Real-Time 3D Navigation for Autonomous Vision-Guided MAVs

Shengdong Xu, Dominik Honegger, Marc Pollefeys, and Lionel Heng

Abstract— **Autonomous navigation of micro aerial vehicles (MAVs) in a-priori unknown environments is one of the most challenging problems in robotics. First, a MAV has to incrementally build a 3D geometric map from raw sensor data. Then, based on the mapping information, the path planner has to search for a cost-optimal trajectory to the goal in real-time. It is common practice to discretize the search space into a state lattice; by doing so, we reduce the path planning problem with differential constraints to a graph search problem that is easier to solve. However, a regular 3D state lattice requires a large amount of memory while graph search in a regular 3D state lattice incorporating numerous states is computationally intensive. In this paper, we introduce a novel path planning algorithm which extends the concept of a regular state lattice to an octree-based state lattice, and searches for an optimal trajectory in the octree-partitioned search space. Our octree-based state lattice representation discretizes large swathes of free space into few symbolic octants, and thus, encodes a significantly fewer number of states. As a result, memory consumption is kept to a minimum, and at the same time, graph search is made more efficient. Simulation experiments demonstrate the efficiency of path planning with an octree-based state lattice, and further field trials prove the viability of this path planning algorithm.**

I. INTRODUCTION

Due to their small size and high maneuverability, MAVs have become a powerful tool for rescue, surveillance, and exploration. To complete these types of tasks, the MAVs have to fly autonomously in a-priori unknown environments. Autonomous navigation requires the MAVs to simultaneously perform pose estimation, environment map building, and path planning. Ideally, all processes should be run on-board as off-board processing requires a connection to the ground station, and such a connection is susceptible to interference. However, the amount of on-board computational resources is constrained by the payload limit. All on-board processes therefore need to be computationally efficient. The focus of our work is to develop an efficient 3D path planning algorithm for MAVs.

MAVs are able to maneuver in 3D space; one challenge that comes along with this high-dimensionality maneuver is planning an optimal path in 3D space. Such a task is difficult because the environment might be prohibitively too large for a 3D path planner to search for an optimal trajectory. Another problem is that the path planning algorithm has to run at the same time as the mapping module which builds the map of the environment. This combined processing leads to intensive computation, and thus, real-time planning and re-planning are difficult to achieve.

For well-established path planning algorithms, the planner usually discretizes the 3D search space into a regular state lattice [1]; the state space discretization is regular in the 3D translational coordinates. Given the fact that the quadrotor dynamics are differentially flat [2], we can express the control inputs as a function of four flat outputs $[x, y, z, \psi]$ and their derivatives, where $[x\ y\ z]^T$ are the coordinates of the center of mass of the quadrotor and ψ is the yaw angle. Hence, we define the MAV's state to be $[x\ y\ z\ \psi]^T$ with zero roll and pitch, which results in a 4D state space. Edges between states are established by motion primitives which can be generated by a polynomial function with proper boundary conditions. The memory usage of the state lattice representation grows significantly as the range of flight heights increases, making path planning in full 3D space nearly impossible. Another drawback of this regular state lattice representation is that the path planner will always search for high-resolution paths even in large volumes of free space or unexplored space, which turns out to be inefficient.

In our work, we discretize the 3D search space into an octree-based state lattice which inherently represents large free areas as large octants and areas near obstacles as small octants. As a result, memory usage is kept to a minimum, and at the same time, the path planner is able to search for high-resolution paths in cluttered regions, and low-resolution paths in sparse or unexplored regions. Consequently, the planner only consumes a small portion of memory but is able to plan and re-plan near-optimal paths quickly.

The contributions of this work are as follows: we come up with a novel octree-based state lattice which is a memory-efficient discrete representation of the search space, and we present a method of finding a near-optimal path based on the octree-partitioned 3D space by using standard graph search algorithms such as A*, or its variants D* [3] and AD* [4]. We use a previously developed 3D mapping module [5] to incrementally map out the environment, and show both simulation and real-world results from our novel path planning algorithm.

II. RELATED WORK

Autonomous operation of MAVs in a-priori unknown environments has become a hot research topic in the last few years. In particular, a computer-vision-based approach is advantageous because a camera is light-weight, has a low power consumption, and has a two-dimensional field of view.

S. Xu, D. Honegger, and M. Pollefeys are with the Department of Computer Science, ETH Zürich, Switzerland. {xus@ethz.ch, dominik.honegger@inf.ethz.ch, marc.pollefeys@inf.ethz.ch} L. Heng is with the Information Division, DSO National Laboratories, Singapore. {hjianyon@dso.org.sg}

Dominik Honegger is partially supported by the Swiss National Science Foundation (SNF) under grant 150151.

However, computer vision algorithms usually require intensive computation and large amounts of memory, which makes it difficult for other computationally intensive algorithms such as occupancy mapping and path planning to run at the same time. Thus, in order to achieve on-board vision-guided autonomous 3D navigation, an efficient 3D path planning algorithm is needed.

There are several well-known issues that make on-board 3D path planning for MAVs challenging. One issue is the differential constraints with respect to the motion of MAVs and which increase the complexity of the path planning problem. In addition, grid-based discrete representations of the search space consume a lot of memory. Another issue is that the path planner should be able to plan and re-plan an optimal path in real-time with limited computational resources.

For two-dimensional path planning problems, there are many existing successful implementations [6], [7]. In contrast to the 2D case, path planning in 3D space is far more computationally expensive and memory-intensive, and thus, more difficult. In [8], path planning was performed in a full octree-partitioned 3D space but without due consideration to differential constraints, and the applied potential field algorithm does not guarantee an optimal path. In [9], path planning was performed in a regular 3D state lattice but with a limited range of flight heights. As the flight height range increases, the memory usage will grow significantly, eventually reaching a point at which the required computational resources will exceed those already available. Representing the entire motion space with a regular state lattice is inefficient in several applications such as path planning over long distances. A lower resolution suffices for approximating sparse or unexplored regions, while cluttered regions require high-resolution discretization. To address this problem, [10] created both low-resolution and high-resolution state lattices. The planner performs high-resolution search in the vicinity of the robot while it performs low-resolution search elsewhere. Similarly, [11] creates a high-resolution state lattice that moves with the robot, and a low-resolution grid elsewhere. The combination of the state lattice representation and motion primitives [12] was found to be a good method for addressing the MAV path planning problem with differential constraints. We further improve on both the regular and multi-resolution state lattice representations by creating the concept of an octree-based state lattice which significantly reduces the required computational resources at the cost of a slightly lower path optimality.

III. Octree-Based 3D Path Planning

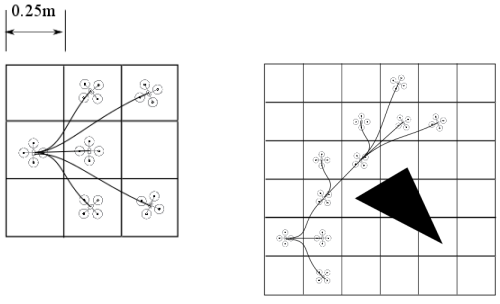
In our octree-based path planning algorithm, the 3D search space is discretized into an octree data structure based on the 3D occupancy map built by the 3D mapping module. At the beginning, a single octree node or *root node* represents the entire and empty 3D space. We then update the octree with information about obstacles from the 3D occupancy map. For each labeled voxel in the 3D occupancy map and whose label either corresponds to free or occupied space, we recursively subdivide the root node into 8 child octants until we reach the maximum resolution, and subsequently, update the leaf node's label. Our algorithm plans a path that is constrained to go through the centres of octants.

For path planning, an octree offers a significant advantage over a regular 3D grid: the amount of memory required is significantly smaller, especially in sparse environments. The octree is able to store information about the environment layout without any loss in accuracy. Moreover, the octree provides a means for decomposing large sections of free space into fewer symbolic units, and thus, reduces the computational burden of graph search.

We leverage the concept of a state lattice to enforce differential constraints in the path planning. However, a conventional state lattice only works for a 3D regular grid, and hence, we modify the state lattice concept to work for an octree-based search space.

A. Octree-Based State Lattice

A state lattice is a discrete representation of the configuration space; this representation comprises a set of states, and transitions between states can be represented by a series of motion primitives. The primitive motions can be seen as a canonical set of short feasible control samples that satisfy the differential constraints of the system. This set of motion primitives can be used to form the state transitions in the lattice. In our implementation, we discretize the state space with a maximum resolution of $25cm$ and 22.5° yaw intervals. The pre-computed canonical set of maximum-resolution motion primitives consists of turning on the spot in both the left and right directions, moving up and down vertically, and moving forward and backward in several directions. The cost of a primitive motion is proportional to its path length except for the turning movement which has a cost corresponding to a $25cm$ path length. Backward movements are penalized with a weighted factor greater than 1 as obstacles at the rear cannot be observed with a forward-facing camera, and thus, we want to avoid backward movements if possible. We use Figure 1 to illustrate examples of primitive motions in a 2D plane and how they form state transitions in the lattice.



(a) (b)

Fig. 1: (a) motion primitives in a 2D plane. (b) one example of the state transitions.

We call the states within the octree-based state lattice *octree node states*. Before the octree-based state lattice can be reused for graph search, we have to resolve two issues described below.

1) Adjacency between octree node states: The first issue is establishing the adjacency between octree node states [13] in the octree-based state lattice. Here, we use a naive method to determine whether two octants are adjacent to each other; we check whether the distance between the centres of the two octants along each of the x, y and z directions exceeds half of the sum of the two octants' cell sizes, and if so, we ascertain that the two octants are not adjacent. Each time a node gets split, we look for adjacency relationships with respect to the node's children. The pseudocode of the octant-neighbor-finding algorithm is shown in Algorithm 1.

Algorithm 1 Algorithm to find neighbors for each of a *node*'s children when the node is split.

if *node* is no longer a leaf due to updated map information **then**
  Split *node* into eight children (child[i], $i \in 1, 2, .., 8$)
  for $i = 1 \rightarrow 8$ **do**
    child-node = child[i]
    add the *child-node*'s brothers (child[j], $j \neq i$, $j \in 1, 2, ...8$) as neighbors
    for each neighbor of *node*, *neighbor*[k] **do**
      if *neighbor*[k] is still the neighbor of *child-node* after the split **then**
        add *neighbor*[k] as the neighbor of *child-node*
      end if
    end for
  end for
end if

Adjacency relationships between octants are not sufficient for establishing edges between octree node states; we still need to find feasible paths between each pair of adjacent octants. If two adjacent octree node states are located on different levels of the octree, it is not possible to represent the edge between these two octree node states as a single motion primitive in the canonical set. However, it is possible to represent the path between the two states as a sequence of motion primitives which can be precomputed beforehand for every possible pair of octree levels.

We precompute a lookup table; given a small fixed-size state lattice, for a state located at the center of the lattice and with a given heading direction, we store the cost and decomposition of a path from that state to all other states in the state lattice. The precomputation step shown in Algorithm 2 can be performed via the Dijkstra's algorithm. The lookup table can be used to inform the path planner how two octree node states connect to each other and the cost of a feasible path between the two states. For any two octree node states $(x_1, y_1, z_1, \theta_1)$ and $(x_2, y_2, z_2, \theta_2)$, the lookup index is ($\theta_1$, $x_1 - x_2$, $y_1 - y_2$, $z_1 - z_2$, θ_2). The size of this lookup table depends on the number of the heading direction intervals and the maximum number of levels of the octree.

Algorithm 2 Multi-resolution path lookup-table construction.

for $i = 1 \rightarrow 16$, $x = -N \rightarrow N$, $y = -N \rightarrow N$, $z = -N \rightarrow N$, $j = 1 \rightarrow 16$ **do**
  LUT_COST[i][x][y][z][j] = infinity
  LUT_PATH[i][x][y][z][j] = undefined
end for
for $i = 1 \rightarrow 16$ **do**
  for every state v in the state lattice **do**
    dist[v] := infinity
    previous[v] := undefined
  end for
  Q := empty priority queue
  s_start := the origin of the lattice with an orientation index i
  dist[s_start] = 0
  insert s_start into Q
  while Q is not empty **do**
    u := vertex in Q with minimum dist[u]
    remove u from Q
    for each neighbor v of u **do**
      j := the orientation index of v
      (dx, dy, dz) := the 3D coordinate difference between u and v
      checkdist := dist[u] + cost(u,v)
      if checkdist $<$ dist[v] **then**
        dist[v] := checkdist
        previous[v] := u
        LUT_COST[i][dx][dy][dz][j] = checkdist
        $waypoint$ = v
        clear LUT_PATH[i][dx][dy][dz][j]
        while $waypoint \neq s\_start$ **do**
          push back $waypoint$ to LUT_PATH[i][dx][dy][dz][j]
          $waypoint$ = previous[$waypoint$]
        end while
      end if
    end for
  end while
end for

To reduce the memory requirement of the lookup table, we exploit the symmetry of the state lattice. For instance, the path from $(0, 0, 0, \theta_1)$ to (x, y, z, θ_2) can be reflected in the $z = 0$ plane to get the path from $(0, 0, 0, \theta_1)$ to $(x, y, -z, \theta_2)$. We only need to store ascending paths, and we can simply infer descending paths from ascending ones. Similarly, we do not need to store information corresponding to all 16 possible states located at the origin of the state lattice; the 22.5° heading direction intervals result in 16 possible heading directions. In fact, we only need to store information corresponding to 3 states with the following headings: 0°, 22.5°, and 45°, because we can do reflection operations to obtain information corresponding to the other 13 states. Thus, we can reduce the memory requirement by

90% by exploiting the symmetry of the state lattice.

2) Pre-discretization: Another issue that has to be resolved for the octree-based state lattice to be usable for graph search is that large octants may be present in the octree, and as a result, the path planner may compute highly suboptimal paths. In addition, we need a large multi-resolution lookup table to store edges between large octants and other octants, which means that a large amount of memory has to be allocated. In order to address this issue, before performing path planning, we first enforce a minimum octree level on all leaf nodes by splitting nodes whose levels exceed the minimum level. In this way, the optimality of the final path is improved, and at the same time, we significantly reduce the size of the multi-resolution lookup table. We use Figure 2 to illustrate the impact of pre-discretization on the optimality of the path in the 2D case. In the 3D case, the influence of pre-discretization on the optimality of the final path is far greater.

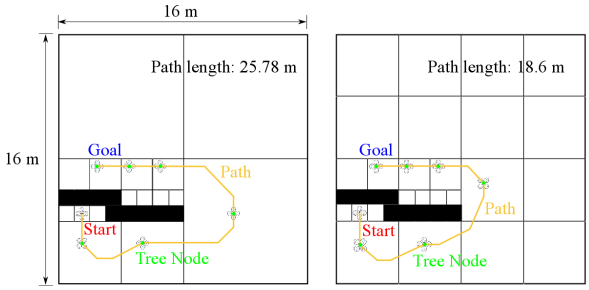


Fig. 2: The final path obtained from an octree-based-state-lattice path planner without pre-discretization is shown in (a). The path with pre-discretization is shown in (b). The length of the path in (a) is 25.78 m while the length in (b) is only 18.61 m.

The minimum octree level used for pre-discretization is a very important factor that influences the performance of the octree-based-state-lattice path planner. If the minimum octree level is too high, the graph would nearly resemble a regular 3D state lattice. Furthermore, the graph search becomes slower due to the increased number of states. On the other hand, if the minimum octree level is too low, the planned path becomes highly suboptimal. Here, we choose the minimum octree level to be a third of the maximum height of the octree. Empirically, this strategy performs well.

B. Local 3D State Lattice

Considering the fact that path planning in the vicinity of the robot is critical for obstacle avoidance, we maintain a small local regular 3D state lattice with maximum resolution and which moves along with the robot. Here, we simply adopt the state lattice data structure which was used to build the multi-resolution lookup-table in the previous step. This local high-resolution state lattice centered on the MAV enables the path planner to do precise path planning in the vicinity, and thus, help the MAV navigate around nearby obstacles. The graph search finds a near-optimal path by iterating through the states in both the local regular and global octree-based state lattices.

C. Graph Search

After creating both the local regular state lattice and the global octree-based state lattice, the next step is to perform graph search to find an optimal path. Any edge in the octree-based state lattice can be decomposed into a series of high-resolution primitive motions. The edge corresponds to a feasible path whose costs and decompositions have been pre-computed and stored in the multi-resolution path lookup table described in the previous section. The octree-based state lattice is dynamically changing with constant 3D map updates and as the robot moves.

1) Optimal Path Finding: To show the efficiency of our octree-based state lattice, we use a simple A^* graph search algorithm to generate optimal paths. Of course, it is possible to implement AD^* or any other variant of the A^* graph search algorithm for the octree-based state lattice. The edge costs between octree node states can be obtained from the multi-resolution path lookup table. The graph search algorithm finds an optimal trajectory that consists of octree node states.

The performance of the A^* algorithm heavily depends on the quality of the heuristic function. Without informative heuristics, the A^* algorithm expands as many states as Dijkstra's algorithm does, and thus, is inefficient when searching for an optimal path. The heuristic estimate should underestimate the actual path cost; otherwise, the A^* algorithm will return suboptimal solutions. On the other hand, the heuristic estimate should be as close as possible to the actual path cost, such that the graph search processes fewer states and is more efficient. In our implementation, we applied the holonomic-with-obstacles heuristic [14] which ignores the differential constraints but takes obstacles into consideration when estimating the future cost. As the differential constraints are not considered here, this heuristic function is simple to compute, and thus, can be performed online. In addition, the heuristic function considers obstacles, and thus, is informative enough to guide the graph search along promising directions.

The octree-based path planner is constrained to plan trajectories through the centers of octree node states and states in the local regular 3D state lattice. By using an octree-based state lattice, we greatly reduce the number of candidate states for the A^* algorithm to expand. As a result, the A^* algorithm is able to find the best path in a short time. This fast planning strategy inevitably introduces a drawback: the octree-based-state-lattice path planner returns a near-optimal path in comparison to the optimal path obtained from a conventional state-lattice-based planner.

2) Path Reconstruction: Suppose that we have already obtained an optimal trajectory represented by a series of octree node states from the A^* algorithm. We still need to reconstruct the full path because the actual path from one octree node state to another is actually a series of

high-resolution primitive motions. To complete this step, we can simply look up the path decompositions in the multi-resolution path lookup table. An example of the graph search and path reconstruction in the 2D case is shown in Figure 3.

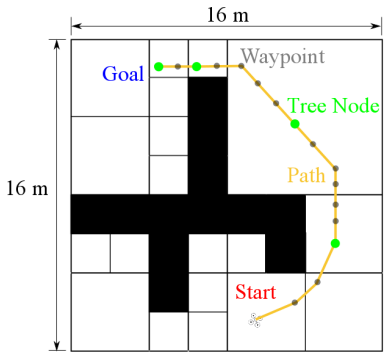


Fig. 3: Full path reconstruction from an octree node state trajectory. The green dots represent octree node states, the grey dots represent actual waypoints, and the clay-colored line represents the final full path.

In the above example, there are only four tree node states along the path but there are many more waypoints along the full path. This is because the actual path from one tree node state to another is decomposed into a series of high-resolution primitive motions.

IV. Simulation Experiments

We first tested our approach in a simulator. The simulation experiment involved the use of both a simulator and ROS. We use the V-REP simulator [15] from Coppelia Robotics. V-REP was used to simulate a MAV with a RGB-D sensor, and the environment which the simulated MAV operates in. The maximum velocity of the MAV was set to be 0.25m/s. We used rviz to provide a 3D visualization of the processed data generated by the MAV. We did two sets of experiments to compare the performance of graph search based on the octree-based state lattice representation and that based on the regular 3D state lattice.

In the first set of experiments, each planner was provided with the full 3D map of an office-like environment beforehand, and its task was to search for a path from a fixed start state to randomly selected goal states. We did tests in varying environments to compare both planners.

The second set of experiments was similar to the first set except that the planner did not have prior knowledge of the environment. The task of each planner was to navigate the a-priori unknown environment and guide the MAV from a fixed start state to a fixed goal state as fast as possible. We ran our octree-based path planner in large environments such as a two-floor office environment which was prohibitively too large for the planner based on a regular 3D state lattice to find an optimal path.

A. Simulation Experiments in Fully Known Environments

These simulation experiments were carried out in an office-like environment ($20m \times 20m \times 4m$ with $0.25m$ maximum resolution) as shown in Figure 4a. In order to navigate a large environment in real-time, we chose a maximum resolution of $0.25m$.

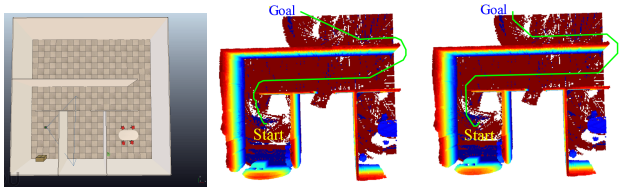
The full 3D map of the environment was provided to the MAV beforehand. The fixed start state is located at the bottom left corner of Figure 4a, and goal states were randomly generated. We did an experiment for each planner with 50 different goals. Both our octree-state-lattice-based path planner and the regular-state-lattice-based path planner were provided with the same 3D map, the same graph search method, and the same set of high-resolution motion primitives. The only difference between both planners was the space discretization representation. We compared both planners using the path planning time and path optimality metrics. Here, we assume that the path obtained from the regular-state-lattice-based path planner is fully optimal, and thus, determines how optimal the path obtained from our path planner is. The statistical results are recorded in Table I. The results show that in comparison with the regular-state-lattice-based path planner, our octree-state-lattice-based path planner sped up the path planning process by nearly 24 times at the cost of a 11% decrease in path optimality.

TABLE I: Statistical Results from Simulation Experiments.

	Our Path Planner	**Baseline Path Planner**
Map Update Time[1](s)	0.0991	0.0185
Graph Search Time[2](s)	0.299	10.1803
Heuristics Time[3](s)	0.0288	0.0288
Total Time (s)	0.428	10.23
Total Path Length (m)	1108.32	1009.21
Optimality Ratio	1.11	1
Memory Usage (Gb)	0.474	1.39

[1] the time taken to update obstacle information and construct graph
[2] the time taken to run A* algorithm on the given graph
[3] the time taken to compute heuristics

An example of paths returned by both path planners is shown in Figure 4. As we can see from this example, both paths are very similar.



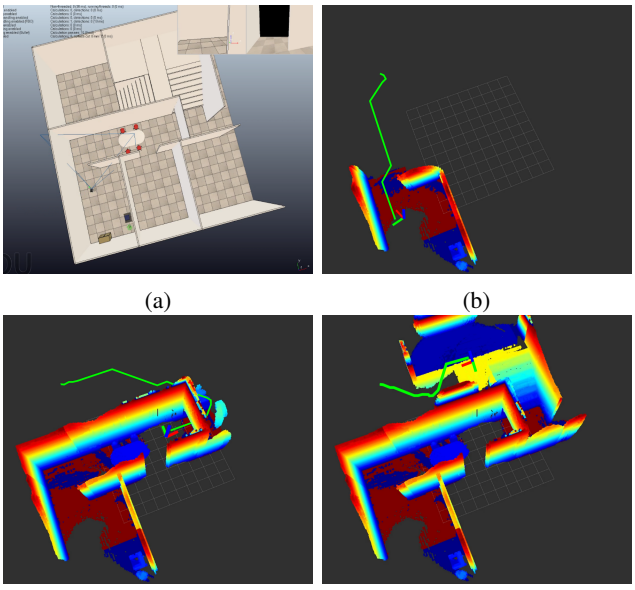
(a) (b) (c)

Fig. 4: Simulated environment and path quality comparison. In the colored 3D map, the color intensity is proportional to the value of height/elevation. The green line represents the final path. (a) shows the simulated environment. (b) shows the path obtained from the regular-state-lattice-based path planner and (c) shows the path obtained from our octree-state-lattice-based path planner.

B. Simulation Experiments in Entirely Unknown Environments

In these experiments, the MAV does not have prior knowledge about the environment. The MAV had to consistently

plan and re-plan optimal paths to the goal while exploring the environment. The test environment we used in this section is a double-floor office ($20m \times 20m \times 8m$ with $0.25m$ maximum resolution) as shown in Figure 7c, which was prohibitively too large for the regular-state-lattice-based path planner to compute an optimal path. We recorded the timing data as the MAV moved from the bottom left corner to the top left corner. Snapshots of the 3D navigation are shown in Figure 5. Throughout the experiments, our path planner was able to return optimal trajectories within 1 second. The MAV took 105 seconds in total to reach the goal position.



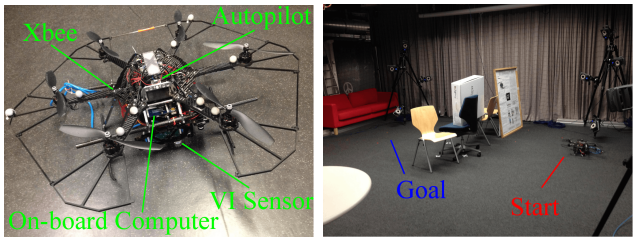
(a) (b) (c) (d)

Fig. 5: Autonomous 3D navigation in an entirely unknown environment. The coordinate frame represents the current position of the MAV. The green line represents the current planned path to the goal. (a) shows the current state of the V-REP simulator and (b-d) visualize the mapping and planning processes. (a) and (b) correspond to the same time instance, and show that the MAV is moving from the room center in the bottom right corner towards the door opening. (c) shows the MAV moving towards the stairs, maintaining a much larger 3D environment map, and constantly replanning a path to the goal in 3D space. (d) shows that the MAV has reached the second floor of the building, and continues to plan a path to the goal.

V. Field Trials

In this section, we first present the MAV platform used for the field trials, and subsequently, show the results in an indoor lab environment. We use an AscTec Firefly platform that is equipped with an on-board computer with a Intel Core i7-3517UE dual-core 1.7 GHz processor and 4 GB DDR3 memory. For the exteroceptive sensor, we used a visual-inertial (VI) sensor [16] from Skybotix AG. The VI sensor includes an FPGA to do dense disparity estimation in real-time; dense disparity estimation is computationally expensive on CPUs. The disparity images are transmitted to the on-board computer for further processing. The power supply is provided by an on-board battery which can support a flight time of about 10 minutes. The Asctec Firefly platform including the VI sensor is $60.5cm \times 66.5cm \times 16.5cm$ in size and 1.3 kg in weight. Obstacles are present in the room and the MAV does not have prior knowledge of these obstacles. All computational processes which include mapping, planning, and path following were performed on the on-board computer. We set the maximum usable range of the sensor to be $3.5m$.

Figure 6 shows the MAV and our indoor test environment. The goal state was located behind the row of chairs, and thus, unobservable from the start state. The task of the AscTec Firefly was to navigate safely in this a-priori unknown environment and reach the goal state autonomously. For the sake of safety, we restricted the flight height to be between $1-2m$ and used an external motion capture system to obtain measurements of the MAV's pose. The subfigures in Figure 7 show different states of the path planning process; in each subfigure, we show the current map, the image from the left camera, and the disparity map generated by the VI sensor.



(a) (b)

Fig. 6: (a) shows the robot platform and the on-board hardware. (b) shows the environment setup, the start position and the goal position.

The MAV was able to perceive the environment and incrementally build a 3D map in real-time. At the same time, the octree-state-lattice-based path planner was constantly able to provide near-optimal trajectories which were collision-free and led the MAV to the goal state. Throughout the whole process, our path planner took less than one second to find a near-optimal path. The CPU load was less than 30 percent during the experiment and the overall amount of memory used was 400 MB.

VI. Conclusions

We showed how to use an octree data structure, a state lattice, and multi-resolution motion primitives to search for an optimal path in large and complex 3D environments and in real-time. The experimental results obtained from the simulation experiments demonstrate that our octree-state-lattice-based path planner is able to find near-optimal paths in a very short time. The simulation experiment results also prove that our algorithm is able to guide the MAV to the goal safely and quickly in entirely unknown environments. The field trial proves that our algorithm can achieve real-time

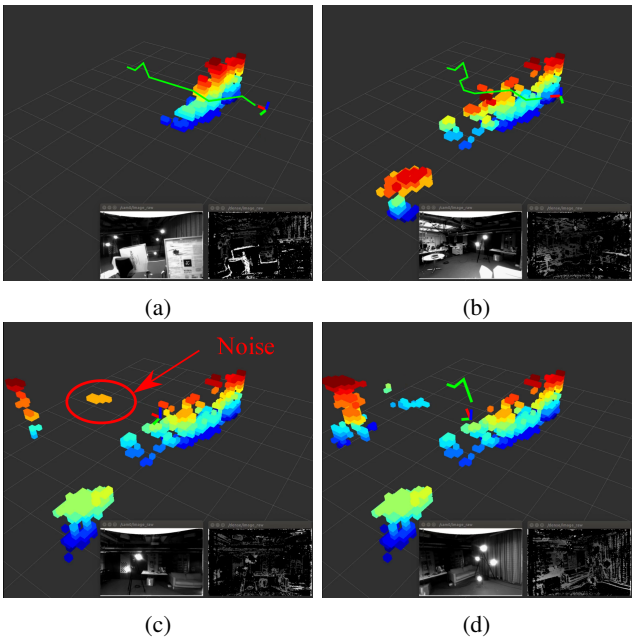


Fig. 7: An example of a field trial. The green line represents the currently planned path. The coordinate frame represents the current pose of the MAV. (a) The MAV took off and observed the obstacles in front. (b) The MAV planned a path around the obstacles. (c) The path planner could not find a path to the goal because of noisy map. (d) The MAV had corrected the map and re-planned a new path to the goal.

on-board 3D navigation for the MAV in entirely unknown environments. The field trial also shows that our octree-based path planning algorithm is highly efficient and can be run in real-time with other computationally expensive algorithms on-board.

There are many avenues of future work that can improve the overall performance. Currently, the tiled octree-based map built by the mapping module and the octree data structure used by the planner for 3D space discretization are separately maintained. In fact, there is no significant difference between the two. It is possible to only use the tiled octree-based map to do both mapping and planning. This modification can save a significant amount of memory. The octree-based path planning algorithm speeds up the path search process but at the expense of path optimality. The optimization of the resulted path remains as one of the most challenging areas for future work.

We look at improvements that are not closely related to our work but can significantly enhance 3D navigation capabilities for MAVs. One improvement would be to make the 3D map more accurate and precise by estimating stereo disparities with subpixel accuracy. Currently, the 3D map is rather noisy and inaccurate as we currently estimate integer stereo disparities, and this might result in unexpected behaviors when applying our 3D navigation algorithm in the real world. Also, the reliance on an external pose source is a limitation that constrains the flight to the lab environment. Accurate on-board state estimation will enable the MAVs to navigate 3D environments without any dependence on external resources.

References

[1] Mihail Pivtoraiko and Alonzo Kelly, Efficient Constrained Path Planning via Search in State Lattices. *The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Sept. 2005.*

[2] D. Mellinger and V. Kumar, Minimum snap trajectory generation and control for quadrotors. *IEEE International Conference on Robotics and Automation (ICRA), pp.2520-2525, 2011.*

[3] Joseph Carsten, Dave Ferguson, and Anthony Stentz, 3D Field D*: Improved Path Planning and Replanning in Three Dimensions. *International Conference on Intelligent Robots and Systems (IROS), pp.3381-3386, Oct. 1995*

[4] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, Anytime dynamic A*: An anytime, replanning algorithm. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), June. 2005.*

[5] Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys, Autonomous Visual Mapping and Exploration With a Micro Aerial Vehicle. *Journal of Field Robotics (JFR), 31(4):654-675, 2014.*

[6] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys, Autonomous Obstacle Avoidance and Maneuvering on a Vision-Guided MAV Using On-Board Processing. *International Conference on Robotics and Automation (ICRA), pp.2472-2477, May. 2011.*

[7] Alex Yahja, Anthony Stenz, Sanjiv Singh, and Barry L.Brumitt, Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environment. *International Conference on Robotics and Automation (ICRA), vol.1 pp.650-655, May. 1998.*

[8] Kitamura, Y. ; ATR Commun. Syst. Res. Labs., Kyoto, Japan ; Tanaka, T. ; Kishino, F. ; Yachida, M., 3-D path planning in a dynamic environment using an octree and an artificial potential field. *International Conference on Intelligent Robots and Systems (IROS), vol.2 pp.474-481, Aug. 1995*

[9] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev, Path Planning for Non-Circular Micro Aerial Vehicles in Constrained Environments. *International Conference on Robotics and Automation (ICRA), pp.3933-3940, May. 2013.*

[10] Maxim Likhachev and Dave Ferguson, Planning Long Dynamically-Feasible Maneuvers for Autonomous Vehicles. *The International Journal of Robotics Research, pp.933-945, Aug. 2009.*

[11] Mihail Pivtoraiko and Alonzo Kelly, Differentially Constrained Motion Replanning Using State Lattices with Graduated Fidelity. *International Conference on Intelligent Robots and Systems (IROS), pp.22-26, Sept. 2008.*

[12] Mihail Pivtoraiko, Daniel Mellinger and Vijay Kumar, Incremental Micro-UAV Motion Replanning for Exploring Unknown Environments. *International Conference on Robotics and Automation (ICRA), pp.2452-2458, May. 2013.*

[13] Hanan Samet, Neighbor Finding in Images Represented by Octrees. *Computer Graphics and Image Processing, vol.46 issue.3 pp.367-386, June. 1989.*

[14] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel, Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments. *The International Journal of Robotics Research, vol.29 no.5 pp.485-501, April. 2010.*

[15] E. Rohmer, S. P. N. Singh, and M. Freese, V-REP: a Versatile and Scalable Robot Simulation Framework. *Proc. of The International Conference on Intelligent Robots and Systems (IROS), pp.1321-1326, Nov. 2013*

[16] Janosch Nikolic, Joern Rehder, Michael Burri, Pascal Gohl, Stefan Leutenegger, Paul T. Furgale and Roland Siegwart, A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM. *International Conference on Robotics and Automation (ICRA), pp.431-437, May. 2014.*