# Path Planning for Non-Circular Micro Aerial Vehicles in Constrained Environments

Brian MacAllister[‡], Jonathan Butzke[‡], Alex Kushleyev[†], Harsh Pandey[‡], Maxim Likhachev[‡]

***Abstract*— Operating micro aerial vehicles (MAVs) outside of the bounds of a rigidly controlled lab environment, specifically one that is unstructured and contains unknown obstacles, poses a number of challenges. One of these challenges is that of quickly determining an optimal (or nearly so) path from the MAVs current position to a designated goal state. Past work in this area using full-size unmanned aerial vehicles (UAVs) has predominantly been performed in benign environments. However, due to their small size, MAVs are capable of operating in indoor environments which are more cluttered. This requires planners to account for the vehicle heading in addition to its spatial position in order to successfully navigate. In addition, due to the short flight times of MAVs along with the inherent hazards of operating in close proximity to obstacles, we desire the trajectories to be as cost-optimal as possible. Our approach uses an anytime planner based on A$^\star$ that performs a graph search on a four-dimensional (4-D) (x,y,z,heading) lattice. This allows for the generation of close-to-optimal trajectories based on a set of precomputed motion primitives along with the capability to provide trajectories in real-time allowing for on-the-fly re-planning as new sensor data is received. We also account for arbitrary vehicle shapes, permitting the use of a non-circular footprint during the planning process. By not using the overly conservative circumscribed circle for collision checking, we are capable of successfully finding optimal paths through cluttered environments including those with narrow hallways. Analytically, we show that our planner provides bounds on the sub-optimality of the solution it finds. Experimentally, we show that the planner can operate in real-time in both a simulated and real-world cluttered environments.**

## I. Introduction

Within the past decade micro aerial vehicles (MAVs), such as quadrotors (Fig. 1), have become an affordable and commonly used research platform. With improved on-board sensing and processing power, they are capable of exploring terrain that would otherwise be unreachable by a ground vehicle. Our efforts are focused on one of the many challenges of operating MAVs in cluttered environments - path planning.

Autonomous path planning and navigation has its own series of challenges that must be addressed. The first challenge is due to the three-dimensional (3-D) nature of the world. A planner must not only plan to go around obstacles, but must anticipate having to go over or under them as well in order to find the optimal trajectory. The second challenge is due to the fact that the MAV often does not initially possess a full map of the environment and must re-plan as new obstacles are detected or as the world differs from its prior information. This leads to a requirement for real-time re-planning, as opposed to the computation of a single plan.

Fig. 1: Photo of a quadrotor used for testing. The long pole extending out to the left holds a camera at the end.

Another aspect of planning to consider is how the planner accounts for the shape of the vehicle. For MAVs that are circular (or nearly so) planning for a single point to represent the vehicle pose is both safe and arguably less complex than using a full body pose. However, this approach fails to be complete when the footprint of the robot is non-circular. For example, if the footprint of the robot in question was long and narrow, using the outer circumscribed circle for collision checking could be overly conservative, disallowing trajectories down narrow hallways that the robot could physically fit through. Our planner alleviates this issue by planning in the three spatial dimensions $x, y, z$ and in orientation about the vertical axis $\theta$ while performing collision checks on the actual MAV's footprint. By taking the footprint of the MAV

(a) Overhead (b) Actual Quadrotor

Fig. 2: Planning with orientation. By taking into account the asymmetric footprint, the planner can reason about the camera boom and avoid colliding with obstacles when moving in and back out of this position.

This work was partially sponsored by the DARPA grant D11AP00275 and TATRC contract W81XWH-07-C-0043. Jonathan Butzke was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program

[‡] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA {bmaca, jbutzke, hpandey}@andrew.cmu.edu, maxim@cs.cmu.edu

[†] GRASP Lab, University of Pennsylvania, Philadelphia, PA, USA akushley@seas.upenn.edu

into account, the planner is capable of getting closer to the obstacles, as shown in Fig. 2.

For our experiments, we have built a small quadrotor (Fig. 1) with on-board sensing and processing that is capable of flying through and mapping indoor environments autonomously. We have also developed a motion planner that is capable of returning close-to-optimal trajectories in real-time for the given footprint of the MAV and have tested the performance of this algorithm on the quadrotor navigating in a partially known environment.

This paper begins with a discussion of related work in Section II, followed by the algorithmic details of the planner in Section III. We then discuss our results in both simulation, in Section IV-B, and on an actual MAV, in Section IV-C.

## II. Related Work

A wide variety of approaches to planning feasible trajectories around obstacles in deterministic environments for MAVs have been proposed. Common approaches involve using potential fields or navigation functions to generate a gradient followed by performing a gradient ascent (or descent), constructing a graph covering the state space followed by searching the graph for a feasible path, or building a path from a subset of randomly sampled states

Navigation functions generate a gradient that directs the MAV away from obstacles and towards the goal. By carefully constructing this gradient, local minima can be eliminated, ensuring convergence to a goal state. For speed considerations these can be implemented as hierarchical planners [1] with both a local and global planning component. However, since these methods do not evaluate the orientation of the MAV while generating the gradient field for the global planner they are restricted to using the circumscribed circle of the vehicle as their footprint. This limitation means that environments requiring traversal of narrow passages cannot be completed. In addition, having two separate planners of different dimensionality can lead to inconsistencies between the two resulting in sub-optimal trajectories or failing to find a solution when the two disagree.

Sampling-based approaches such as Rapidly-exploring Random Tree (RRT) algorithms [2] and Probabilistic Road Maps (PRM) [3] are frequently used for planning in high-dimensional spaces. These planners sample a subset of the configuration space and generate a trajectory from these sampled points. Most of these approaches require a path smoothing step after the planner has generated the original path in order to get a usable, kinodynamically feasible trajectory. For example, one of the best currently available implementations provides several short-cutting options to improve generated trajectories [4]. There have been efforts applicable to MAV navigation that have attempted to avoid this extra processing by constructing the search graph or tree using smooth curves or motion primitives [5][6]. However, due to the random nature of these algorithms sampling-based approaches tend to produce paths that are highly sub-optimal. This is especially pronounced for MAVs operating in constrained environments where post-processing steps cannot change the homotopic class of the paths. We show this in the experimental analysis of this paper.

A recently introduced hybrid planning methodology combines the speeds of the sampling-based planners with cost minimization. RRT$^*$ and PRM$^*$ [7] provide guarantees on optimality in addition to the guarantees on completeness from their base algorithms both in the limit of the number of samples. We show that for cluttered environments when planning with an MAV footprint our algorithm provides lower cost solutions for a given planning time.

Path planning approaches based on heuristic graph searches such as A$^\star$ have been popular for navigation but have not been as widely used for planning MAV trajectories due to the relatively high dimensionality of the planning problem. An early method for generating collision free trajectories in cluttered environments was to segment the free space into intersecting spheres, then plan through this network of spheres using A$^\star$ [8]. This method had the advantage of not requiring any additional collision checking during the planning stage, and due to the reduced size of the search graph was capable of quick planning times. The drawback to this approach was that the complete map was needed beforehand in order to perform the segmentation, and by requiring a spherical footprint, valid trajectories were pruned from the search graph.

To address the speed concerns of planning in a high-dimensional space, two-phase planners first generate a coarse global plan over the entire domain followed by higher fidelity local plans in the vicinity of the MAV [9]. While these methods help reduce planning times, the large discretization of the global planner prevents them from taking an accurate footprint of the MAV into account. While such a footprint could be applied at the local planner level, it is necessarily constrained by the overall trajectory of the global planner, and thus would not be able to take advantage of narrow passages. Additionally, as with the hierarchical navigation functions previously discussed, these planners are prone to producing suboptimal solutions in higher fidelity and may even fail.

Our approach is a search-based algorithm that combats the computational complexity issues using the following three components:

- an anytime and incremental version of A$^\star$
- an informative heuristic that is very fast to compute
- a lattice graph representation that compactly models the planning problem

## III. Search-Based Path Planning for a MAV

Our approach to path planning for non-circular vehicles is search-based. We make use of what is called a state lattice [10] combined with the incremental and anytime performance of a variant of A$^\star$ to plan feasible trajectories. The planner also makes use of an informative and well optimized three-dimensional $(x, y, z)$ breadth first search (3-D BFS) heuristic to come up with solutions in significantly less time.

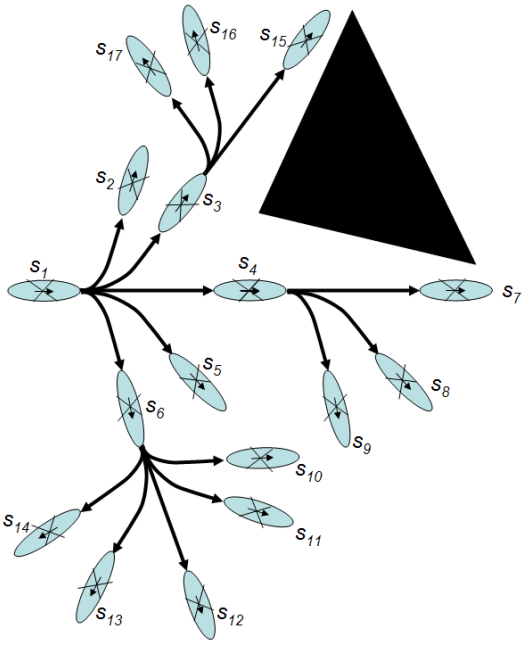
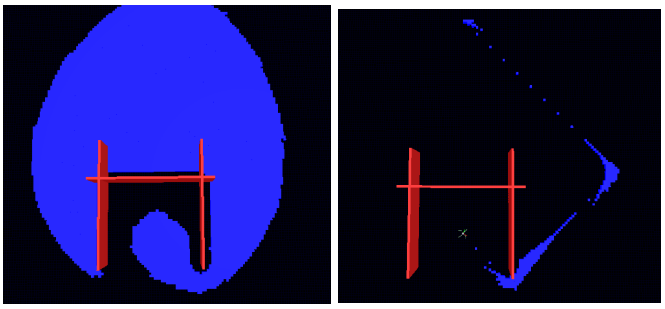



Fig. 3: Simple 2-D example of motion primitives forming transitions between states inside a state lattice.

### A. State Lattice

A state lattice is a type of discretized search space that adheres to nonholonomic motion planning constraints, such as minimum turning radius, for graph search. In our approach we plan in a four-dimensional state-space, in which each state is a combination of its position in Euclidean space and its yaw angle about the global z-axis. Transitions between states are obtained using a combination of motion primitives. Motion primitives can be thought of as short, dynamically feasible path segments that are used as building blocks to form paths of varying sizes and curves. Any motion primitive can come before or after another as long as neither results in a collision. Every motion primitive has a precomputed cost of traversal. In addition, each motion primitive has a user-defined weight. The product of traversal cost and weight forms the motion primitive cost. We use weights to express a user-defined preference for certain motions over others, e.g., placing a higher penalty on the backwards motion to preferentially move forwards (and thus keep our sensors pointing ahead of us). A sketch of how motion primitives form transitions between states in a two-dimensional plane can be seen in Fig. 3.

Motion primitives play a significant role in planner completeness, path quality, and planning time. If a pair of start and goal points inside the state space required a feasible motion that could not be produced from a combination of the set of available motion primitives, the planner would fail to find a path between them. For example, the planner would fail to find a path between two points if the solution required a motion that rotated the MAV in-place and the given set of motion primitives did not include any rotation-in-place motions. In general, the more motion primitives that are available to the planner, the more flexibility it has in finding paths. It is important to consider that there is a trade-off in the number of motion primitives that are used in the search and the planning time, since the addition of each motion primitive for a given state, increases the branching factor of each state by one.



(a) (b)

Fig. 4: Examples of expansions using (a) Euclidean distance and (b) Breadth-first search as the heuristic. Goal is near the top and the start is inside the red (light gray) structure.

### B. Graph Search

The motion planner makes use of a graph search algorithm to generate trajectories using the previously described lattice. The graph search used is a variant of A$^\star$ called Anytime Dynamic A$^\star$ (AD$^\star$) [11]. Like A$^\star$, AD$^\star$ makes use of an informative heuristic to reduce planning times, and given sufficient time will find the optimal solution with respect to the graph representing the original planning problem. It is also both an anytime and incremental variant of A$^\star$. As an anytime variant it quickly computes a solution which has an $\epsilon$ sub-optimal upper bound on solution cost: the cost of the found solution is no more than $\epsilon$ times the cost of an optimal path in the graph. It then improves the solution together with its sub-optimally bounded cost until either the planning time runs out or a provably optimal solution is found. As an incremental variant, AD$^\star$ can repair a previous solution when changes in edge costs occur, such as when new obstacles are detected, which can be, in terms of computation, significantly less expensive than computing a new solution from scratch. To minimize the amount of re-computation required during planning iterations, the planner works backward searching out from the goal to the current robot pose. In this way only the final nodes of the graph are changed as the robot moves or the map is updated based on the latest sensor input.

### C. Heuristic

The efficiency of a heuristic search algorithm such as A$^\star$ and in particular anytime variants such as AD$^\star$ depends heavily on the quality of the heuristic. A$^\star$ without a heuristic performs the same number of expansions as a Dijkstra search (terminated as soon as the goal state is reached), but an informative heuristic function can greatly reduce the number of expansions by guiding the search in a promising direction. The heuristic value must be an underestimate of the distance from the robot's current pose to any other state in question

(the heuristic estimates distances from the robot pose rather than to the goal pose because the search itself is done backwards). A common approach in path planning is to use the Euclidean distance as the heuristic for a state. However, this can be highly uninformative since it does not take obstacles into account, potentially resulting in local minima. Another alternative is to use a lower dimensional Dijkstra's search to calculate the cost from the robot pose to all states. This approach eliminates the local minima problem, but is substantially slower to compute, especially since it needs to be done every time the MAV moves or the map is updated. If we make the assumption (just for our heuristic) that all edge costs are uniform, we can instead shift to a 3-D BFS for our heuristic. This simplification allows us to use a first-in first-out (FIFO) queue instead of a heap data structure required for a Dijkstra search which significantly improves performance. Prior to performing the BFS we expand all of the obstacles by the inscribed radius of the MAV footprint. This eliminates the need to evaluate states that the robot is unable to occupy in any orientation, thus saving time without compromising completeness.

For a simple case, shown in Fig. 4, where the search must find a solution around a wall, the Euclidean heuristic expands a large number of states near the wall, while the 3-D BFS heuristic leads the search to expand states around the wall, leading to less expansions overall. In contrast, a less informative Euclidean distance heuristic requires the search to expand a large ring of states around the goal before it finds a solution. As such the A$^\star$ search is able to find a solution in significantly less time using the more informative BFS heuristic than if it were to use a less informative Euclidean distance heuristic.

A downside to a more informative heuristic typically is longer computation times. A 3-D BFS for large environments can be prohibitively expensive with representative times being 1.96 seconds to compute a 3-D grid with 7.5 million states. Upon examination, we can find several areas of optimization in implementing a 3-D BFS for our domain. One area of optimization is at the data structure level. A typical BFS implementation requires a distance grid which stores the distance of each cell from the origin. In a typical 3-D BFS, the distance grid may be implemented as a 3-D array - thus to access a given cell you must supply its x, y, and z coordinates. This creates some overhead, as each access to the 3-D array requires address computation. We optimize in this area by storing all the coordinates as 1-D coordinates, which allows us to use a 1-D array for the distance grid, thus removing the overhead. Note that given any 3-D coordinates (x, y, and z) and grid dimensions (width, height, and length), we can convert the 3-D coordinates to 1-D coordinates by the following formula: x + (y * width) + (z * width * height).

Then, we can change the x dimension by adding or subtracting 1, we can change the y dimension by adding or subtracting width, and we can change the z dimension by adding or subtracting (width*height). This leads to the second area of optimization, which is loop unrolling. In a typical BFS implementation, there is an expansion loop which examines each successor of the current cell and decides whether or not to add it to the FIFO queue. Loop unrolling is a common optimization technique which removes the overhead of having a loop. It is a simple matter to get the coordinates of all 26 successors of a given cell (represented by a 1-D coordinate), as it is easy to change any of its dimensions by the computations described above.

One final area of optimization that we implement is eliminating bounds checking by doing some preprocessing. It can be expensive to make sure each successor cell falls within the boundaries of the given grid. Bounds checking can be eliminated entirely by adding another layer around the perimeter of the grid (thereby increasing the number of cells along each dimension by 2) and marking all of these cells to be non-traversable. Thus whenever a successor beyond the boundary is expanded, it will be in our new layer of cells added and will be marked as non-traversable, and thus will be safely discarded. These optimization techniques each lead to significant increases in performance. Our results show that we are now able to evaluate a grid of 7.5 million states in 0.13 seconds instead of the 1.96 seconds previously. The additional memory requirements for our optimized BFS are proportional to the size of the grid it searches through. For a grid of size $N^3$ where $N$ is the number of cells along a particular dimension, the optimized search would build a larger grid of size $(N+2)^3$.

## IV. Experimental Results

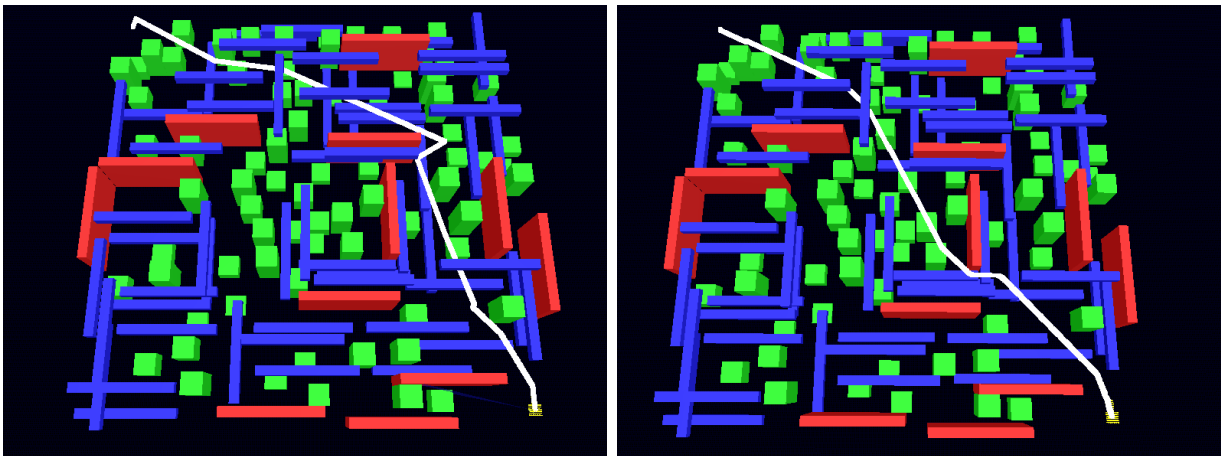
In this section we evaluate the performance of our planner and compare it against RRT- and RRT$^*$-based planners.

### A. Implementation

For both our simulations and for our actual robot experiments we discretize the state space into 10cm cells for $x, y$, and $z$ and into $22.5^\circ$ increments for $\theta$. Also, our planner was given a simple set of motion primitives consisting of turning in place in both directions, moving up or down in altitude, and moving forward a short or long distance, and moving backwards. The backward motion was assigned an additional penalty so that the planner would preferentially keep the front of the MAV aligned with the direction of travel. This provided us with a set of 112 different motion primitives per grid cell.

We use a 3-D footprint of the MAV for collision checking for any given $(x, y, z, \theta)$ of the MAV. Each motion primitive has a set of finely spaced intermediate 4-D poses and each of these must be checked to ensure that a given motion is valid between two states. For each motion primitive, we optimize collision checking for the footprint by precomputing the set of all cells that correspond to the 3-D footprint being moved along this motion primitive. Doing so prevents the collision checker from having to re-compute this set during planning and prevents it from checking the same cell twice for a given motion primitive between two poses. This type of optimization was also used in [12].

The cost of a transition between two states connected by a single motion primitive is set to the product of the motion



(a) RRT* trajectory. (b) AD* trajectory.

Fig. 5: Trajectory (white line) generated by RRT*(a) and AD*(b). Red (gray) walls extend floor-to-ceiling, green (light gray) boxes extend from the floor to a random height. Blue (dark gray) beams are placed at varying heights and do not intersect the floor or ceiling. Therefore, whenever the path goes over an obstacle it means that the quadrotor flies above the obstacle.

primitive cost, the maximum cost of the cells it traverses over, and the deviation from the desired height. Cell costs for our approach were either one for unoccupied cells or infinity for occupied or padded cells. We prefer that the MAV follows a path at a desired height. Therefore, every vertical motion that deviates from this height is penalized by the difference. Such a cost is thresholded such that plans that do require the helicopter to deviate from the desired height, such as moving above a high wall, are not too costly.

### B. Simulation

*1) Setup:* We simulated our approach by randomly generating 100 maps at each of two different sizes: ($250 \times 250 \times 30$) cells and ($500 \times 500 \times 30$) cells. These maps were then populated with randomly placed obstacles such that approximately 20 percent of the volume of each map was occupied, e.g., Fig. 5. The obstacles generated for those maps were impassable floor to ceiling walls, boxes projecting from the floor up to a random height, and fixed width beams mounted at random elevations. The sizes of these obstacles were scaled to maintain a constant ratio with respect to the map dimensions.

Two kinds of experiments were performed for each of the planners on each of the maps. In the first experiment each planner is given the complete map of the environment and is tasked to compute a single plan from a start position in the bottom right corner of the map to a goal position in the upper left corner of the map. Each planner was given 10 seconds to come up with a plan. If the planner was unable to find a plan within 10 seconds, it was counted as a failure.

The second experiment was similar to the first except the map was not provided beforehand. In this way the MAV discovered obstacles through its mounted horizontal and vertical scanning LIDAR sensors within a 30 cell distance as it moved towards the goal, forcing it to re-plan frequently along the way. For this experiment data was logged as the quadrotor moved from its start position in one corner to the goal position in the diagonally opposite corner. To reduce variance in simulation results, the same map and pose information that was used by our planner at each search episode was also provided to the other planners. In cases where a planner was unable to find a solution within a second for a specific episode, it was counted as a failure. It should be noted that since RRT and RRT* are not dynamic planners, they would start from scratch every time the MAV observes new obstacles or moves.

All three planners used the same cost function and collision checking approach presented in Section IV-A. Additionally, the RRT* planner was also given access to the 3-D BFS heuristic. This heuristic was used to prune the states that cannot improve the current solution. For RRT and RRT* collisions were checked between sampled states every 5cm of translation and for every $22.5^\circ$ of rotation. This is sparse considering that intermediate positions within motion primitives in our approach were more finely discretized. We also post-processed solutions produced by both randomized methods with a short-cutter between sampled vertices to improve path quality in terms of cost.

*2) Results:* Results for the first experiment can be seen in Table I. The values shown are the average values for all planning episodes for each map size, along with the percentage of plans that were failures. Our approach found an initial solution in a comparable time to both randomized methods but was able to find one with lower cost on average. Our approach was also able to provide the best solution on average at the end of the 10 second planning time. Between different sized maps solution quality and path length nearly double for all methods, but the initial planning time for our approach did not grow as significantly compared to the other methods. Fig. 5 shows an example of the generated trajectories from RRT* and our approach in which both planners were given the same start/goal pair and map.

TABLE I: Results from planning with full map

| Map | Planner | Initial Plan Time | | First Solution Cost | | Final Solution Cost | | Final Path Length | | Percent |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg. (s) | Std. Dev. | Avg. | Std. Dev. | Avg. | Std. Dev. | Avg.(m) | Std. Dev. (m) | Failure |
| Small | AD* | 0.17 | 0.73 | 44568 | 6844 | 32301 | 5220 | 43.77 | 5.74 | 0 |
| | RRT* | 1.09 | 1.24 | 58880 | 12894 | 40590 | 4985 | 54.46 | 8.64 | 2 |
| | RRT | 0.26 | 0.34 | 89129 | 25101 | n/a | n/a | 87.35 | 19.76 | 0 |
| Large | AD* | 0.16 | 0.50 | 81659 | 14485 | 61696 | 6026 | 88.14 | 7.44 | 0 |
| | RRT* | 4.28 | 2.74 | 99531 | 15736 | 85366 | 12421 | 118.08 | 15.50 | 28 |
| | RRT | 0.66 | 0.60 | 144045 | 37098 | n/a | n/a | 146.87 | 23.20 | 2 |

TABLE II: Results from planning with unknown map

| Map | Planner | Initial Plan Time | | First Solution Cost | | Final Solution Cost | | Final Path Length | | Percent |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg. (s) | Std. Dev. | Avg. | Std. Dev. | Avg. | Std. Dev. | Avg.(m) | Std. Dev. (m) | Failure |
| Small | AD* | 0.05 | 0.08 | 21832 | 10116 | 17621 | 8915 | 24.79 | 12.05 | 2 |
| | RRT* | 0.25 | 0.25 | 34091 | 16865 | 25827 | 13158 | 33.56 | 15.38 | 33 |
| | RRT | 0.09 | 0.15 | 41349 | 24033 | n/a | n/a | 41.16 | 20.41 | 16 |
| Large | AD* | 0.08 | 0.10 | 38137 | 16800 | 34497 | 15712 | 49.69 | 23.52 | 4 |
| | RRT* | 0.38 | 0.24 | 56087 | 26754 | 44914 | 20919 | 59.49 | 25.23 | 45 |
| | RRT | 0.17 | 0.21 | 67319 | 39420 | n/a | n/a | 69.06 | 31.60 | 24 |

Our approach was able to provide initial plans slightly faster than the randomized methods, which can be attributed to the difficulties randomized methods have with cluttered environments and narrow passageways. This is due to the complete randomness of choosing states without taking into consideration the surrounding environment or restrictions of the system. Cluttering an environment reduces the set of all valid paths and as the clutter increases, the ratio between the number of valid paths over hypothetical paths decreases.

The randomized methods must check all cells of a footprint at a given angle every 5cm along a trajectory between two states, which results in re-checking cells that have already been checked for collision. Because our approach uses precomputed motion primitives, we never have to recheck a cell more than once for collision as the cells that a motion would collide with are predetermined.

Results for the second experiment can be seen in Table II. The averages reported are over all valid planning episodes, which do not include failed plans. The percentage of failed plans is calculated over all planning episodes. As with experiment 1, our approach has comparable planning times but significantly lower initial and final solution costs for the same reasons.

### C. Actual Robot

*1) Setup:* We performed experiments using a custom built quadrotor with on-board sensing and processing as shown in Fig. 1. For sensing the quadrotor is equipped with an IMU and two LIDAR sensors. One LIDAR sensor is mounted on top of the quadrotor to provide a horizontal scan up to a maximum range of 30m. The second LIDAR is attached to a panning servo to provide a vertical scan with a maximum range of 5m. Although not used for path-planning, the quadrotor was also equipped with a web-cam attached to a long rod extending forward approximately 90cm from the center of the chassis. On-board computations are performed on an attached Mac-mini motherboard with an Intel 2 GHz Core 2 Duo processor and 4 GB of RAM. The quadrotor uses a combination of a 2-D SLAM algorithm with the horizontal LIDAR and height estimation with the vertical LIDAR to localize itself inside indoor environments. The path planner was capped to use no more than 60% of one CPU core so that other processes required for autonomous flight could be run simultaneously with the planner. The planner was provided with a footprint of the quadrotor which contained a $66 \times 66 \times 30$cm cuboid attached to a thin $58 \times 1 \times 30$cm cuboid representing the rod. With a 10cm discretization the footprint occupied approximately 150 cells. 50cm padding was provided to ensure the quadrotor would be at a safe distance from obstacles during flight.

Fig. 6: The prior map provided to the planner.

Tests were run inside a $16 \times 12 \times 1.5$m ($160 \times 120 \times 15$ cell) indoor environment, a prior map of which is displayed in Fig. 6. This prior map included just the rough floor plan; none of the obstacles were provided to the MAV prior to the tests. To demonstrate the anytime performance of the planner,

Fig. 7: The quadrotor navigating under a newly discovered obstacle.

Fig. 8: The quadrotor at first way-point with camera sticking out of passage (circled).

obstacles were placed near locations that the quadrotor had to traverse through, as can be seen in Fig. 7. During each run the quadrotor started from the bottom left corner of the map and was expected to visit three separate way-points in a specific order.

To demonstrate that the planner was planning with the footprint dimensions described earlier the following scenario was provided. When the quadrotor arrives at the first way-point, it has to be situated in such a manner that the camera is able to see around the corner near the top left portion of the map as shown in Fig. 8. The placement of the two obstacles near the passageway combined with the 50cm padding around them prevents the quadrotor from advancing forward or turning in place. As a result it must back up about a meter before it can proceed to the next way-point.

*2) Results:* The quadrotor was run twice in the previously described test scenario, during which the planner was consistently able to provide new trajectories that prevented the quadrotor from colliding with obstacles. With the 60% CPU cap the planner was able to return solutions approximately every 2 seconds on average. A video has been provided that shows the quadrotor autonomously following plans inside the previously described indoor environment. Pictures of plans changing over the course of the quadrotor's traversal can be seen in Fig. 9a through Fig. 9f.

## V. Conclusion

Our approach for generating MAV trajectories in cluttered environments is capable of planning times comparable to those of sampling-based planners while producing lower cost solutions both in simulation and in real-world environments. The improvements are a result of using an anytime and incremental version of A$^\star$, an informative heuristic that is very fast to compute, and a lattice graph representation that compactly models the planning problem.

## References


[1] S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli, "Flying fast and low among obstacles," in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 2023 –2029.

[2] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[3] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566 –580, aug 1996.

[4] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012, to appear. [Online]. Available: http://ompl.kavrakilab.org

[5] E. Frazzoli, M. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *Robotics, IEEE Transactions on*, vol. 21, no. 6, pp. 1077 – 1091, dec. 2005.

[6] A. Neto, D. Macharet, and M. Campos, "Feasible rrt-based path planning using seventh order bezier curves," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, oct. 2010, pp. 1445 –1450.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011. [Online]. Available: http://dx.doi.org/10.1177/0278364911406761

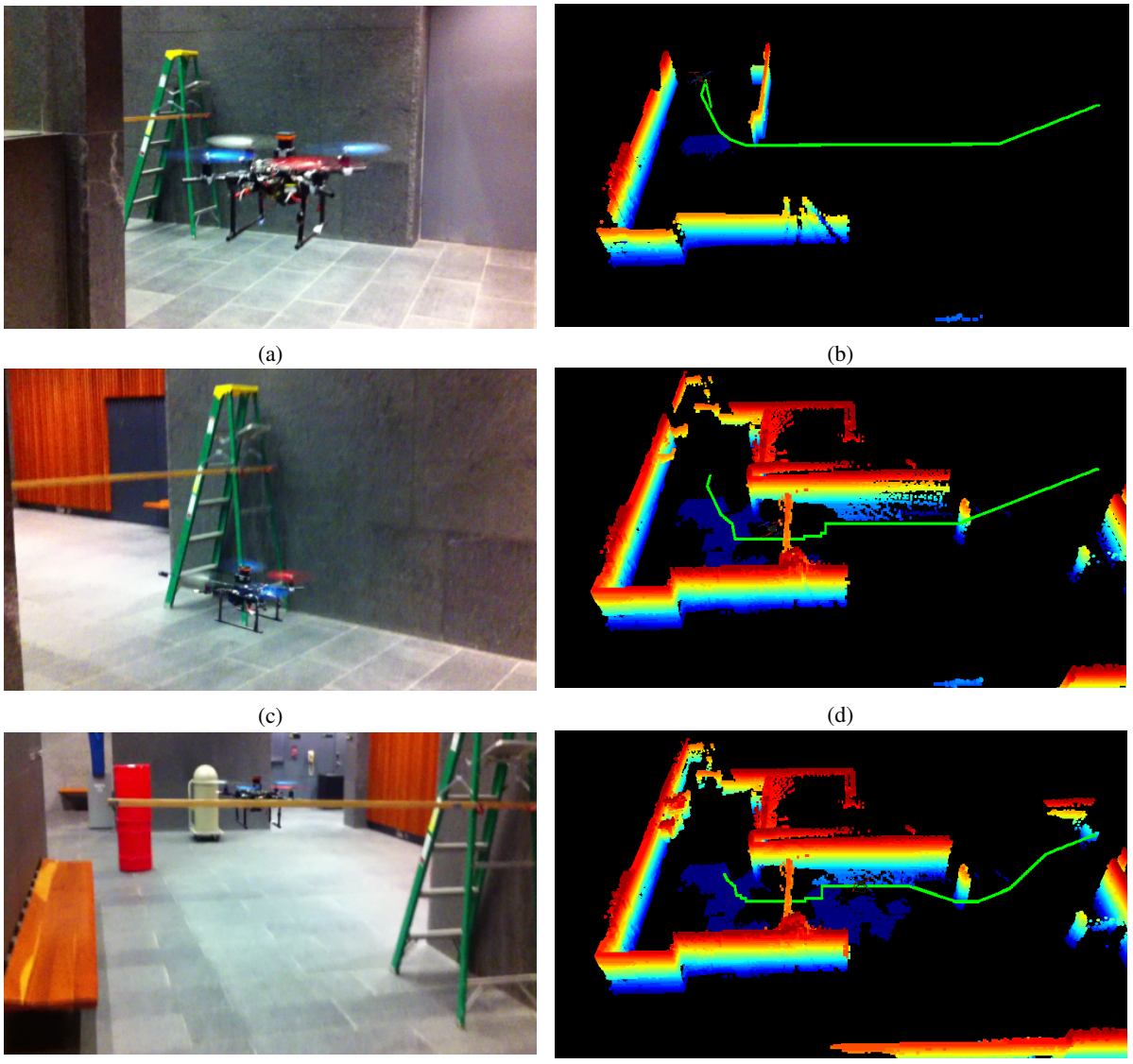
[8] N. Vandapel, J. Kuffner, and O. Amidi, "Planning 3-d path networks in unstructured environments," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, April 2005, pp. 4624 – 4629.

[9] M. Hwangbo, J. Kuffner, and T. Kanade, "Efficient two-phase 3d motion planning for small fixed-wing uavs," in *Robotics and Automation, 2007 IEEE International Conference on*, April 2007, pp. 1035 –1041.

[10] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, August 2005, pp. 3231 – 3237.

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

[12] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Rob. Res.*, vol. 28, no. 8, pp. 933–945, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1177/0278364909340445



(a) (b) (c) (d) (e) (f)

Fig. 9: (a) The quadrotor beginning its journey. (b) The initial trajectory. (c) The quadrotor has encountered a horizontal bar across its path at the desired height. (d) Updated trajectory to account for the stick. (e) Quadrotor discovers a red barrel in its path. (f) Trajectory now veers to the right to maintain a safe distance around the barrel.