Sparse Tangential Network (SPARTAN): Motion Planning for Micro Aerial Vehicles

Hugh Cover, Sanjiban Choudhury, Sebastian Scherer and Sanjiv Singh

***Abstract*— Micro aerial vehicles operating outdoors must be able to maneuver through both dense vegetation and across empty fields. Existing approaches do not exploit the nature of such an environment. We have designed an algorithm which plans rapidly through free space and is efficiently guided around obstacles. In this paper we present SPARTAN (Sparse Tangential Network) as an approach to create a sparsely connected graph across a tangential surface around obstacles. We find that SPARTAN can navigate a vehicle autonomously through an outdoor environment producing plans 172 times faster than the state of the art (RRT*). As a result SPARTAN can reliably deliver safe plans, with low latency, using the limited computational resources of a lightweight aerial vehicle.**

I. Introduction

The demand for UAVs that can carry out missions in outdoor remote environments is ever increasing. These vehicles, with little or no prior information of the area and relying on limited on-board sensing capabilities, will have to carry out high level mission plans.

Typically the robot (Fig 1) is given a goal point, and a high level mission planner generates a sequence of waypoints for the vehicle to reach. Such environments could be confined, such as under bridges or between trees, or sparse like empty fields. Flying autonomously in such unstructured outdoors environment is challenging because it implies consistently delivering safe, feasible paths in real time. Such environments tend to have clusters of objects separated by vast empty space. Standard motion planning approaches which spend as much effort on free space as in obstacle dense regions do not exploit this situation. There is a need for a light weight planner which computes only when it needs to and sits idle otherwise.

The main contribution of this paper is to present a planner which leverages the intermittent sparsity of its environment and creates plans 172 times faster than the state of the art. We present SPARTAN, (Sparse Tangential Network) which constructs a graph wrapped in a tangential surface around obstacles. The large speedups have been achieved by creating vertices during the obstacle cost update process as well as by creating careful graph construction rules to remove redundancies. Using limited onboard resources, SPARTAN has flown autonomously around trees, under bridges & powerlines and over changing terrain with missions up to 200 meters in length.

H. Cover, S. Choudhury, S. Scherer and S. Singh are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A., {hcover,sanjiban,basti,ssingh}@cmu.edu

Fig. 1. Micro Aerial Vehicle navigating autonomously outdoors

We begin by summarising in Section II the progress made in motion planning for UAVs so far and describe the specific planning problem in Section III. In Section IV we present our approach and the key components. In Section V we state the properties of the algorithm and finally in Section VI we show results for comparison with a state of the art planner.

II. Related Work

The survey by Kendoul[1] classifies most practical planning approaches for UAVs in outdoor environments, while Goerzen et al.[2] make a more broad classification. SPARTAN falls in the category of generating a roadmap for free space using a visibility graph like representation and searching over this roadmap.

The most popular approach to planning in 3D spaces is by performing a Heuristic Graph Search as summarized by Ferguson et al.[3]. This search can be performed over a generic roadmap or over regularly connected grids. Even though the latter is more popular because the graph is easier to construct, it becomes very computationally heavy for fine resolutions. Multi-resolution methods have effectively overcome this problem, for example, unmanned helicopters have been flown by Tsenkov et al.[4] and Whalley et al.[5] using a quad-tree grid representation and performing A* search. However, such methods still incur discretization errors and produce unnatural oblique paths.

Probabilistic methods allow solutions of arbitrary complexity and resolution and probabilistically converge on a solution. By sampling uniformly in free space, probabilistic roadmaps (Kavrakil et al.[6]) can be generated for graph search, or rapidly exploring random trees (LaValle[7]) for

faster solutions. However the convergence of these methods are slow, the plans found are not optimal and no performance bounds can be claimed about them. More recently, the introduction of the RRT* by Karaman et al.[8] improves the performance by bringing asymptotic optimality to sampling based planners.

Visibility graphs and their reduced forms have been analysed in detail by LaValle[9]. Despite the computational complexity, it has been used for navigation in an outdoor scenario by Wooden et al.[10]. In UAV planning, visibility graphs have been used by Hoffmann et al.[11] to generate a 2D initial guess to navigate within a polygonal environment. It also has been used to plan in a limited horizon by Kuwata et al.[12]. However, these methods projected real data into a polygonal environment and planned in a geometric space, thus making the method sensitive to sensor noise. Similar in nature to visibility methods, the tactic of staying close to obstacle and moving tangentially to them is present in reactive planning techniques (Hrabar [13]). This algorithm checks collisions within a cylindrical safety volume, and finds an escape point. The 3D Dodger approach by Scherer et al.[14] also results in trajectories lying on a tangential like safety distance from obstacles.

III. Problem

We wish to compute command plans for an UAV as the shortest path to goal while avoiding obstacles. Since the paths are to be tracked by a controller, we impose a limit on the angle between two consecutive segments of the path. Thus the planning problem is to find a solution trajectory $\sigma(t) = (x(t), u(t))$ for

$$\begin{array}{ll} minimize: & J = \int_0^{t_f} \|x(t)\| + [max(0, d_{max} - d(x(t)))]^2 dt \\ constraints: & x(0) = x_0, x(t_f) = x_f \\ & g(x(t), u(t), t_f) \leq 0 \end{array} \quad (1)$$

where d_{max} is the maximum distance upto which an obstacle cost is applicable, x_0 the initial conditions, x_f the goal point and g denotes the consecutive segment angle constraint. The specific details of the problem are:

1) Environment: Unstructured outdoors with varying clutter. Obstacle density may increase in a forest region or be sparse over empty fields. No prior is provided.
2) Sensor Range: Limited sensor range implying that obstacle information is updated on the fly.
3) Planning horizon: Limited to 3-4 times the sensor range. A local scrolling map upto this horizon is maintained and updated as the planner only navigates through a local area.
4) Planning time: It should have a frequency around 10 Hz.

An effective approach to solve such problems is to plan quickly through free space and focus efforts on critical regions near obstacles.

IV. SPARTAN

SPARTAN (Sparse Tangential Network) is a planning approach that uses a sparsely connected graph to quickly find good paths through cluttered environments. The vertices of the graph exist only nearby obstacles because that is where avoidance maneuvers must be carried out. Only edges that are useful to guide the graph search around nearby obstacles or to cut quickly across free space are included. SPARTAN works by defining a planning surface, or in 2 dimensions a line, around obstacles with some nominal clearance. As a vehicle approaches the obstacle a short and safe path can be made by taking the tangent to the circle, then following the circle until another safe tangent can be followed to leave the circle and cut quickly through free space to the next interesting region. Tangential connections can also be made between different parts of the surface around a more complex obstacle, e.g. across the entrance of a cul-de-sac. The center of narrow corridors between obstacles, known as a ridges, are also included in the planning surface which allows for paths that get closer to obstacles than the nominal clearance distance when required, e.g. to pass through a doorway.

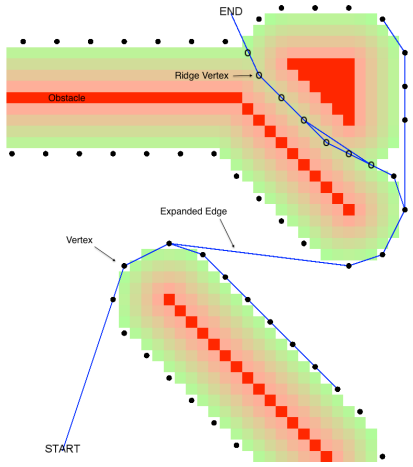

Fig. 2. A simple 2D example; obstacles are shown in bright red; distance to the closest obstacle up to the nominal clearance distance is shown as a color scale from light red to green; vertices are shown as solid circles; vertices on a ridge are shown as empty circles; edges expanded during graph search are shown as blue lines.

By cutting through free space using only tangential jumps SPARTAN can provide real-time solutions in unstructured environments with both cluttered and clear regions. A simple 2D example showing the SPARTAN approach can be seen in Fig. 2, only edges that were expanded during graph search are shown. The example shows that only a small number of possible path branches have to be searched.

A. Planning Surface

The SPARTAN approach can be thought of as a reduced visibility graph generalized for unstructured 3D environments. Obstacles are represented by a set of discrete points. We define a surface made from the visible exterior of the union of constant radius spheres drawn around each obstacle location. Moving across the surface is equivalent to moving tangentially to the current closest obstacle. Lines that are tangential to the surface at both ends are used as connections between one region of the surface and another or between

TABLE I

VARIABLE AND FUNCTION DEFINITIONS

$dist_n$	Distance of the closest occupied cell to cell n
$obst_n$	Reference to the closest occupied cell to cell n
inQ_n	Flag indicating cell n is in the open queue
$vdist_n$	Distance to the closest vertex to cell n
INSERT(O,n,k)	Insert cell n into the priority queue O using the key k
ISVALID(n)	Check if $obst_n$ points to an obstacle that currently exists
ADDVERTEX(n,V_{tan})	Adds cell n to the list of vertices V_{tan} and sets $vdist$ of surrounding cells to values consistent with n being a vertex. This is done by propagating a wavefront similar to the distance transform's lower wavefront
REMOVEVERTEX(n,V_{tan})	Removes cell n from the list of vertices V_{tan} and sets $vdist$ of surrounding cells to higher values consistent with n not being a vertex. This is done by propagating a wavefront similar to the raise wavefront

Algorithm 1 $D,V_{tan} = \text{LOWER}(s,D,V_{tan})$
Input: s = Cell being expanded, D = Distance transform, V_{tan} = Vertex list
Output: D = Distance transform, V_{tan} = Vertex list

```
1  if dist_s = ρ and vdist_s = v_res
2     ADDVERTEX(s, V_tan)
3  foreach n ∈ Adj(s)
4     d ← || n − obst_s ||^2
5     if dist_n > d
6        dist_n ← d
7        obst_n ← obst_s
8        if ¬inQ_n
9           inQ_n ← true
10          INSERT(O, n, d_max + d)
11       if vdist_n = 0
12          REMOVEVERTEX(n, V_tan)
13    else if dist_n < ρ and || obst_s − obst_n || > 2d_min
14       if dist_s ⩾ dist_n and vdist_s = v_res
15          ADDVERTEX(s, V_tan)
16       if dist_n ⩾ dist_s and vdist_n = v_res
17          ADDVERTEX(n, V_tan)
18 inQ_s ← false
```

Algorithm 2 $D,V_{tan} = \text{RAISE}(s,D,V_{tan})$
Input: s = Cell being expanded, D = Distance transform, V_{tan} = Vertex list
Output: D = Distance transform, V_{tan} = Vertex list

```
1  foreach n ∈ Adj(s)
2  if ¬inQ_n
3     if ISVALID(n)
4        inQ_n ← true
5        INSERT(O, n, d_max + dist_n)
6     if dist_n ≠ d_max
7        obst_n ← ∅
8        inQ_n ← true
9        INSERT(O, n, dist_n)
10       dist_n ← d_max
11       if vdist_n = 0
12          REMOVEVERTEX(n, V_tan)
13 inQ_s ← false
```

two surfaces around different objects separated by empty space. The idea behind SPARTAN is to only move either across the surface or along these tangential connections. In the simplest case the surface radius ρ can be chosen as the minimum distance allowed between a path and obstacles, i.e. the C-space, however it can also be chosen as a nominal clearance distance. In addition if two obstacles have intersecting planning surfaces, i.e. they are closer together than 2ρ, but do not have intersecting C-spaces, we also include the ridge between them in the planning surface. The ridge is defined as all the locations with equal distance to both obstacles, i.e the voronoi surface.

B. Distance Transform and Vertex Sampling

The aim of vertex sampling is to get a set of uniformly spaced vertices over the planning surface. The vertices can be iteratively updated whenever obstacle information is changed.

Obstacles are represented using an occupancy grid and a distance transform, a grid that stores the distance to the closest occupied cell at each location, is also maintained. The distance transform allows for the calculation of an obstacle cost and also provides the information required to sample the planning surface. Fig. 2 shows a representation of the distance transform in 2 dimensions using a color scale. The incremental distance transform algorithm we use is presented in detail in [15]. It uses expanding wavefronts to efficiently update only distances nearby newly occupied or cleared cells. The wavefront is terminated if it reaches a maximum expansion distance of d_{max}.

Sampling requires that we can test if a location is on the planning surface. This is done during the update of the distance transform with vertices found as the wavefronts are propagated from newly occupied or cleared cells. This has two major benefits; the first is that the limited expansion of the transform means that only the region nearby obstacles has to be checked. The second is that as new obstacle information causes occupied cells to be added or removed vertices are only added or removed in the region around the change, this makes the vertex sampling process iterative.

Algorithms 1 and 2 show the key functions used during the distance transform and vertex sampling update step, Tab. I provides the definition of the variables and functions that are used. When a cell s in the distance transform becomes newly occupied it has $dist_s$ set to zero, $obst_s$ set to itself and is added to the open queue to start a wavefront propagation. When a cell s becomes newly empty it has $dist_s$ set to d_{max}, $obst_s$ set to null and is added to the queue. The distance transform's wavefronts are propagated by taking a cell from the queue and calling either the LOWER or RAISE functions. The LOWER function is called if the cell taken from the queue has a valid distance, i.e. $dist < d_{max}$, it propagates out new valid distances. The RAISE function is called if the cell being expanded has $dist = d_{max}$, it propagates cleared cells by raising their distance to d_{max}. As cells have their distances changed invalid vertices must be removed and new valid ones added. The LOWER and RAISE functions also check for invalid vertices to be removed and valid locations for new vertices to be added.

Lines 1 and 2 of the LOWER function check if the cell being expanded is a good vertex. Lines 3 to 10 update $dist_n$ and $obst_n$ of the surrounding cells and add them to queue only if they have been modified and are not already in the queue. Lines 11 and 12 remove a vertex if it was just given a new $dist$. This removes existing vertices when a new obstacle is added nearby causing them to no longer

Algorithm 3 e_i = EXPAND(i)
Input: i = Index of vertex to be expanded
Output: e_i = List of vertices connected to vertex i

```
1  for j = 1...n
2    if x_j = x_i
3      continue
3    l ← ||x_j − x_i||
4    v_ij ← (x_j − x_i)/l
5    if DOT(norm_i, v_ij) > ξ or DOT(norm_j, v_ij) > ξ
6      continue
7    if ¬WILLCOLLIDE(x_i, x_j)
8      e_i ← j
```

be on the planning surface. Lines 13 to 17 test if a cell is a ridge point, the approach used is similar to the method for finding the voronoi diagram presented in [16]. When line 5 evaluates as false it indicates that the wavefront that the LOWER function is expanding has either expanded up to d_{max} or has hit a cell that has a different closer occupied cell. The second case will always occur at the ridge between the two obstacles and the first case is excluded by Line 13 which prevents the addition of ridge points with $dist_n > \rho$. In addition the distance between the two obstacles must be greater than twice the minimum clearance distance, this is to prevent the addition of ridge points when it is not possible to follow the ridge all the way through. Lines 14 and 16 determine which of the cells is actually on the ridge, this is necessary because it is unknown which of the wavefronts expanded from the two obstacles reached the ridge first.

Lines 1 and 2 of RAISE check if any of the surrounding cells are not already on the queue. They would be on the queue if they were either already cleared by a different raise wavefront or given a valid distance by a lower wavefront. In either case they can be ignored which effectively terminates the raise wavefront in regions where it has hit any other wavefront. Lines 3 to 5 check if a neighboring cell is valid and adds it to the queue to later propagate a lower wavefront. This allows only the minimum number of cells to be cleared and new valid distances propagated from nearby valid cells. Lines 6 to 10 clear neighboring cells. Lines 11 and 12 check if the cell being cleared is a vertex and remove it if it is.

C. Graph Connectivity

The edge connection rules reflect the underlying approach of moving only over the planning surface. A normal vector at each vertex pointing towards it's closest obstacle is obtained from the distance transform during sampling. Edges, defined here as straight lines, are required to be perpendicular to the normal vector of the vertices at both ends. Some slack is allowed to account for the discrete sampling of the surface that the vertices represent. This first condition eliminates the majority of potential edges, the remaining are checked to ensure they are collision free. Alg. 3 shows the function used to determine the set of vertices e_i that share an edge with the i^{th} vertex. x_i indicates the cartesian position of the i^{th} vertex and ξ is the slack allowed on the cosine of the angle between an edge and the surface normal.

D. Graph Search and Construction

The optimal path through the underlying graph is found using the A* search algorithm. The start and goal locations are added to the list of vertices with their normal vectors set to zeroes. To prevent extra work edges are only found as needed by A*, the EXPAND function shown in Alg. 3 can be used to find the neighbors of a node being expanded by A*.

V. PROPERTIES

A. Completeness

SPARTAN is resolution complete if a feasible path through vertices on the planning surface exists and the graph search method is complete (proven by Rina et al. [17]). The assumptions are:

1) Let $\sigma = \{x(t), u(t)\}$ be a path such that it is composed of straight line segments joining its vertices V.
2) $V \in \{v_{root}, v_{goal}, V_{tan}\}$ where v_{root} and v_{goal} are the root and goal points
3) $V_{tan} = \{x_i \mid d(x_i) = \rho, \|x_i - x_j\| > v_{res}\}$ where d is the distance from the nearest obstacle, ρ is the nominal distance of SPARTAN, v_{res} is the vertex separation of SPARTAN

If $\exists\sigma$, SPARTAN is resolution complete, where the resolution is v_{res}.

B. Optimality

The graph search method applied to SPARTAN is optimal (proven by Rina et al. [17]), thus SPARTAN will find the optimal path belonging to the set described in Section V-A.

A reduced visibility graph has been proven to be optimal for minimum length problems by Latombe [18]. SPARTAN, being the discrete analogue of the reduced visibility graph, is thus optimal for a minimal length cost with obstacle constraints. If an obstacle cost exists in free space, SPARTAN is no longer optimal but has suboptimal bounds. These bounds depend on the discretization error and the cost penalty for choosing to be restrained to a discrete tangential surface. The upper bound to sub-optimality is the projection error η of the optimal path σ^* to the tangential surface V_{tan}

$$\begin{aligned} \eta &= \frac{J(\sigma_{tan}) - J(\sigma^*)}{J(\sigma*)} \\ \sigma_{tan} &= \arg\min_{\sigma, \sigma \in V_{tan}} \|\sigma - \sigma^*\| \end{aligned} \tag{2}$$

where $J(\sigma)$ is the cost function.

C. Complexity

The worst case complexity of the entire search in SPARTAN is $O(N^2)$ where N is the number of vertices. The upper bound of N for a grid of length l_{grid} is

$$N \leq \frac{0.7408 * l_{grid}^3}{\frac{4}{3}\pi\rho^3} * pf(\rho, v_{res}) \tag{3}$$

where pf is the packing fraction of the maximum number of points at distance v_{res} from each other that can be fitted on

a sphere of radius ρ. The value 0.7408 is the Kepler problem [19].

This is far greater than the true running time because the edges considered are tangents to the planning surface. Each vertex in practice considers $\ll N$ connections. Other speedups including the heuristic design and prioritization of ray tracing also boosts speed, reducing the average complexity. However, SPARTAN is not optimized for solving problems over distances much larger than ρ.

VI. Experiments

A number of experiments were carried out to test the SPARTAN planning approach in real world scenarios. All the experiments were performed in an unstructured outdoor environment that contained trees, dense vegetation, power lines, bridges and non-flat terrain. The experiments rely on a small aerial vehicle with attached lidar scanner and state estimation system. The vehicle can be seen in Fig. 1 and more details on the perception and state-estimation systems can be found in [15]. The cost function used was the same as that presented in section III.

A. Offline

The offline experiments were run using a dataset collected by flying the vehicle manually through the environment. The SPARTAN planner was compared to a generic RRT* approach in two scenarios. In the first the data was played back in real time with the environment revealed up to the sensor range as the vehicle moves through it. The two planners were required to plan to a goal point with an average distance of roughly 30 meters from the vehicle and achieve a reaction time of 0.1 second (10 Hz). Different goal points were used as the vehicle moved through the environment which were chosen manually with the aim of creating interesting planning problems. Cases with trivial solutions were ignored. Table II shows that SPARTAN always found a solution in the allowed time while RRT* has a lower success rate. The RRT* is always given the maximum allowed 0.1 seconds per problem, SPARTAN uses less time on average and sits idle for the remaining allowed time. Despite this SPARTAN's paths have a lower average cost and standard deviation.

In the second scenario the two planners are required to plan through a section of fully observed environment over a distance of about 40 meters. SPARTAN was run until completion while RRT* was allowed to run until convergence. Figure 4 shows a plot of RRT*'s cost over time with the cost of the SPARTAN path (4429) also shown. The RRT* eventually converges to a slightly lower cost path (4302), however it takes a long time (94.8 seconds) to find a path that is better than the SPARTAN solution which was found in only 0.55 seconds. This factor of 172 is indicative of the long time a sampling based planner takes to converge to the fact that the optimal answer lies on the tangential surface (which is inbuilt within SPARTAN). The small difference in cost between the converged RRT* and SPARTAN solutions can be attributed primarily to SPARTAN's discretization error. Figure 3 shows the static test case environment and resulting paths. The solution found by RRT* at the time SPARTAN terminated is shown to be a low quality path with SPARTAN finding one of higher quality. These two paths are also typical of the quality of path observed during the replanning experiment.

TABLE II
Comparison for 116 planning problems

	Success	Average Cost	Average Time (ms)
SPARTAN	100%	2878(±950)	71(±87)
RRT*	76.72%	3472(±1240)	102(±3)

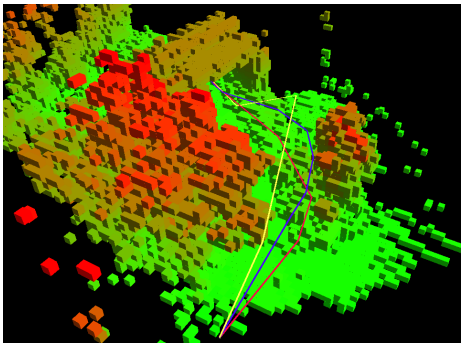

Fig. 3. Occupancy grid representation of the problem environment (colored by height) along with the SPARTAN solution (blue), the RRT* solution at the time SPARTAN terminates (yellow) and the converged RRT* solution at 400 seconds (red)

B. Online Closed-Loop Flights

The SPARTAN planner was also run online on the vehicle flying in a fully autonomous mode. The planner was run onboard using the vehicle's 1.8 GHz core 2 Duo flight computer and ran alongside the vehicle's perception (including lidar and stereo camera processing) and state-estimation systems. The planner ran at 2Hz and fed paths to a flight control system which followed the paths. A number of missions have been flown with length's of up to 200 meters and which involve dense vegetation, power lines, low bridges and terrain of changing elevation. The goal location was manually set and remained constant for the length of each mission. Figure 5 shows logs of vehicle position during

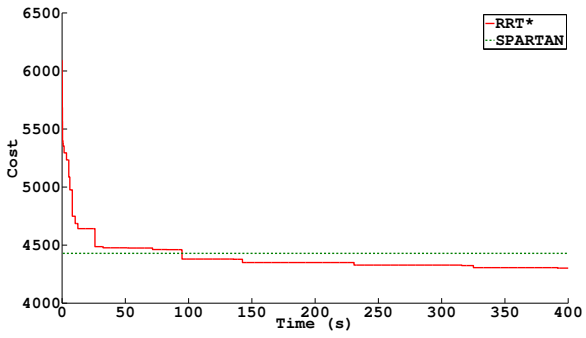

Fig. 4. Plot of cost with time. SPARTAN has a cost of 4429 after 0.55s. RRT* at the same time has a cost of 5397. After 94.8s RRT* has the same cost as SPARTAN. After 400s, RRT* has a cost of 4302

Fig. 5. A selection of autonomous flights by the MAV using SPARTAN; nominal path in yellow, actual path in red. a) flying around a single tree b) flying under a low bridge c) following a narrow tree lined road.

a selection of missions. The first shows a simple tree avoidance, the second shows the ability to fly under a low bridge and the third shows a longer mission through dense vegetation up a hill which resulted in the vehicle following a road before eventually finding a way around the line of trees to it's goal. In addition to the cost function used above the vehicle's altitude above ground was constrained for autonomous flights.

VII. Conclusion

We have presented a planning approach, SPARTAN, that can find paths in real-time through unstructured outdoor environments with no prior knowledge and obstacles of varying clutter that are regularly updated by a sensor with limited range. SPARTAN can create plans 172 times faster than the state of the art and using limited onboard resources, has flown autonomously around trees, avoided powerlines, under bridges and over changing terrain with missions over 200 meters in length. This is made possible using a graph wrapped in a tangential surface around obstacles and leveraging the sparsity achieved by only considering edges tangential to the surface.

As future work the path update required when changes are made to obstacles by the sensor could be made fully incremental. While the vertices are currently updated iteratively the graph must be searched from beginning because it is unknown which edges have been affected. In addition some slack is currently allowed when checking if an edge is tangential, this can cause multiple similar edges to be added and create significant extra work. A better method could be to check neighboring vertices for the existence of a more tangential edge and only add edges that are locally the most tangential.

Acknowledgments

The authors gratefully acknowledge Lyle Chamberlain, and Andrew Chambers for their help with experimentation. The work described in this paper is funded by the Office of Naval Research under grant number N00014-10-1-0715.

References

[1] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, 2012.

[2] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent & Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.

[3] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, 2005.

[4] P. Tsenkov, J. Howlett, M. Whalley, G. Schulein, M. Takahashi, M. Rhinehart, and B. Mettler, "A system for 3d autonomous rotorcraft navigation in urban environments," in *AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, HI*, 2008.

[5] M. Whalley, P. Tsenkov, M. Takahashi, G. Schulein, and G. Goerzen, "Field-testing of a helicopter uav obstacle field navigation and landing system," in *65th Annual Forum of the American Helicopter Society, Grapevine, TX*, 2009.

[6] L. Kavraki, P. Svestka, J. 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, 1996.

[7] S. LaValle, "Rapidly-exploring random trees a new tool for path planning," 1998.

[8] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proc. Robotics: Science and Systems*, 2010.

[9] S. LaValle, *Planning algorithms*. Cambridge Univ Pr, 2006.

[10] D. Wooden and M. Egerstedt, "Oriented visibility graphs: Low-complexity planning in real-time environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 2354–2359.

[11] G. Hoffmann, S. Waslander, and C. Tomlin, "Quadrotor helicopter trajectory tracking control," in *AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, Hawaii*, 2008.

[12] Y. Kuwata and J. How, "Three dimensional receding horizon control for uavs," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, vol. 3, 2004, pp. 2100–2113.

[13] S. Hrabar, "Reactive obstacle avoidance for rotorcraft uavs," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4967–4974.

[14] S. Scherer, S. Singh, L. J. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 549–574, May 2008.

[15] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske, and S. Singh, "River mapping from a flying robot: state estimation, river detection, and obstacle mapping," *Autonomous Robots*, pp. 1–26, 2012.

[16] B. Lau, C. Sprunk, and W. Burgard, "Improved updating of euclidean distance maps and voronoi diagrams," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan*, 2010.

[17] D. Rina and J. Pearl, "Generalized best-first search strategies and the optimality of a*," *Journal of the ACM*, vol. 32, no. 3, pp. 505–536, 1985.

[18] J.-C. Latombe, *Robot motion planning*. Springer, 1991.

[19] T. Hales, "A proof of the kepler conjecture," *The Annals of Mathematics*, vol. 162, no. 3, pp. 1065–1185, 2005.