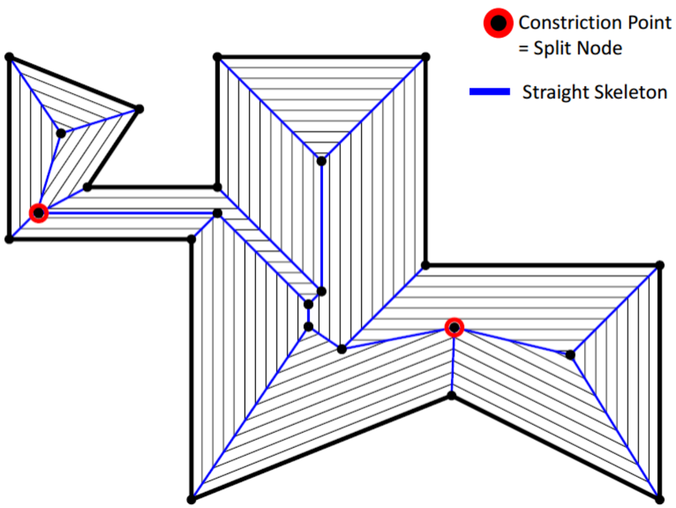
Real-Time 3D Navigation for Autonomous Vision-Guided MAVs

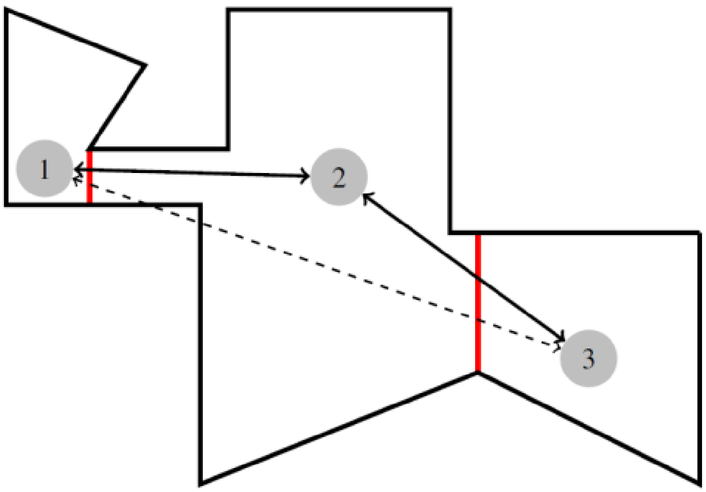
Seungwon Song

2017.05.23
CS686
Paper Presentation #2

KAIST

Suzi Kim's Presentation





Cell Decomposition
- Shrink
- Split

Cell Visit
Using TSP

Contents

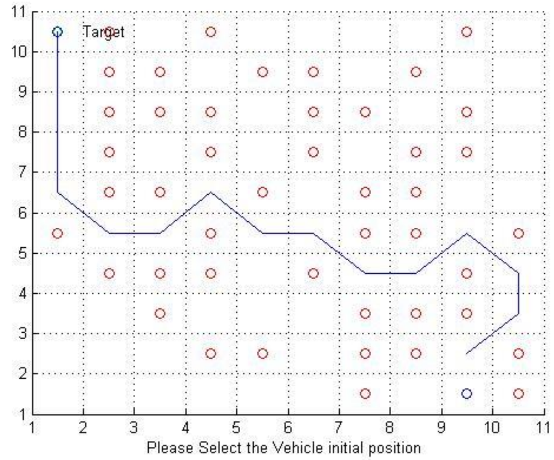
- Introduction
- Conventional approach
- Basic concepts
- Detail of each concepts
- Result

Introduction

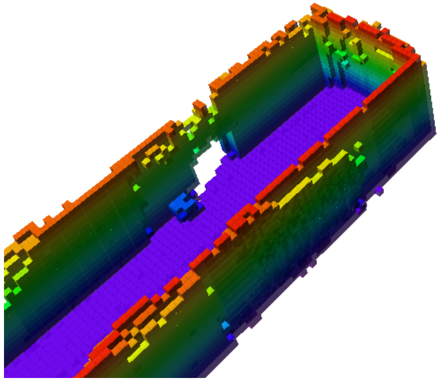
- Regular 3D state lattice requires a large amount of memory while graph search even though problem is easier to solve.
- Using Octree-based state lattice which represent discretizes large swathes of free space into few symbolic octants.
- Warning!
  - It does not contain any Math, just in robotical perspective!
  - So, just basic result comparison with conventional method.

Conventional Approach

- There are several grid-based path planning method in 2D.
- In 3D, there are too many points, so reduced them by using Octomap.
- Using reduced 3D grid, Researchers can use conventional A* or other algorithms



Grid based pathplanning



Octomap

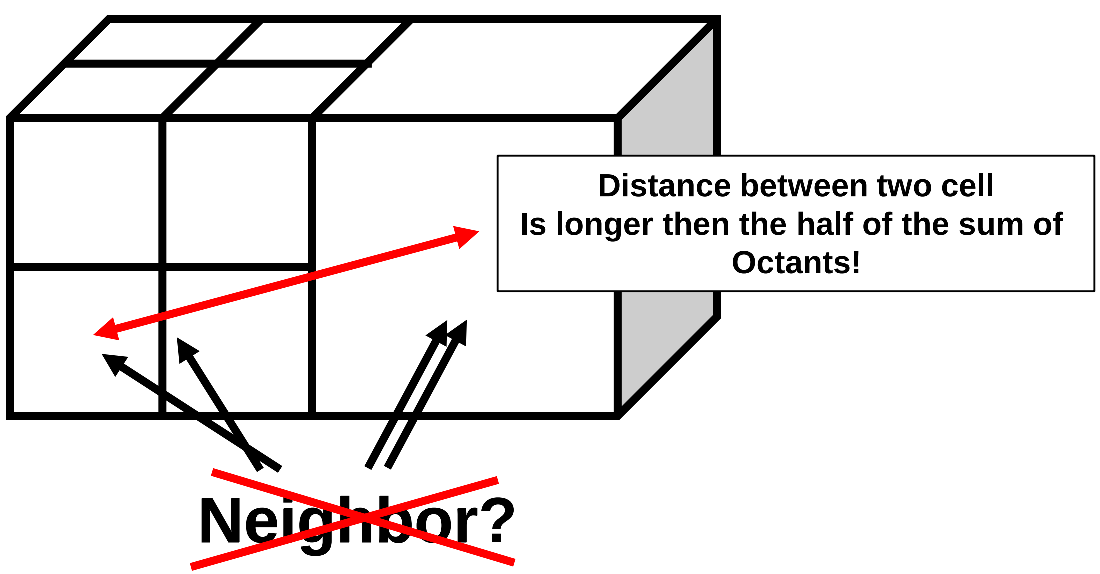
Basic Concepts

- Simplify Quadrotor dynamic
- Reduce resolution of Octomap (octants)

- Octree-Based State Lattice
  - Adjacency between octree node states
  - Multi-resolution path lookup-table
  - Pre-discretization
- Local 3D State Lattice
- Graph search
  - Optimal path finding
  - Path reconstruction

Octree-Based State Lattice

- **Adjacency between octree node states**
  - **To determine whether two octants are adjacent to each other.**
  - **If distance between two cell's center exceeds half of the sum of two octants' cell size, two octants are not adjacent.**



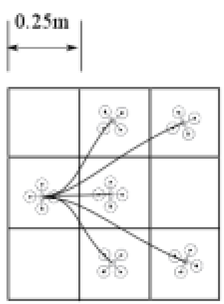
Algorithm 1 Algorithm to find neighbors for each of a *node*'s children when the node is split.

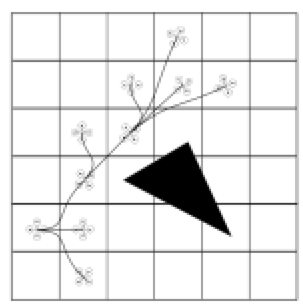
if *node* is no longer a leaf due to updated map information **then**
  Split *node* into eight children (child[i], $i \in 1, 2, .., 8$)
  for $i = 1 \to 8$ **do**
    child-node = child[i]
    add the *child-node*'s brothers (child[j], $j \neq i$, $j \in 1, 2, ...8$) as neighbors
    for each neighbor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

KAIST

Octree-Based State Lattice

- **Multi-resolution path lookup-table**
  - **Computing path between every octants' consume much computational cost.**
  - **How about save all pre-computed cost and path in the table?**
  - **They set 16 states in yaw angle (22.5 deg inc)**
  - **They set lookup index (θ_1,x_1-x_2,y_1-y_2,z_1-z_2, θ_2)**





Algorithm 2 Multi-resolution path lookup-table construction.

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

Octree-Based State Lattice

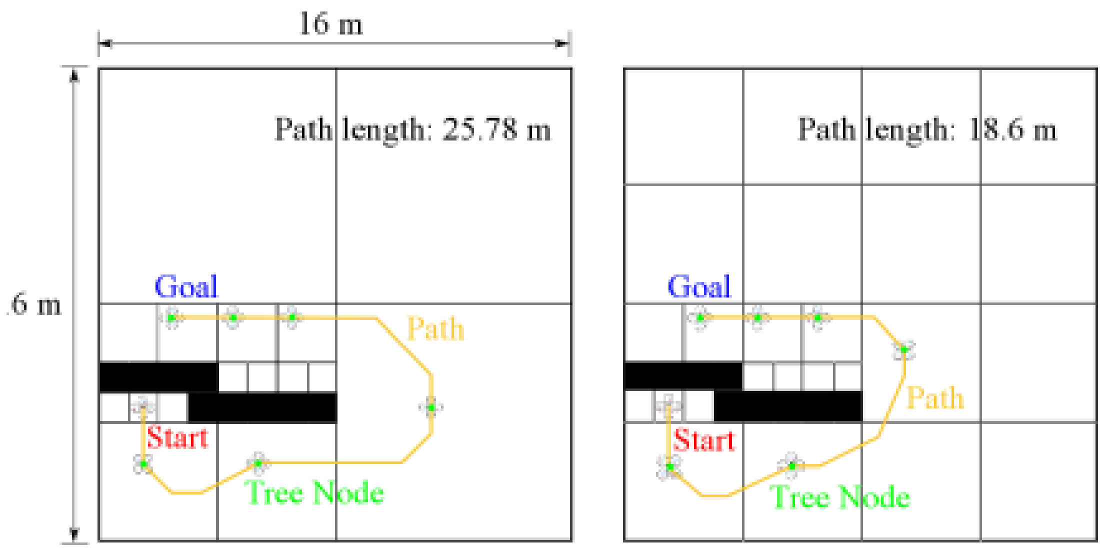
- **Multi-resolution path lookup-table**
  - **But save all computation result consume lots of memory!**
  - **They just consider 'distance' as cost.**
  - **So, $(0,0,0,\theta_1)$ to (x,y,z,θ_2) can be reflected to $(0,0,0,\theta_1)$ to $(x,y,-z,\theta_2)$!**
  - **Also, all 16 possible θ_1 can bereduced to 0,22.5,45 degrees.**
  - **So, they say memory requirement reduced by 90%**

Algorithm 2 Multi-resolution path lookup-table construction.

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

Octree-Based State Lattice

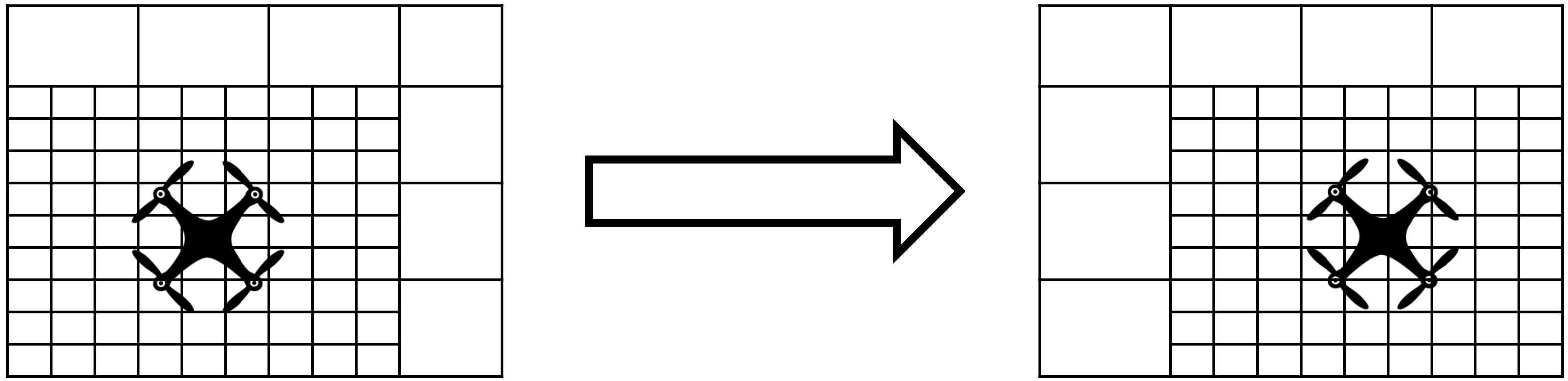
- **Pre-discretization**
  - **Octree-based state lattice may compute highly suboptimal path.**
  - **More octree level means large pre-computed cost and path table.**
  - **So they enforce a minimum octree level on all leaf node.**



(a) (b)

Local 3D State Lattice

- Path planning is critical especially for obstacle avoidance.
- They make local high-resolution state lattice centered on the MAV.



- These method can maintain octree-based graph structure.
- Can help the MAV navigate around nearby obstacles.

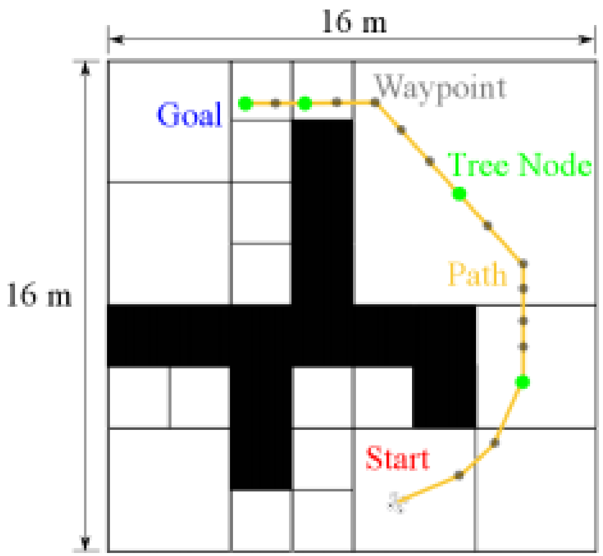
Graph Search

- **Optimal Path Finding**
  - **Use simple A* graph search algorithm. (Using the method above, any A* based algorithm can be used)**
  - **A* algorithm heavily depends on the quality of the heuristic function.**
    - **They applied holonomic-with-obstacles heuristic [1]**
      - Ignores the non-holonomic nature of robot, and then make 2D path with obstacle map
      - 3D space into 2D space by $f_2(x, y) = \min_\theta f_3(x, y, \theta)$. which means that 2D state is assumed to be safe (no collision) if there exists at least one safe 3D state with same 2D projection.
    - **They reduced candidate states, so A* able to find the best path in short time.**

[1] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel, Path Planning for Autonomous Vehicles in Unknown Semistructured Environments. *The International Journal of Robotics Research, April. 2010*.

Graph Search

- **Path reconstruction**
  - **Path obtained by A* is actually a series of high-resolution primitive motion.**
  - **They look up the path decompositions in the multiresolution lookup table.**



Green dot : Node achieved by A*
Grey dot : actual waypoints
Clay line : final full path

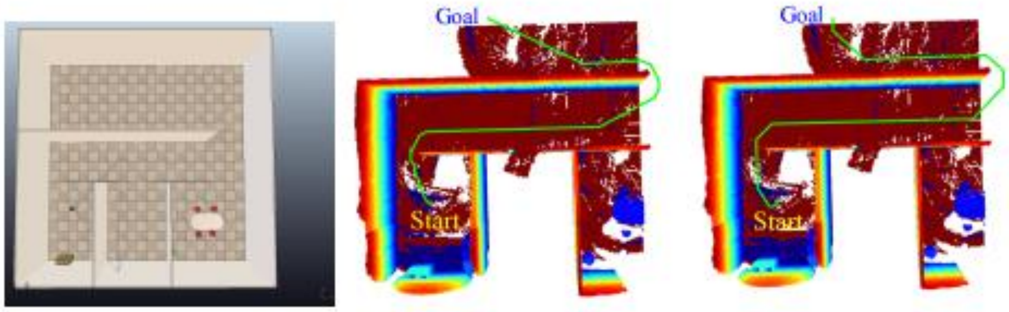
Result

- **Time & Memory usage reduce**
  - **For 50 different goals with maximum resolution of 0.25m**
  - **Compare with regular-state-lattice-based path planner.**

TABLE I: Statistical Results from Simulation Experiments.

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

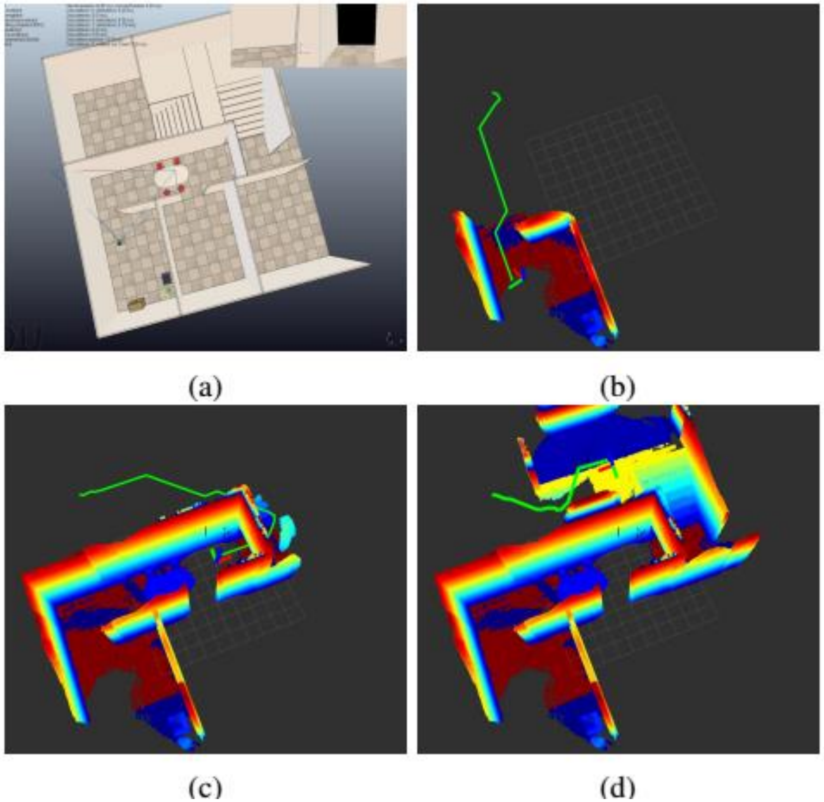
[1] the time taken to update obstacle information and construct graph
[2] the time taken to run A* algorithm on the given graph
[3] the time taken to compute heuristics



(a) (b) (c)

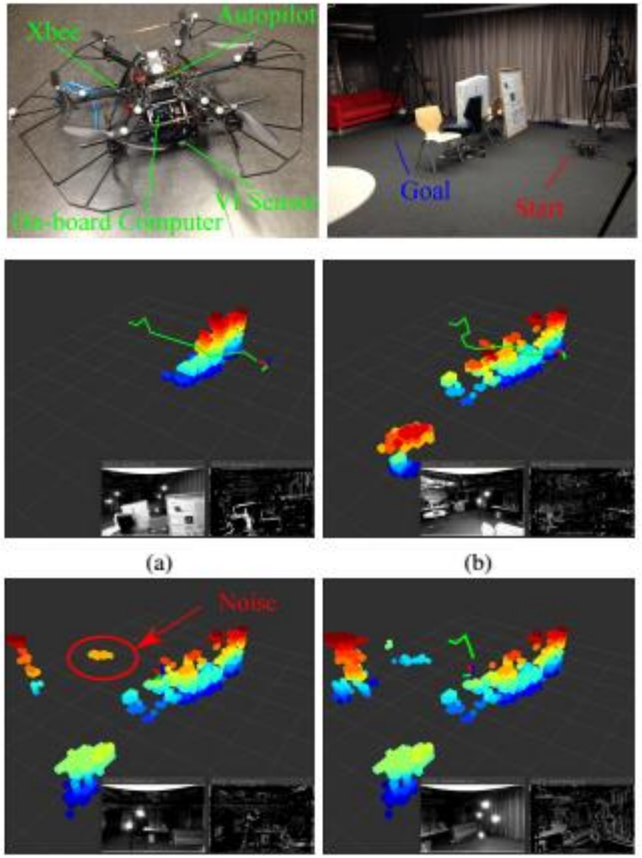
Result

- **Unknown Environment**
  - **A : Entire Environment**
  - **B : Initial Search to goal**
  - **C : UAV goes through Stairs**
  - **D : Successfully find path**



Result

- **Real Environment**
  - **Also in real environment, Algorithm works well.**
  - **UAV found obstacle, and planned path.**



ANY QUESTION?