodes. An example of a non-traversable connection is the one made between a visibility node and the goal node. The edge cost for a visibility node is the Euclidean distance estimate cost-to-goal from the visibility node. A series of adjacency vectors store the sparse graph to allow for efficient modification and search. Checks for local additions and updates (e.g., removing goal connection after using visibility node) to the sparse graph are performed at each planning iteration using the kd-tree and adjacency structures. In cases where the robot experiences significant position drift during operation, the topological graph could be included in a tightly-coupled SLAM framework for global updates; however, this is outside the scope of this work.

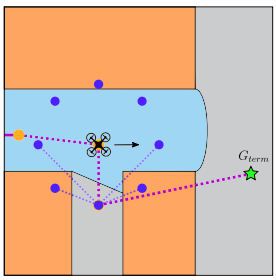
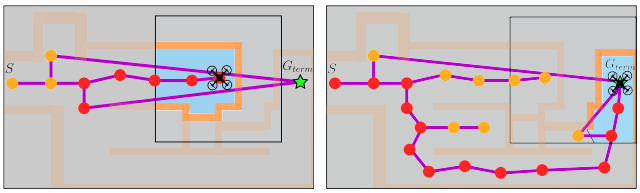


Fig. 5: Adding *visibility nodes*. If connections between test points (●) are blocked and the node is visible to the current robot position, it is added to the sparse graph with a connection to the goal node. Additionally, a position node is added at the robot's current location to maintain graph structure. Newly added edges are shown as dashed purple lines.



(a) Alternate Path (b) Return to Home

Fig. 6: Searching for paths through the sparse graph (yellow nodes and purple edges). The black rectangle represents the sliding map boundaries. Walls not in the sliding map are translucent. The found path is shown in red. (a) If a dead end is reached, the sparse graph may be queried for an alternate pathway towards the goal. (b) Once the robot reaches the goal, the sparse graph may be queried to safely return home by ensuring only traversable edges exist in the path.

B. Searching the Sparse Graph

Searches may be performed on the sparse graph either to find an alternative, unexplored path in hopes of reaching a desired goal or to return home to the starting location, as shown in Fig. 6. By planning on the sparse topological graph representation, both of these searches can be done very efficiently and utilize information outside of the robot's sliding map. Therefore, a notion of longer term memory is added—no longer limited to plan only within the boundaries and computational limits of a sliding map—that can aid the robot in navigating large, complex environments.

Searching through the sparse graph amounts to an A* search over a bidirectional graph to find the shortest (lowest cost) path from the robot's current location to a desired end location. Alternative path searches must contain a visibility node to connect to the goal node, while return to home searches are constrained to traversable edges to ensure the path lies in previously explored space. The returned paths are utilized by the global planner to guide the robot safely back through the environment.

V. INTEGRATION OF LOCAL AND GLOBAL PLANNERS

With the introduction of the sparse topological graph, the robot now has the ability to construct and plan upon a global representation of the environment, opening possibilities for the global planner to solve problems that were previously not achievable. This section details the integration between the local (Sect. III) and global (Sect. IV) planners. Refer back to Fig. 2 for the planning architecture system diagram. The combined planning routine is provided in Algorithm 1.

Algorithm 1: Planning Architecture

Input: $\mathbf{x}_{ref}, \mathcal{G}_s, G_{term}$
1 **function** PLAN()
2 $G_{int} \leftarrow G_{term}$, $\xi \leftarrow \emptyset$,
3 **while** $\mathbf{x}_{ref} \neq G_{term}$ **do**
4 $G_{int} \leftarrow$ GETINTERMEDIATEGOAL($\xi, \mathbf{x}_{ref}$)
5 $\gamma_j, G_{local} \leftarrow$ SELECTTRAJECTORY(G_{int}, $\mathbf{x}_{ref}$)
6 **if** $\gamma_j = \emptyset$ **then**
7 Execute stopping maneuver
8 **else**
9 Execute primitive
10 **if** $G_{int} = G_{term}$ **then**
11 $b_r \leftarrow$ UPDATESPARSEGRAPH($\mathbf{x}_{ref}, \mathcal{G}_s$)
12 **if** $G_{local} = \emptyset$ || $b_r = True$ **then**
13 $\xi \leftarrow$ SEARCHSPARSEGRAPH()
14 **function** SELECTTRAJECTORY($G_{int}, \mathbf{x}_{re}$)
15 $G_{proj} \leftarrow$ project G_{int} onto sliding map boundary
16 $\Gamma \leftarrow$ REGENERATEPRIMITIVELIBRARY(v_{ref})
17 $G_{local} \leftarrow$ GETLOCALGOAL(G_{proj}, $\mathbf{x}_{ref}$)
18 **forall** $\gamma_i \in \Gamma$ **do**
19 $P(C, \gamma_i) \leftarrow$ GETCOLLISIONPROBABILITY(γ_i)
20 $c_i \leftarrow$ PRIMITIVECOST($P(C, \gamma_i), G_{local}, \mathbf{x}_i(\tau)$)
21 $j \leftarrow$ index of minimum cost primitive
22 **return** γ_j, G_{local}
23 **function** UPDATESPARSEGRAPH($\mathbf{x}_{ref}, \mathcal{G}_s$)
24 $\mathbf{t}_n \leftarrow$ UPDATETESTNODES($\mathbf{x}_{ref}$)
25 $n_v \leftarrow$ CHECKFORVISIBILITYNODES($\mathbf{x}_{ref}, \mathbf{t}_n$)
26 $n_p \leftarrow$ CHECKFORPOSITIONNODES($\mathbf{x}_{ref}$)
27 **if** n_v || n_p **then**
28 $\mathcal{G}_s \leftarrow$ update sparse graph structure and kd-tree
29 **return** REVISITCHECK($\mathbf{x}_{ref}, \mathcal{G}_s$)

At the onset of each planning iteration, the global planner provides an intermediate goal G_{int} for the local planner (line 4), selecting either the terminal goal G_{term} or a waypoint from the path ξ returned from searching the sparse graph. The local planner takes in this intermediate goal and uses it during the process to select the primitive for the robot to execute (lines 5, 14-22). Additionally, during each planning cycle updates are made to the sparse graph if applicable, i.e., not following an alternate path (lines 11, 23-29).

One crucial interaction within the hierarchical planning architecture is to determine when to search over the sparse graph. There are two reasons to trigger a search (line 12): no local path can be found from the kinodynamic/JPS planner within the sliding voxel map for an extended time period (a threshold of 2s is used) or if the global planner detects that the robot is backtracking, i.e., revisiting a node within the sparse graph $\mathcal{G}_s$. If a search over the sparse graph is performed, the closest waypoint in ξ to the robot's current location is set as the intermediate goal. Once the robot moves within a distance ϵ to the waypoint, the intermediate goal shifts to the next point in the path. This continues until the intermediate goal returns to G_{term}.

TABLE I: Comparison of the proposed adaptive motion primitive library approach against a fixed size library. Results are for 20 trials in random block and forest environments.

Motion Primitive Library Type	# of Primitives	Comp. Time (s)	Success Rate	Avg. Time (s)	Min. Obstacle Dist. (m)	% Longer than RRT* Path
Adaptive	85	**0.008**	**1.0**	10.14	**0.61**	2.5
Fixed	272	0.02	0.95	**9.98**	0.58	**1.6**

VI. RESULTS

The proposed approach is evaluated over four simulation experiments to test the performance of the local planner and the full architecture. For all experiments, an adaptive motion primitive library containing 85 primitives and a collision probability threshold of 0.3 is utilized. The robot has dynamic limits $v_{max} = 10$m/s, $a_{max} = 10$m/s^2, and $j_{max} = 35$m/s^3, a 20m×20m×4m sliding map, and a depth camera with a FOV of 90° and range of 10m updated at 10Hz. Simulations are run in real time on an Intel Core i7-5820K with 32GB RAM.

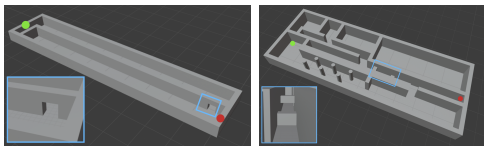
The proposed adaptive motion primitive library is compared against a fixed library over the course of 20 trials in random block and forest environments measuring 60m×40m×10m with 100-200 obstacles to evaluate its effect and the performance of the local planner. The results are shown in Table I with the adaptive approach having a slightly longer solution time and path length, but a significantly faster computation time due to the lower number of primitives that need to be iterated over. Since all obstacles are convex, the path length are compared to those found using RRT* [5] to highlight the local planner gets close to the optimal shortest path solution (∼2.5% longer).

To further assess the performance of the local planner and to put it into the context of state-of-the-art methods for planning in unknown environments, the proposed approach is compared against FASTER in the bugtrap environment provided in their work [20]. The results are shown in Table II. In terms of time to reach the goal, the proposed approach performs comparably being only 0.6s slower, but travels 6m further. Since the local path search currently only relies on the fused map data and not the most recent sensor information, as in FASTER, there is a delay before the robot reacts, leading to the robot taking a non-optimal path. Overall, the proposed local planner performs on a similar level as state-of-the-art methods, but this result highlights areas that could be improved.

To evaluate the performance of the integrated planning architecture, simulation experiments are run in two large-scale, complex environments (Fig. 7) that feature dead ends that force the robot to reroute in order to reach the goal. Due to the size of the environments and the start/goal locations, competing approaches that rely solely on sliding maps will

TABLE II: Comparison between the proposed local planner and FASTER on a bugtrap environment [20].

Method	Distance (m)	Time (s)
FASTER [20]	**55.2**	**13.8**
Proposed	61.26	14.38



(a) Corridor (b) Multi Corridor

Fig. 7: Simulation environments. The corridor measures 100m×20m×6m with the doorway 5m from one end, while the multi corridor (100m×35m×8m) contains many dead ends and obstacles. Start locations are shown in red and goals in green. Insets highlight the doorway and 3D obstacles.

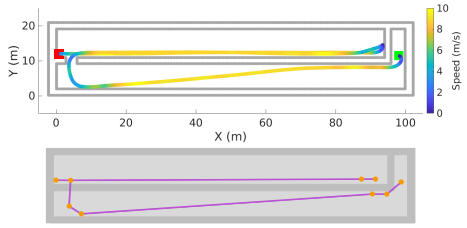


Fig. 8: (Top) Trajectory through the corridor environment colored according to flight speed. (Bottom) The global sparse graph created during the flight.

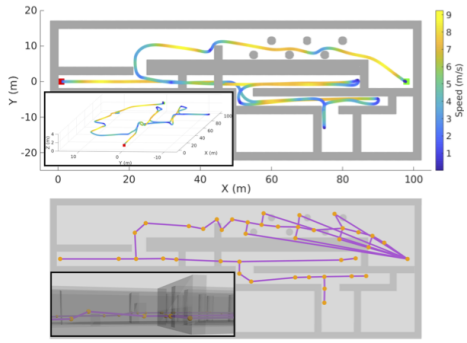


Fig. 9: (Top) Trajectory through the multi corridor environment colored according to flight speed. (Bottom) The global sparse graph after the robot reached the desired goal. Insets highlight the 3D geometry of the trajectory and graph as the robot maneuvers above and below obstacles within the environment.

fail. Using the proposed planning framework in the corridor environment, the robot is able to successfully reach the desired goal by searching over the sparse graph to plan back to the doorway encountered early on. The resultant trajectory and sparse graph are shown in Fig. 8, with the robot reaching a maximum speed of 9.88m/s. Due to the complexity of the multi corridor environment, the global sparse graph must be searched on three separate occasions to identify potential pathways before the robot finally reaches the desired goal. The trajectory and sparse graph for this environment is shown in Fig. 9. A maximum speed of 8.77m/s was achieved, while the average speed for the flight was 4.33m/s. Images from the flights are provided in Fig. 1.

TABLE III: Timing results and memory usage measured during the multi corridor trial. Note: Search times for the sparse graph are dependent upon size and connectivity of the underlying graph, so no exact times are provided; however, the search times are orders of magnitudes faster than searching directly over a global voxel grid and are typically under 1ms. Refer to [15] for a comparison of sparse topological planning times to A* and RRT* approaches.

Component	Avg. Time (s)	Memory Usage (MB)
Collision Checking (Single Primitive)	0.00016 ± 0.00046	–
Primitive Selection	0.0141 ± 0.0085	–
Local Path Search	0.0009 ± 0.0065	–
Sparse Graph — Add Node	0.00054 ± 0.0009	–
Sparse Graph — Search	$<$ 1ms	–
Sparse Graph	–	0.00128
Global Voxel Map	–	27.32

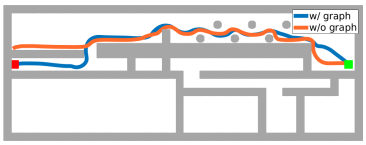


Fig. 10: Comparison of flight trajectories during a return to home maneuver with (succeeds) and without (fails) using the sparse graph.

The timing breakdown and memory usage for the proposed planner tested in the multi corridor environment is shown in Table III. Each iteration of the local planner takes on average 14ms (faster than typical sensor rate of 30Hz), while additions to the sparse graph only take 0.5ms. If a global voxel map were stored for the duration of the flight, the memory required would be 27.32MB. Contrast this with the proposed global sparse graph that requires only 0.00128MB, or over 21,000 times less memory. These numbers indicate that the proposed approach is sufficient for real-time operation, while generating safe, reactive trajectories even at high-speeds.

Since the history of the robot's path underpins the global sparse graph, it may be utilized to accurately return to the starting location. A comparison (Fig. 10) is performed for planning back to the start location after reaching the goal in the multi corridor environment with and without using the sparse graph. With the sparse graph, the robot can safely and accurately return to the starting location along its previously explored route. When the sparse graph is not utilized, the robot takes a longer initial path since it is unaware of the obstacles, gets stuck in a previously unseen section of the environment, and fails to reach the home position.

VII. CONCLUSIONS

This work presented an efficient planning architecture consisting of a local receding horizon component and a global sparse topological graph constructed online to enable high-speed MAV flight in large-scale, unknown environments. Results show the proposed approach generates safe trajectories at flight speeds up to 10m/s, and highlight the computational and memory efficiency of the approach. Future work includes extending to dynamic obstacles, adding visual features to make the sparse graph addition more robust, and the incorporation of frontiers into the local goal selection.

REFERENCES

[1] Fabian Blochliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Roland Siegwart. Topomap: Topological mapping and navigation based on visual slam maps. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[2] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[3] Daniel Damir Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[4] Markus Hehn and Raffaello D'Andrea. Quadrocopter trajectory generation and control. *IFAC Proceedings Volumes*, 44(1):1485–1491, 2011.

[5] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[6] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2872–2879. IEEE, 2017.

[7] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018.

[8] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.

[9] Brett T Lopez and Jonathan P How. Aggressive 3-d collision avoidance for high-speed navigation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5759–5765. IEEE, 2017.

[10] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525. IEEE, 2011.

[11] Kartik Mohta, Ke Sun, Sikang Liu, Michael Watterson, Bernd Pfrommer, James Svacha, Yash Mulgaonkar, Camillo Jose Taylor, and Vijay Kumar. Experiments in fast, autonomous, gps-denied quadrotor flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7832–7839. IEEE, 2018.

[12] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.

[13] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5332–5339. IEEE, 2016.

[14] Helen Oleynikova, Zachary Taylor, Alexander Millane, Roland Siegwart, and Juan Nieto. A complete system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments. *arXiv preprint arXiv:1812.03892*, 2018.

[15] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Sparse 3d topological graphs for micro-aerial vehicle planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[16] Markus Ryll, John Ware, John Carter, and Nick Roy. Efficient trajectory planning for high speed flight in unknown environments. In *2019 IEEE international conference on robotics and automation (ICRA)*, pages 732–738. IEEE, 2019.

[17] Alex Spitzer, Xuning Yang, John Yao, Aditya Dhawale, Kshitij Goel, Mosam Dabhi, Matt Collins, Curtis Boirum, and Nathan Michael. Fast and agile vision-based flight with teleoperation and collision avoidance on a multirotor. In *Proc. of the Intl. Sym. on Exp. Robot*. Springer, 2018, to be published.

[18] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[19] Jesus Tordesillas, Brett T Lopez, John Carter, John Ware, and Jonathan P How. Real-time planning with multi-fidelity models for agile flights in unknown environments. *arXiv preprint arXiv:1810.01035*, 2018.

[20] Jesus Tordesillas, Brett T Lopez, Michael Everett, and Jonathan P How. Faster: Fast and safe trajectory planner for flights in unknown environments. *arXiv preprint arXiv:2001.04420*, 2020.

[21] Xuning Yang, Ayush Agrawal, Koushil Sreenath, and Nathan Michael. Online adaptive teleoperation via motion primitives for mobile robots. *Autonomous Robots*, pages 1–17, 2018.

[22] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[23] Hai Zhu and Javier Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.