Efficient C-Space and Cost Function Updates in 3D for Unmanned Aerial Vehicles

Sebastian Scherer, Dave Ferguson, Sanjiv Singh

***Abstract*— When operating in partially-known environments, autonomous vehicles must constantly update their maps and plans based on new sensor information. Much focus has been placed on developing efficient incremental planning algorithms that are able to efficiently replan when the map and associated cost function changes. However, much less attention has been placed on efficiently updating the cost function used by these planners, which can represent a significant portion of the time spent replanning. In this paper, we present the Limited Incremental Distance Transform algorithm, which can be used to efficiently update the cost function used for planning when changes in the environment are observed. Using this algorithm it is possible to plan paths in a completely incremental way starting from a list of changed obstacle classifications. We present results comparing the algorithm to the Euclidean distance transform and a mask-based incremental distance transform algorithm. Computation time is reduced by an order of magnitude for a UAV application. We also provide example results from an autonomous micro aerial vehicle with on-board sensing and computing.**

I. Introduction

Unmanned aerial vehicles (UAVs) have many civilian and military applications. However currently such vehicles can only operate at a high altitude because they are not able detect and avoid collisions with obstacles. Our research focuses on UAVs that operate close to obstacles such as buildings, vegetation and power lines. It is necessary to fly low because one can get a better viewpoint, and can avoid manned air traffic.

We assume that the robot (Fig. 1) is given a sequence of sparse waypoints that have to be reached to make observations and that the robot has to find its way around obstacles that might be in its path. Flying close to and among obstacles is difficult because of the challenges in sensing small obstacles, and in repeatedly planning a safe path that avoids obstacles in three dimensions.

Some aspects of collision avoidance are easier for air vehicles than ground vehicles. Any object close to the intended path of an air vehicle must be avoided as opposed to ground vehicles where the ground is always close and deviations from the nominal ground plane indicate obstacles. The use of rotorcraft, rather than fixed wing aircraft also simplifies the problem because in the worst case it is possible for rotorcraft to come to a hover in front of an obstacle. Still, the availability of appropriate sensors, logistical issues of mounting a vehicle with sufficient sensing, computing and communication gear, and the risk involved in such experimentation has kept researchers from significant progress in this area.

Sebastian Scherer and Sanjiv Singh are with the Robotics Institute, Carnegie Mellon University. Dave Ferguson is with Intel Research Pittsburgh.
email: [basti, dif, ssingh]@cmu.edu

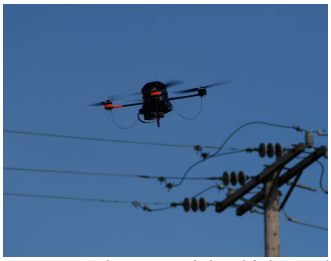

Fig. 1. The autonomous quad-rotor aerial vehicle used for testing. Here, the vehicle is close to a typical pole and wire obstacle in the environment. The system is equipped with a computer, a GPS, an inertial measurement unit, and a ladar scanner to sense the environment. The size of the vehicle with rotors is less than one meter.

Fig. 2. Data flow in a completely incremental planning paradigm. Sensor data causes an update to the environment map which propagates to changes in the cost function used by the planner. Finally an incremental path planner calculates a new path to the goal with the changed cost values.

The main contribution of this paper is a completely incremental framework that enables frequent replanning of paths in 3D for aerial vehicles (Fig. 2) at an order of magnitude lower computation time than current approaches to obstacle expansion. Incremental planning has been well studied [1], [2], however in prior work it has mostly been assumed that planning is performed on a C-space expanded cost map and that calculating this cost map is not difficult. In 2D the number of cells affected to update the cost function based on a new obstacle is a function of the square of the maximum expansion. For example for an expansion of 20 cells about 400 cells are affected. However, in 3D it is not trivial to update the cost function after new sensor data has been received because the number of cells that potentially need to be recomputed changes cubicly. So for the same expansion we have to potentially look at roughly 8000 cells.

In this paper, we present the Limited Incremental Distance Transform algorithm (LIDT) to efficiently perform this cost function update. In our approach, changes to the environment grid map [3] are propagated via the described limited incremental distance transform. The list of changed costs is then used by an incremental planning algorithm [2] to update the current plan.

In previous work we have addressed the problem of navigating among obstacles with a layered architecture of a reactive avoidance algorithm that learns parameters by observing operator behavior and a global planning algorithm that operates on an evidence grid [4], [5]. A path planner for a

fixed-wing UAV using a grid lattice that operates on C-Space expanded obstacles is proposed by Hwangbo et al. [6]. Griffiths et al. use an RRT based planner with a grid height map to plan a path through the environment for an autonomous fixed wing UAV in an urban environment [7]. Whalley et al. explore an environment with an autonomous helicopter. Paths among obstacles are planned to use a Voronoi diagram [8]. Andert & Goormann build an occupancy grid for planning with a stereo rig on a UAV [9].

All of the related work described above uses a grid based approach to represent obstacles. Since a C-space expansion and cost function have to be computed for planning our approach could be used to update the costs in the map.

We begin by describing in Section II how navigation cost functions are typically computed and their relationship to distance transforms. In Section III we discuss several potential approaches on computing such cost functions. Section IV describes our novel algorithm and Section V presents experiments and results, including examples from our autonomous aerial vehicle.

II. C-Space Expansion and Cost Functions

Incremental replanning using D* or its variants [1], [2] is a general and efficient approach to adapt to a partially-known or dynamic environment. However it is not always easy to determine how the map for planning changed on a 3D grid since it is necessary to plan in the changed free configuration space [10] with changed costs.

Even though it is theoretically possible to plan with an arbitrary C-space expansion and cost function, the dominant factor is the distance to the closest obstacle. Therefore we assume that we can calculate the cost for planning around obstacles from distance.

In this paper we assume a spherical robot, a reasonable assumption for a rotor-craft. A spherical expansion is easy to compute if we know the distance to the closest obstacle. If the distance is closer than the radius r_v of the UAV one is in contact with an obstacle. We set the cost for such an edge to be infinite cost.

We can express the general cost function between two vertices as follows:

$$c(k,l) = \begin{cases} \infty & if\ d(l)_o^2 < r_v^2 \\ \gamma \cdot obst(l) + dist(k,l) & otherwise \end{cases} \quad (1)$$

where $c(k,l)$ is the cost between position k and l, and the closest obstacle is $d(l)_o^2$. The cost consists of a scale factor γ that scales between the cost of obstacles $obst(l)$ and the cost of the distance $dist(k,l)$. Since we can express the C-space expansion in the cost function as an infinite cost, we will from here on refer to the cost function as the cost of an edge that also includes the C-space expansion.

The $dist$ function can be any valid distance metric but one common metric is the squared Euclidean distance:

$$dist(k,l) = (k_x - l_x)^2 + (k_y - l_y)^2 + \alpha(k_z - l_z)^2 \quad (2)$$

where the x, y, z components are the displacement in the respective axis. If $\alpha = 1$ going left/right or to climb/sink is equal cost. If $\alpha > 1$ the robot will prefer to move laterally and if $\alpha < 1$ it will prefer to move vertically.

Several interesting cost functions for UAVs depend on the distance to the closest obstacle. For example, the shortest path with a clearance to obstacles:

$$obst(l) = max(0, d_{max}^2 - d(l)_o^2) \quad (3)$$

A maximum distance d_{max}^2 determines a cutoff beyond which the closest obstacle does not influence the path anymore. In the extreme case if d_{max}^2 and γ is large the path found will correspond to the solution of the Generalized Voronoi Graph (GVG) [10] since the path will first lead away from the obstacle to get onto the Voronoi graph and then the lowest cost path will be on the graph and finally will go away from the graph to the goal point. In a natural outdoor environment the separation of obstacles is in many cases unbounded so that planning on that boundary would lead to too long paths.

Another useful cost function is to stay close to obstacles up to a desired distance d_{des} but not too close. This can be important for stealth reasons but also one might want to stay closer to obstacles to avoid wind or to stay localized. In this case the cost function can be expressed as follows:

$$obst(l) = \mid d_{des}^2 - min(d_{max}^2, d(l)_o^2) \mid \quad (4)$$

Note that for both $obst(l)$ functions it is necessary to know $d(l)_o^2$, the distance to the closest obstacle up-to the maximum distance d_{max}^2. Naively computing the distance $d(l)_o^2$ is expensive for large d_{max}^2 that are typically used in planning for UAV. The contribution of this paper is an efficient algorithm for calculating the changes to $d(l)_o^2$.

III. Distance Transform Algorithms

The distance $d(l)_o^2$ is the result that is computed by the distance transform algorithm. There are many possible distance metrics that can be applied, however the squared Euclidean distance is most useful for our application since we want the obstacle expansion and C-space expansion to be spherical. The property of the distance transform that we want can be expressed for a $mxnxo$ grid with boolean obstacles $b[i,j,k]$ as follows:

$$EDT(x,y,z) = \\ min(b[i,j,k] : (x-i)^2 + (y-j)^2 + (z-k)^2) \quad (5)$$

This property says that for every coordinate in the distance transform $EDT(x,y,z)$ we determine the minimum of the distance to all the obstacles $b[i,j,k]$. This would of course not be a very efficient algorithm in most cases however it shows what we need to compute. The Manhattan distance L_1 transform can be written like this:

$$MDT(x,y,z) = \\ min(b[i,j,k] :\mid x-i \mid + \mid y-j \mid + \mid z-k \mid) \quad (6)$$

An efficient non-incremental linear time algorithm to calculate the distance transform was proposed by Meijster et al. [11]. Even though this algorithm is very efficient, we will show in section V that repeatedly recomputing the result takes too long to be useful for navigation on a large grid. The

algorithm scans the grid in three phases. In each phase the grid is scanned along a different axis forward and backward to determine a minimum. Overall the work performed is six passes through the grid for three dimensions. In two dimensions four passes are necessary.

A simple incremental approach to update the cost function in a grid is to update the grid with a mask of the distances to the obstacles. We will refer to this algorithm as "mask algorithm" in the algorithm evaluation. Every time an obstacle is added a convolution of the surrounding area is performed to check if any of the distances is larger than the distance in the mask. In the case of obstacle removal all non-obstacle cells that are in the mask of the obstacle are set to infinity and a region of two times the size of the mask is scanned for all obstacles. The region inside the removed obstacle is checked for any obstacle and the closest distance is restored. This algorithm serves as an incremental algorithm that one could implement easily.

While the runtime of the Meijster et al. algorithm depends on the size of the grid and is therefore non-incremental, the runtime of the mask algorithm depends on the number of obstacles added and removed. The algorithm we propose also depends on the number of obstacles that changed however if only a small number of distances changes less work has to be performed by the LIDT algorithm.

Kalra et al. [12] developed an incremental algorithm to reconstruct the Generalized Voronoi Diagram (GVD) in 2D. The GVD is based on a quasi-Euclidean distance transform of the obstacles. The algorithm is the basis of the algorithm presented in this paper, however we have modified the incremental GVD algorithm to make it suitable for C-space and cost function updates. The presented algorithm also adds another variable to keep track of the changes in the distance transform while it is being computed that can then be used in an incremental planning algorithm (Also see Fig. 2) . Furthermore we have generalized the algorithm to be applicable for different distance metrics (such as the squared Euclidean distance metric) while the original algorithm only allowed a quasi-Euclidean expansion. Also one can control the maximum amount of computation per obstacle in the LIDT algorithm because one controls the maximum distance d_{max} that needs to be expanded into account.

IV. Limited Incremental Distance Transform Algorithm

A. Intuition

The Limited Incremental Distance Transform algorithm provides an efficient solution to keep an updated distance transform for changes to the cost function in the environment.

The algorithm is an incremental version of the brushfire algorithm and, as with the original brushfire algorithm it propagates a wavefront of distances to update the distance for each cell to its closest obstacle. For a good explanation of the brushfire algorithm also see Choset et al. [10]. The open list O keeps track of the wavefront and contains the cells that need to be expanded. Initially if only obstacles are added, the values of cells are lowered from d_{max} to consistent (or correct) distance values in the same way that brushfire would proceed. Since the values are sorted by increasing distance the cells with the smallest distance get updated first. Finally, the wavefront that is moving outwards terminates if the distance has reached a value that is larger than any of the neighboring cells or if the grid boundary has been reached.

```
INITIALIZE()
1  O ← ∅
2  foreach cell s
3    dist_s ← d²_max
4    dist_s^new ← d²_max
5    dist_s^old ← d²_max
6    obst_s ← ∅

SETOBSTACLE(o)
1  if dist_o^new ≠ 0
2    dist_o^new ← 0
3    obst_o ← o
4    UPDATEVERTEX(o)

REMOVEOBSTACLE(o)
1  dist_o^new ← d²_max
2  obst_o ← ∅
3  if dist_o < d²_max
4    UPDATEVERTEX(o)

CALCULATEKEY(o)
1  return min(dist_o, dist_o^new)

UPDATEVERTEX(o)
1  key ← CALCULATEKEY(o)
2  if o ∈ O
3    UPDATE(O, o, key)
4  else
5    INSERT(O, o, key)

DISTANCE(n, s)
1   Squared Euclidean:
2  v ← pos_n − pos_obst_s
3  return v · v

DISTANCE(n, s)
1   Quasi Euclidean:
2  v ← pos_n − pos_s
3  return v · v + dist_s^new
```

Fig. 3. The Limited Incremental Distance Transform Algorithm (Helper functions).

If an obstacle is removed a similar sweep outward propagates the changes to cells whose previous distance values are based on the removed obstacle and updates the distance for those cells since they now have a too close distance value. The cost to each of these cells is then updated based on the closest valid obstacle. Once the removal wavefront terminates each cell that does not have a valid obstacle will be updated with a valid obstacle (up to d_{max}).

It is important to note that the size of the queue in the wavefront depends on the radius of expansion. The number of cells in the queue is dependent on the radius r of the wavefront and grows linearly with the radius $\mathcal{O}(r)$. However since we are calculating the expansion in 3D the number of cells on the surface of the sphere grows with the square of the radius $\mathcal{O}(r^2)$. Also the maximum radius that has to be expanded depends on the size of the Voronoi region that is affected. One worst case example is an empty grid with one obstacle that is removed. In that example first all the cells going outward have to be invalidated and then all cells have to be lowered correctly again. In the worst case one therefore has to look at the grid twice for every obstacle removed.

For our application we are interested in computing the distance transform only out to a maximum distance d_{max}. As such the incremental distance transform propagation can be terminated once this distance is reached. This can save a significant amount of computation if the Voronoi region that changes is large.

B. Details

The algorithm pseudocode is split in two parts the helper functions are shown in Fig. 3 and the main functions are shown in Fig. 4.

In INITIALIZE all cell distances are set to d^2_{max} and the obstacle pointer is emptied. As the environment changes obstacles are removed and added with SETOBSTACLE and REMOVEOBSTACLE. If an obstacle is added its distance is

```
LOWER(s)
1  foreach n ∈ Adj(s)
2   if dist_n^new > dist_s^new
3     d' ← DISTANCE(n, s)
4     if d' < dist_n^new
5       dist_n^new ← d'
6       obst_n ← obst_s
7       UPDATEVERTEX(n)

RAISE(s)
1  foreach n ∈ Adj(s)
2   WAVEOUT(n)
3  WAVEOUT(s)

WAVEOUT(n)
1  if n ≠ obst_n
2    dist_n^new ← d_max^2
3    obst_n^old ← obst_n
4    foreach a ∈ Adj(n)
5     if VALID(obst_a)
6       d' ← DISTANCE(n, a)
7       if d' < dist_n^new
8         dist_n^new ← d'
9         obst_n ← obst_a
10   if obst_n ≠ obst_n^old
11     UPDATEVERTEX(n)

INCREMENTALDISTANCETRANSFORM(O)
1  C ← ∅
2  while O ≠ ∅
3   s ← POP(O)
4   if dist_s^new < dist_s
5     dist_s ← dist_s^new
6     LOWER(s)
7     if dist_s ≠ dist_s^old
8       INSERT(C, s)
9       dist_s^old = dist_s
10  else
11    dist_s ← d_max^2
12    RAISE(S)
13    if dist_s ≠ dist_s^new
14      updateVertex(s)
15 return C
```

Fig. 4. The Limited Incremental Distance Transform Algorithm (Main functions).

set to zero and the obstacle points to itself. Then the obstacle is added to the queue to be expanded. Similarly a removed obstacles distance is set to d_{max}^2 and it is added to the queue with the priority of the old distance it used to have.

Since the update to the grid should always be with increasing priority the key is calculated from the smaller of the two distance values in CALCULATEKEY and the heap is updated in UPDATEVERTEX with the new priority unless the element has an infinite priority.

Using our algorithm one can calculate a squared Euclidean distance in DISTANCE or a quasi-Euclidean distance that is the shortest distance on a 26-connected grid. It is possible to calculate the squared Euclidean distance because we always keep track of the location of the closest obstacle in $obst_s$. The $obst_s$ pointer tells us if a grid cell needs to be updated because it points to an obstacle. If that obstacle changes all cells pointing to that obstacle need to be updated.

The main work of updating the distance transform and keeping track of changed cells happens in INCREMENTALDISTANCETRANSFORM which returns a list of updated distances C. If we added or removed obstacles the open list O will not be empty and so we take the first element of the list and LOWER the node if it is over-consistent and RAISE it otherwise. Since all nodes have to be made LOWER eventually we can keep track of the changed distances in lines 7-9.

LOWER updates the distance of each adjacent node and adds it to the queue if the distance changed. Also we update the associated obstacle if it changed.

RAISE on the other hand propagates out a removed obstacle in WAVEOUT and so we first set the distance to be infinity and try to get a new distance for an adjacent node. If the associated obstacle changed we put the item back on the queue.

The algorithm terminates when the open list is empty. At this point all the cells in the grid have consistent distance values and have a valid obstacle pointer if their distance is less than d_{max}. A list of changed cells is in C.

We assume that the open list O has three operations: INSERT inserts an element in the open list with a given priority key, UPDATE updates the key of an element already in the queue, and POP returns and removes the top element from the priority queue.

Even though one can implement the open list O as a binary heap there is a fast data structure that can be used in our application because the maximum distance is limited to d_{max} and we are operating on a grid with integer values of the keys. Since there is only a small range of key values we create a hash table with the distances as key values. On every update we keep track of the lowest distance element. If we pop an element we update the lowest distance if it changes. To insert we just add the element to the list at the appropriate key value. This data structure allows $\mathcal{O}(1)$ for INSERT, UPDATE, and POP.

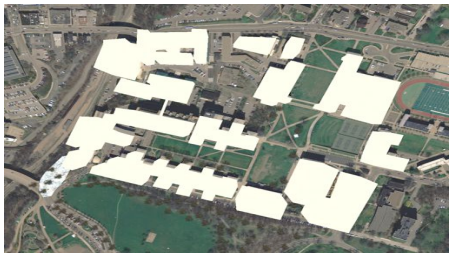

Fig. 5. The virtual campus environment of Carnegie Mellon University, Pittsburgh, PA that is used in the simulation experiments. Buildings that were added to the digital elevation model are shown in white. A hemispherical 200m range 3D range sensor is simulated to update the environment map held by the robot.

V. EXPERIMENTS

There are certain tradeoffs between using an incremental and non-incremental algorithm that need to be considered. In this section we examine some parameters that influence the performance of the LIDT algorithm and compare it to the fastest non-incremental algorithm and a simple incremental algorithm we denote the "mask" approach (described in Section III). A recent survey [13] showed that the algorithm developed by Meijster [11] is fastest in almost all test cases over a variety of problems. We therefore also compare the incremental algorithms to a 3D implementation of this algorithm.

A. Simulation experiments

To determine the effectiveness of the limited incremental distance transform algorithm for aerial vehicle planning we evaluated it for missions in a simulated environment and compared it with two other distance transform algorithms: the non-incremental distance transform algorithm by Meijster et. al and the simple incremental 'mask' algorithm. See Section III for more details on the two competing algorithms.

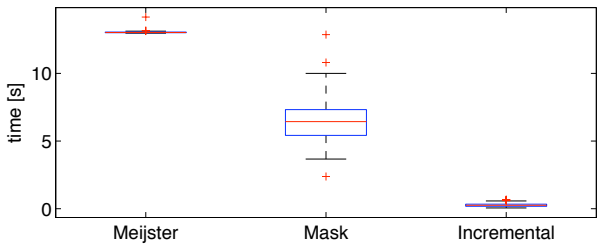

Fig. 6. A comparison of running three distance transform algorithms in the environment shown in Fig. 5. The algorithms are run with the same sensor inputs on a grid that was initialized with an elevation model. *Meijster* is the non-incremental distance transform algorithm by Meijster et al. *Mask* is a simple incremental algorithm that updates based on a distance mask for each obstacle. *Incremental* is the limited incremental distance transform algorithm. The environment map changes as the robot discovers new obstacles and removes invalid obstacles from its initial map. $d_{max} = 20$. The mean of the number of obstacles added was 370 ± 32.3. The mean of the number of obstacles removed was 14 ± 6.5. The grid size considered is 512x512x80. Box and whisker plot legend: The red line is the median, the blue box extends from the lower quartile to the upper quartile, and the whiskers extend to 1.5 of the interquartile range. Red crosses represent single run outliers.

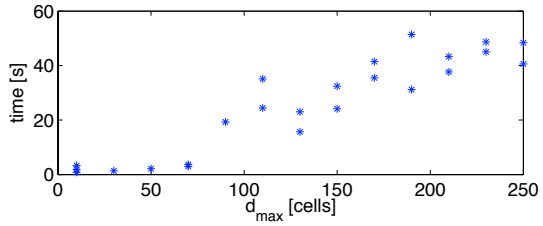

Fig. 7. Initial calculation times for an empty 512x512x80 grid initialized with obstacles from a digital elevation model for different expansions. As the expansion increases so does the initialization time because a lot of cells need to be updated for a large expansion in the limited incremental distance transform.

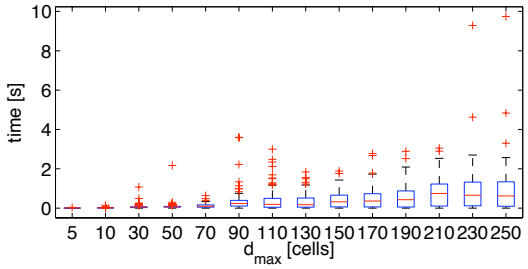

Fig. 8. A box and whisker plot for the limited incremental distance transform of the computation time in seconds for an increasing value of the maximum distance expanded d_{max} . The computation time spreads out more as d_{max} increases since a changed obstacle cell can affect a larger Voronoi region however the region affected can also be small if the obstacle added is close to existing obstacles. The grid size considered is 512x512x80. For the legend of the box and whisker plot see Fig. 6.

Algorithm	Incremental	Mean calc. time	Std. Dev.
Meijster	No	13.05s	0.16s
Mask	Yes	6.61s	2.21s
LIDT	Yes	0.27s	0.12s

TABLE I

CALCULATION TIMES OF ONE UPDATE FOR THE ALGORITHM BY MEIJSTER ET AL., THE MASK ALGORITHM AND THE LIMITED INCREMENTAL DISTANCE TRANSFORM ALGORITHM (LIDT). $d_{max} = 20$

We want to simulate an algorithm load that is similar to a real extended mission of our micro aerial vehicle in a simulated environment of the campus at Carnegie Mellon University, Pittsburgh, PA, USA. See Fig. 5 for a screen shot from our simulation.

The simulation consists of a second order dynamic model of our quad-rotor helicopter shown in Fig. 1, a hemispherical 3D range sensor with a range of 200m, and a geometric model of campus.

A path tracking algorithm controls the helicopter and regulates speed based on the distance to the closest obstacle. As soon as a sensor measurement is received the evidence grid is updated and changes are given to one of the three distance transform algorithms. The changes to the grid and the cost function are then propagated to a D* Lite [2] planning algorithm. All algorithms run on a 2.5GHz Intel Core 2 Duo processor.

We ran an experiment in this environment with the robot avoiding obstacles that it saw within its range sensor and the three algorithms calculating the changes to the cost function for D* Lite. The maximum expansion for the grid was set to $d_{max} = 20$ and the environment map was initialized with the prior digital elevation model. During its traverse, a mean of 370 (standard deviation 32.3) obstacles were added and 14 (standard deviation 6.5) obstacles were removed.

Overall the limited incremental distance algorithm performed over an order of magnitude better than the competing approaches (Fig. 6 and Table I), because the expansion distance is large and a number of obstacles have to be removed each iteration. As the number of obstacles removed decreases the 'mask' algorithm performs better because obstacle removal is an expensive operation for this algorithm. The non-incremental Meijster et al. algorithm has to traverse the grid several times and therefore cannot perform as well as the incremental algorithms which only have to update a small local region.

As the expansion distance decreases there is a point at which the mask algorithm becomes more efficient because it can use the processor cache better since it performs lots of sequential accesses. However if d_{max} increases significantly the runtime of the mask algorithm will increase beyond the non-incremental algorithm by Meijster because it must perform double work.

The runtime of the limited incremental distance transform depends on the size of the Voronoi region affected and in many cases if obstacles are close to each other then the affected regions are relatively small. In the case of the simulation and in realistic scenarios the changes to the map will be local and in a neighborhood of existing obstacles. In this case since only a small number of cells need to be updated the limited incremental distance transform has a significant advantage.

The standard deviation in Table I indicates that the computation time for the LIDT algorithm varies less than for the Mask algorithm.

In a second experiment we evaluated the performance of the maximum distance d_{max} on computation time for our campus environment.

Fig. 9. This sequence of images shows our autonomous quadrotor micro aerial vehicle (Fig. 1) avoiding a tree that is in the straight line mission path to its goal. As soon as the robot detects the obstacle it plans a path in 3D with a wide berth around the obstacle. The cost function is set up in such a way that the robot will prefer to move laterally and therefore we only show a top view. Since $d_{max} = 11$ the obstacle is avoided by a large margin. The mission path is shown in black and the planned avoidance path is shown in red. The black line shows the path of the vehicle as recorded by GPS. The tree obstacle is shown in green.

We ran a total of 37461 updates with varying maximum expansion values for the limited incremental distance transform. During testing we reset the environment map periodically and then recalculated the incremental distance transform with obstacles based on the digital elevation model which exemplifies the overhead cost of the incremental distance transform for starting from scratch.

The initial overhead of the incremental distance transform algorithm is significant if a large number of obstacles already exist in the environment map since the expansion has to perform more work per cell than the distance transform algorithm. A scatter plot of the overhead is shown in Fig. 7.

After the initial overhead, however, subsequent updates are relatively inexpensive. With an increase in the distance d_{max} the median computation time increases as well as the overall spread in computation. Since the potential number of cells affected by a change in the environment increases with d_{max} we also see a larger variation in computation times. Even with a large expansion of 250 cells, however, the limited incremental distance transform algorithm can still outperform the algorithm by Meijster. At such an expansion distance the 'mask' algorithm would not be feasible because on every update it essentially needs to check every cell in a 512x512x80 grid.

B. Quad-rotor experiment

Our autonomous quad-rotor robot (Fig. 1) wants to avoid obstacles with a wide berth if possible because it increases the safety of the path and gives a better perspective for sensing. The robot is equipped with a GPS, INS, and a ladar scanner to sense the environment. All computation is performed on-board and a planning cycle is performed at about 3Hz. It can fly missions of up to 20 minutes and can be given a series of waypoints it should reach. Since typically almost all of the waypoints are low the straight line path to the goal is typically obstructed by obstacles. The ladar scanner returns distances to obstacles and the information is processed into a global map as a 3D array in memory and a path is planned using a distance transform expansion. The position of the robot has some uncertainty (2m) that is incorporated as part of the C-space expansion. The robot can avoid obstacles in three dimensions since the planning algorithm also operates in three dimensions.

Fig. 9 shows the quadrotor avoiding a tree in its straight line path to the goal. The cost function used for the planning algorithm in this case is the same as as in Eq. 3 with $d_{max} = 11$. Since that expansion is large and we are planning in 3D it is beneficial to use an incremental distance transform algorithm to update the changes to the cost function. The C-space expansion is set to 2 meters but since it is expensive to go close to obstacles the path gives the obstacle a wide berth. In this experiment the obstacle is avoided with a speed of 2m/s.

VI. Conclusion and Future Work

We have presented a completely incremental framework for planning paths in 3D that enables recomputation of costs an order of magnitude faster than current approaches. This speed up is made possible by using a novel limited incremental distance transform algorithm. This algorithm exploits the local nature of cost function updates when obstacles are added or removed from the map and enables autonomous aerial vehicles to respond to newly observed obstacles (or obstacles that no longer exist) in real-time. We have provided results from simulation demonstrating the benefits of the approach and illustrative examples from a physical implementation on a quad-rotor micro aerial vehicle autonomously navigating in Pittsburgh, PA.

Acknowledgments

The authors would like to thank Lyle Chamberlain, Wenfan Shi, and Maggie Scholtz for developing and testing the aerial robot.

References

[1] A. Stentz, "The d* algorithm for real-time planning of optimal traverses, Tech. Rep. CMU-RI-TR-94-37, Oct 1994.
[2] S. Koenig and M. Likhachev, "D* lite," *Eighteenth national conference on Artificial intelligence*, Jul 2002, twocol.
[3] M. C. Martin and H. Moravec, "Robot evidence grids, Tech. Rep. CMU-RI-TR-96-06, Mar 1996.
[4] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *Int. J. Robotics Research*, vol. 27, no. 5, pp. 549–574, May 2008.
[5] S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli, "Flying fast and low among obstacles," *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2023 – 2029, Mar 2007.
[6] M. Hwangbo, J. Kuffner, and T. Kanade, "Efficient two-phase 3d motion planning for small fixed-wing uavs," *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1035 – 1041, Mar 2007.
[7] S. Griffiths, J. Saunders, A. Curtis, and T. McLain, "Obstacle and terrain avoidance for miniature aerial vehicles," *IEEE Robotics and Automation Magazine*, Jan 2006.
[8] M. Whalley, M. Freed, R. Harris, and M. Takahashi, "Design, integration, and flight test results for an autonomous surveillance helicopter," *Proceedings of the AHS International Specialists' Meeting on Unmanned Rotorcraft*, Jan 2005.
[9] F. Andert and L. Goormann, "Combined grid and feature-based occupancy map building in large outdoor environments," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 2065 – 2070, Jan 2007.
[10] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*, Apr 2005.
[11] A. Meijster, J. Roerdink, and W. Hesselink, "A general algorithm for computing distance transforms in linear time," *Mathematical Morphology and its Applications to Image and Signal Processing*, pp. 331–340, Jan 2000.
[12] N. Kalra, D. Ferguson, and A. Stentz, "Incremental reconstruction of generalized voronoi diagrams on grids," *Proc. of the International Conference on Intelligent Autonomous Systems*, Mar 2006.
[13] R. Fabbri, L. Costa, J. Torelli, and O. Bruno, "2d euclidean distance transform algorithms: A comparative survey," *ACM Computing Surveys (CSUR*, vol. 40, no. 1, Feb 2008.